z/OS
2.5

Cryptographic Services System Secure Sockets Layer Programming

SC14-7495-50

Note

Before using this information and the product it supports, read the information in “Notices” on page 813.

This edition applies to Version 2 Release 5 of z/OS® (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2023-06-26

Contents

Figures

Tables

About this document

This information contains information about the system Secure Sockets Layer (SSL) component of the z/OS Cryptographic Services element. This information consists of primarily two sets of APIs and a Certificate Management utility. The first set of APIs support the Secure Sockets Layer protocols (SSL V2.0, SSL 3.0, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3) which can be used by C/C++ applications to communicate securely across an open communications network. The other set of APIs (Certificate Management) provide the ability to use function other than the SSL protocols. These functions include the ability to create/manage key database files in a similar function to the SSL Certificate Management utility, use certificates stored in a key database file, SAF key ring or z/OS PKCS #11 token for purposes other than SSL and basic PKCS #7 message support to provide application writers a mechanism to communicate with another application through the PKCS #7 standard.

This information also provides guidance on how to write a client and server secure sockets layer application. The client and server may both reside on z/OS™ systems or reside on different systems.

Who should use this document

This document is intended to assist system administrators in setting up the system to use System SSL support and for application programmers in writing System SSL applications.

How to use this document

The format and organization of this information:

Chapter 1, “Introduction,” on page 1 describes Secure Sockets Layer (SSL) and lists the software dependencies and installation information you need to use the System SSL support.

Chapter 2, “How System SSL works for secure socket communication,” on page 5 provides a general overview of System SSL and the basic structure of a z/OS application using System SSL.

Chapter 3, “Using cryptographic features with System SSL,” on page 9 describes System SSLs use of cryptographic features on z/OS.

Chapter 4, “System SSL and FIPS 140-2,” on page 19 describes how to execute System SSL securely in a mode designed to meet FIPS 140-2 criteria.

Chapter 5, “Writing and building a z/OS System SSL application,” on page 29 describes how to write a System SSL source program and build the System SSL application.

Chapter 6, “System SSL application programming considerations,” on page 37 describes the considerations to think about when designing a System SSL application as well as considerations when upgrading to the TLS V1.2 and TLS V1.3 protocols.

Chapter 7, “API reference,” on page 73 describes the System SSL program interfaces.

Chapter 8, “Certificate Management Services (CMS) API reference,” on page 183 describes the Certificate Management Services (CMS) program interfaces.

Chapter 9, “Deprecated Secure Socket Layer (SSL) APIs,” on page 513 describes the deprecated System SSL program interfaces.

Chapter 10, “Certificate/Key management,” on page 541 describes how to use the **gskkyman** utility to create a key database file, a z/OS PKCS #11 token, a public/private key pair, a certificate request, and other tasks.

Chapter 11, “SSL started task,” on page 611 provides sysplex session cache support, sysplex session ticket cache support, and dynamic trace support

Chapter 12, “Obtaining diagnostic information,” on page 623 provides debugging information.

Chapter 13, “Messages and codes,” on page 629 contains various messages and codes you might encounter using System SSL.

Appendix A, “Environment variables,” on page 763 lists the environment variables used by System SSL.

Appendix B, “Sample C++ SSL files,” on page 791 describes the sample set of files shipped to provide an example of what is needed to build a C++ System SSL application.

Appendix C, “Cipher suite definitions,” on page 795 describes supported cipher suite definitions.

Appendix D, “Object identifiers,” on page 807 describes object identifiers (OIDS) supported by System SSL.

Appendix E, “Migrating from deprecated SSL interfaces,” on page 809 describes how to migrate an existing application which uses the deprecated SSL interfaces to the latest SSL interfaces.

Conventions used in this information

This information uses these typographic conventions:

Bold
: **Bold** words or characters

Highlighting1
: Words or characters **highlighted** in this manner represent system elements that you must enter into the system literally, such as commands, options, or path names.

Italic
: *Italic* words or characters

Highlighting2
: Words or characters *highlighted* in this manner represent values for variables that you must supply.

`Example font`
: Examples and information displayed by the system appear in `constant width type style`.

[ ]
: Brackets enclose optional items in format and syntax descriptions.

{ }
: Braces enclose a list from which you must choose an item in format and syntax descriptions.

|
: A vertical bar separates items in a list of choices.

< >
: Angle brackets enclose the name of a key on the keyboard.

...
: Horizontal ellipsis points indicate that you can repeat the preceding item one or more times.

\
: A backslash is used as a continuation character when entering commands from the shell that exceed one line (255 characters). If the command exceeds one line, use the backslash character \ as the last non blank character on the line to be continued, and continue the command on the next line.

This information uses these keying conventions:

<ALT-c>
: The notation <`Alt-c`> followed by the name of a key indicates a control character sequence.

<Return>
: The notation <`Return`> refers to the key on your keyboard that is labeled with the word Return or Enter, or with a left arrow.

Entering commands
: When instructed to enter a command, type the command name and then press <`Return`>.

Where to find more information

When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead "If you have a technical problem" on page xxi.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function
If your comment or question is about z/OS itself, submit a request through the IBM RFE Community (www.ibm.com/developerworks/rfe/).

Feedback on IBM® Documentation function
If your comment or question is about the IBM Documentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation Support at ibmdoc@us.ibm.com.

Feedback on the z/OS product documentation and content
If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS System SSL Programming, SC14-7495-50
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive authority to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Note: IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under IBM z/OS Product Documentation Update Policy (www-01.ibm.com/servers/resourcelink/svc00100.nsf/pages/ibm-zos-doc-update-policy?OpenDocument).

Summary of changes for z/OS Version 2 Release 5 (V2R5)

The following changes are made for z/OS Version 2 Release 5 (V2R5). The most recent updates are listed at the top of each section.

New

June 2023 refresh

- Added for APAR OA63164:
  - “Server certificate domain-based validation” on page 70.
  - “SSL function return codes” on page 629: 548.
- Added for APAR OA63252:
  - “Interpreting the session cache statistics” on page 615.
  - “Sysplex session ticket cache support” on page 618.
  - “Interpreting the session ticket cache statistics” on page 619.
  - The following SSL function return codes are added (See “SSL function return codes” on page 629): 547.
  - The following SSL started task messages (GSK01nnn) are added (See “SSL started task messages (GSK01nnn)” on page 741): GSK01065E, GSK01066E, GSK01067E, GSK01068E, GSK01069E, GSK01070E, GSK01071E, GSK01072E, GSK01073E, GSK01074E, GSK01075E, GSK01076E, GSK01077I, GSK01078I, GSK01079I, GSK01080I, GSK01081I, GSK01082I, GSK01083E, GSK01084E.

October 2022 refresh

Information about IBM z16 and Crypto Express8 adapter (CEX8C, CEX8P, and CEX8A).

June 2022 refresh

“Limiting key exchange elliptic curves” on page 69 (APAR OA61783, which also applies to z/OS V2R4).

Prior to the June 2022 refresh

- “Certificate diagnostic callback routine” on page 41.
- The following Certificate Management Services (CMS) APIs are new:
  - “gsk_format_long_integer()” on page 325.
  - “gsk_free_x509_diagnostics()” on page 351.
- The following SSL function return codes are added (See “SSL function return codes” on page 629): 546.
- The following deprecated SSL function return codes are added (See “Deprecated SSL function return codes” on page 673): -131.

- The following CMS status codes (03353xxx) are added (See "CMS status codes (03353xxx)" on page 700): 033530C3, 033530C4.

Changed

June 2023 refresh

- Updated for APAR OA63164:
  - Chapter 6, "System SSL application programming considerations," on page 37.
  - "gsk_attribute_get_buffer()" on page 76.
  - "gsk_attribute_get_enum()" on page 87.
  - "gsk_attribute_set_buffer()" on page 100.
  - "gsk_attribute_set_enum()" on page 112.
  - "gsk_environment_open()" on page 136.
  - "gsk_secure_socket_init()" on page 157.
  - Appendix A, "Environment variables," on page 763.
- Updated for APAR OA63252:
  - Updated "Session ID (SID) and session ticket cache" on page 42.
  - Updated "Session ID (SID) and session tickets" on page 43.
  - Updated "gsk_attribute_get_enum()" on page 87.
  - Updated "gsk_attribute_get_numeric_value()" on page 97.
  - Updated "gsk_attribute_set_enum()" on page 112.
  - Updated "gsk_attribute_set_numeric_value()" on page 124.
  - Updated "gsk_environment_open()" on page 136.
  - Updated "gsk_secure_socket_misc()" on page 170.
  - Updated "gsk_initialize()" on page 517.
  - Updated Chapter 11, "SSL started task," on page 611.
  - Updated "GSKSRVR environment variables" on page 611.
  - Updated "Server operator commands" on page 613.
  - Updated "Sysplex session cache support" on page 615.
  - Updated Appendix A, "Environment variables," on page 763.
  - The following SSL function return codes are updated (See "SSL function return codes" on page 629): 477.
  - The following SSL started task messages (GSK01nnn) are modified (See "SSL started task messages (GSK01nnn)" on page 741): GSK01005E, GSK01006E.

February 2023

- Updated "Certificate diagnostic callback routine" on page 41.
- Updated "gsk_attribute_set_callback()" on page 107.
- Updated "gsk_attribute_set_enum()" on page 112.
- The following SSL function return codes are modified for APAR OA63632 (See "SSL function return codes" on page 629):
  - 402, 542.

October 2022 refresh

Updated Appendix A, "Environment variables," on page 763 for APAR OA61783, which also applies to z/OS V2R4.

June 2022 refresh

Updated for APAR OA61783, which also applies to z/OS V2R4:

- Chapter 6, "System SSL application programming considerations," on page 37.
- "gsk_attribute_get_buffer()" on page 76.
- "gsk_attribute_set_buffer()" on page 100.
- "gsk_environment_open()" on page 136.
- "gsk_secure_socket_init()" on page 157.
- The following SSL function return codes are modified (See "SSL function return codes" on page 629):
  - 9, 402, 405, 464, 517.
- Appendix A, "Environment variables," on page 763.
- Table 29 on page 804.

Prior to the June 2022 refresh

- Chapter 6, "System SSL application programming considerations," on page 37.
- "Use of user data" on page 41.
- The following SSL APIs are modified:
  - "gsk_attribute_get_enum()" on page 87.
  - "gsk_attribute_set_callback()" on page 107.
  - "gsk_attribute_set_enum()" on page 112.
  - "gsk_environment_open()" on page 136.
  - "gsk_secure_socket_init()" on page 157.
- The following Certificate Management Services (CMS) APIs are modified:
  - "gsk_get_cms_vector()" on page 362.
  - "gsk_validate_certificate_mode()" on page 491.
- The following Deprecated Secure Socket Layer (SSL) APIs are modified:
  - "gsk_secure_soc_init()" on page 523.
- The following SSL function return codes are modified (See "SSL function return codes" on page 629): 507.
- Appendix A, "Environment variables," on page 763.

Deleted

No content was removed from this information.

Summary of changes for z/OS Version 2 Release 4 (V2R4)

The following changes are made for z/OS Version 2 Release 4 (V2R4). The most recent updates are listed at the top of each section.

New

June 2021 refresh

- The following SSL function return codes are added (See "SSL function return codes" on page 629): 544 and 545
- The following deprecated SSL function return codes are added (See "Deprecated SSL function return codes" on page 673): -129 and -130

Prior to the April 2021 refresh

- “RSASSA-PSS signature support” on page 14
- “TLS V1.3 protocol support” on page 61
- “Upgrading to TLS V1.2 from earlier SSL and TLS protocols” on page 64
- “Upgrading from TLS V1.2 to TLS V1.2 and TLS V1.3 protocols” on page 67
- The following SSL function return codes are added (See “SSL function return codes” on page 629):
  - 441
  - 514
  - 515
  - 516
  - 517
  - 518
  - 519
  - 520
  - 521
  - 522
  - 523
  - 524
  - 525
  - 526
  - 527
  - 528
  - 529
  - 530
  - 531
  - 532
  - 533
  - 534
  - 535
  - 536
  - 537
  - 538
  - 539
  - 540
  - 541
  - 542
  - 543
  - 605
- The following CMS status codes (03353xxx) are added (See “CMS status codes (03353xxx)” on page 700):
  - 033530BF
  - 033530C0
  - 033530C1

  - 033530C2

Changed

June 2021 refresh

- "Overview of hardware cryptographic features and System SSL" on page 10
- Chapter 7, "API reference," on page 73
- The following SSL APIs are modified:
  - "gsk_attribute_get_enum()" on page 87
  - "gsk_attribute_set_enum()" on page 112
  - "gsk_environment_open()" on page 136
  - "gsk_secure_socket_init()" on page 157
  - "gsk_secure_socket_read()" on page 174
- The following Deprecated Secure Socket Layer (SSL) APIs are modified:
  - "gsk_initialize()" on page 517
  - "gsk_secure_soc_init()" on page 523
- The following SSL function return code was modified (See "SSL function return codes" on page 629): 402
- Appendix A, "Environment variables," on page 763

April 2021 refresh

- "Overview of hardware cryptographic features and System SSL" on page 10
- "Algorithms and key sizes" on page 19
- "Application changes" on page 27
- "gsk_fips_state_set()" on page 323
- The following SSL function return codes are modified (See "SSL function return codes" on page 629):
  - 6, 7, 8, 109, 407, 416, 417, 435, 522.
- The following deprecated SSL function return codes are modified (See "Deprecated SSL function return codes" on page 673):
  - 9, 12, 18, 19, -15, -18, -19, -35.
- The following CMS status codes are modified (See "CMS status codes (03353xxx)" on page 700):
  - 0335300E, 03353024, 0335302F, 03353040.

Prior to the April 2021 refresh

- Chapter 1, "Introduction," on page 1
- "RACF CSFSERV resource requirements" on page 15.
- "Algorithms and key sizes" on page 19.
- "Create an SSL environment" on page 29
- Chapter 6, "System SSL application programming considerations," on page 37
- "Session ID (SID) and session ticket cache" on page 42
- "Suite B cryptography support" on page 49
- "Enabling OCSP server stapling" on page 57
- "Using server multiple key label support" on page 59
- Chapter 7, "API reference," on page 73
- The following SSL APIs are modified:

  - "gsk_attribute_get_buffer()" on page 76
  - "gsk_attribute_get_data()" on page 85
  - "gsk_attribute_get_enum()" on page 87
  - "gsk_attribute_get_numeric_value()" on page 97
  - "gsk_attribute_set_buffer()" on page 100
  - "gsk_attribute_set_callback()" on page 107
  - "gsk_attribute_set_enum()" on page 112
  - "gsk_attribute_set_numeric_value()" on page 124
  - "gsk_attribute_set_tls_extension()" on page 130
  - "gsk_environment_init()" on page 134
  - "gsk_environment_open()" on page 136
  - "gsk_secure_socket_init()" on page 157
  - "gsk_secure_socket_misc()" on page 170
- The following Certificate Management Services (CMS) APIs are modified:
  - "gsk_add_record()" on page 188
  - "gsk_construct_certificate()" on page 196
  - "gsk_construct_self_signed_certificate()" on page 209
  - "gsk_construct_signed_certificate()" on page 212
  - "gsk_create_database_signed_certificate()" on page 240
  - "gsk_create_self_signed_certificate()" on page 255
  - "gsk_create_signed_certificate()" on page 259
  - "gsk_create_signed_certificate_record()" on page 262
  - "gsk_create_signed_certificate_set()" on page 266
  - "gsk_create_signed_crl()" on page 271
  - "gsk_create_signed_crl_record()" on page 274
  - "gsk_format_time()" on page 326
  - "gsk_import_certificate()" on page 381
  - "gsk_import_key()" on page 384
  - "gsk_make_signed_data_content_extended()" on page 411
  - "gsk_make_signed_data_msg_extended()" on page 418
  - "gsk_mktime()" on page 423
  - "gsk_read_signed_data_content()" on page 455
  - "gsk_read_signed_data_content_extended()" on page 458
  - "gsk_read_signed_data_msg()" on page 461
  - "gsk_read_signed_data_msg_extended()" on page 464
  - "gsk_replace_record()" on page 469
  - "gsk_sign_certificate()" on page 477
  - "gsk_sign_crl()" on page 480
  - "gsk_sign_data()" on page 483
  - "gsk_validate_certificate()" on page 486
  - "gsk_validate_certificate_mode()" on page 491
  - "gsk_verify_certificate_signature()" on page 503
  - "gsk_verify_crl_signature()" on page 506

  - “gsk_verify_data_signature()” on page 509
- “gskkyman interactive mode descriptions” on page 546:
  - “Database menu” on page 547
  - “Create a self-signed certificate” on page 555
- “Creating a self-signed server or client certificate” on page 567
- “Creating a signed certificate and key” on page 587
- “Creating a certificate to be used with a fixed Diffie-Hellman key exchange” on page 591
- “Using gskkyman to be your own certificate authority (CA)” on page 598
- “gskkyman” on page 601
- The following SSL function return codes are modified (See “SSL function return codes” on page 629):
  - 8
  - 402
  - 405
  - 414
  - 466
  - 467
  - 476
  - 477
  - 478
  - 479
  - 480
  - 508
  - 510
  - 512
  - 601
- The following CMS status codes (03353*xxx*) are modified (See “CMS status codes (03353xxx)” on page 700):
  - 03353030
  - 03353034
  - 03353088
- Appendix A, “Environment variables,” on page 763
- Appendix B, “Sample C++ SSL files,” on page 791
- Appendix C, “Cipher suite definitions,” on page 795
- Appendix D, “Object identifiers,” on page 807

Deleted

No content was removed from this information.

Summary of changes for z/OS Version 2 Release 3 (V2R3)

The following changes are made for z/OS Version 2 Release 3 (V2R3). The most recent updates are listed at the top of each section.

New

- "Using server multiple key label support" on page 59
- "Enabling OCSP server stapling" on page 57
- "gsk_construct_signed_crl()" on page 216
- "gsk_format_time()" on page 326
- "Displaying a newly created certificate request" on page 574
- The following SSL function return codes are added (See "SSL function return codes" on page 629):
  - 496
  - 497
  - 498
  - 499
  - 500
  - 507
  - 508
  - 509
  - 510
  - 511
  - 512
  - 513
- The following deprecated SSL function return codes are added (See "Deprecated SSL function return codes" on page 673):
  - -126
  - -127
  - -128
- The following CMS status codes (03353xxx) are added (See "CMS status codes (03353xxx)" on page 700):
  - 033530B3
  - 033530B4
  - 033530B5
  - 033530B6
  - 033530B7
  - 033530B8
  - 033530B9
  - 033530BA
  - 033530BB
  - 033530BC
  - 033530BD
  - 033530BE
- New environment variables have been added to Appendix A, "Environment variables," on page 763.

Changed

- “gsk_strerror()” on page 181
- Chapter 13, “Messages and codes,” on page 629
- “Installation information” on page 1 (APAR OA54127 Allow strong SSL/TLS ciphers in non-FIPS mode to be used when CPACF feature 3863 is installed)
- “gsk_environment_open()” on page 136 (APAR OA54127 Allow strong SSL/TLS ciphers in non-FIPS mode to be used when CPACF feature 3863 is installed)
- “gsk_make_signed_data_content_extended()” on page 411 (APAR OA54821 Support creation of PKCS #7 SignedData detached signature messages)
- “gsk_make_signed_data_msg_extended()” on page 418 (APAR OA54821 Support creation of PKCS #7 SignedData detached signature messages)
- “gsk_get_cipher_info()” on page 515 (APAR OA54127 Allow strong SSL/TLS ciphers in non-FIPS mode to be used when CPACF feature 3863 is installed)
- “gsk_secure_soc_init()” on page 523 (APAR OA54127 Allow strong SSL/TLS ciphers in non-FIPS mode to be used when CPACF feature 3863 is installed)
- Appendix A, “Environment variables,” on page 763 (APAR OA54127 Allow strong SSL/TLS ciphers in non-FIPS mode to be used when CPACF feature 3863 is installed)
- Appendix C, “Cipher suite definitions,” on page 795 (APAR OA54127 Allow strong SSL/TLS ciphers in non-FIPS mode to be used when CPACF feature 3863 is installed)
- “Overview of hardware cryptographic features and System SSL” on page 10
- Chapter 4, “System SSL and FIPS 140-2,” on page 19
- “Algorithms and key sizes” on page 19
- “Diffie-Hellman key agreement” on page 23
- “Certificate stores” on page 26
- “Key database files” on page 26
- “Suite B cryptography support” on page 49
- The following SSL APIs are modified:
  - “gsk_attribute_get_buffer()” on page 76
  - “gsk_attribute_get_enum()” on page 87
  - “gsk_attribute_get_numeric_value()” on page 97
  - “gsk_attribute_set_buffer()” on page 100
  - “gsk_attribute_set_enum()” on page 112
  - “gsk_attribute_set_numeric_value()” on page 124
  - “gsk_environment_init()” on page 134
  - “gsk_environment_open()” on page 136
  - “gsk_secure_socket_init()” on page 157
  - “gsk_secure_socket_read()” on page 174
- The following Certificate Management Services (CMS) APIs are modified:
  - “gsk_add_record()” on page 188
  - “gsk_change_database_password()” on page 191
  - “gsk_construct_certificate()” on page 196
  - “gsk_construct_renewal_request()” on page 206
  - “gsk_construct_self_signed_certificate()” on page 209
  - “gsk_construct_signed_certificate()” on page 212
  - “gsk_create_certification_request()” on page 231

  - “gsk_replace_record()” on page 469
  - “gsk_set_default_key()” on page 472
  - “gsk_sign_certificate()” on page 477
  - “gsk_sign_crl()” on page 480
  - “gsk_sign_data()” on page 483
  - “gsk_validate_certificate()” on page 486
  - “gsk_validate_certificate_mode()” on page 491
  - “gsk_verify_certificate_signature()” on page 503
  - “gsk_verify_data_signature()” on page 509
- The following Deprecated Secure Socket Layer (SSL) APIs are modified:
  - “gsk_initialize()” on page 517
  - “gsk_secure_soc_init()” on page 523
- Chapter 10, “Certificate/Key management,” on page 541
- “Database menu” on page 547 was updated.
- “Starting gskkyman” on page 559 was updated.
- “gskkyman” on page 601 was updated.
- The following SSL function return codes were modified (See “SSL function return codes” on page 629):
  - 8
  - 466
  - 475
- The following deprecated SSL function return code is modified (See “Deprecated SSL function return codes” on page 673):
  - -35
- The following CMS status codes (03353*xxx*) were modified (See “CMS status codes (03353xxx)” on page 700):
  - 03353027
  - 03353034
  - 03353096
  - 03353099
- The following SSL started task messages (GSK01*nnn*) are modified (See “SSL started task messages (GSK01nnn)” on page 741):
  - GSK01051E
  - GSK01052W
- Appendix A, “Environment variables,” on page 763
- Appendix C, “Cipher suite definitions,” on page 795

Deleted

No content was removed from this information.

Chapter 1. Introduction

Secure Sockets Layer (SSL) is a communications protocol that provides secure communications over an open communications network (for example, the Internet). The SSL protocol is a layered protocol that is intended to be used on top of a reliable transport, such as Transmission Control Protocol (TCP/IP). SSL provides data privacy and integrity including server and client authentication that is based on public key certificates. Once an SSL connection is established between a client and server, data communications between client and server are transparent to the encryption and integrity added by the SSL protocol. System SSL supports the SSL V2.0, SSL V3.0 and TLS (Transport Layer Security) V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 protocols. TLS V1.3 is the latest version of the secure sockets layer protocol that is supported by System SSL.

Note: The phrase SSL is used throughout to describe both the SSL and TLS protocols.

z/OS provides a set of SSL C/C++ callable application programming interfaces that, when used with the z/OS Sockets APIs, provide the functions that are required for applications to establish secure sockets communications.

In addition to providing the API interfaces to use the Secure Sockets Layer and Transport Layer Security protocols, System SSL is also providing a suite of Certificate Management APIs. These APIs give the capability to create/manage your own certificate databases, use certificates that are stored in key databases, key rings or tokens for purposes other than SSL and to build/process PKCS #7 standard messages.

In addition to providing APIs for applications to use for both SSL and certificate management support, System SSL also provides a certificate management utility called **gskkyman**. The **gskkyman** utility allows for the management of certificates that are stored in a key database file or z/OS PKCS #11 token.

System SSL is designed to meet the Federal Information Processing Standard - FIPS 140-2 criteria. See Chapter 4, “System SSL and FIPS 140-2,” on page 19 for more information.

Software dependencies

Cryptographic Services System SSL (FMID HCPT450)
System SSL Version 2 Release 5 is part of the Cryptographic Services Base element of z/OS. (The System SSL Base members are installed in the PDSE *pdsename*.SIEALNKE and the PDS *pdsname*.SGSKSAMP.)

Cryptographic Services Security Level 3 (FMID JCPT451)
When you order the Cryptographic Services Security Level 3 support, GSKSUS31, GSKSUS64, GSKC31F, GSKC64F, GSKS31F, and GSKS64F are installed as members of the *pdsename*.SIEALNKE PDSE. *pdsename*.SIEALNKE is the PDSE in which the System SSL Cryptographic Services Base members are installed.

Japanese (FMID JCPT45J)
Contains Japanese message text files for the **gskkyman** utility. The gskmsgs.cat file is installed in the /usr/lpp/gskssl/lib/nls/msg/Ja_JP.IBM-939 directory.

Appendix C, “Cipher suite definitions,” on page 795 provides information about the encryption capabilities (ciphers) for each protocol and FMID.

Installation information

System SSL is part of the System SSL Cryptographic Services Base element of z/OS. If you choose to install the z/OS Version 2 Release 5 Server Pack, you do not need to install the System SSL Cryptographic Services Base element separately. If you choose the z/OS Custom-Build Product Delivery Offering (CBPDO), you can install the System SSL Cryptographic Services Base element using SMP/E. The *z/OS Program Directory* contains the directions for installing the System SSL Cryptographic Services Base element using SMP/E.

The System SSL base FMID HCPT450 provides cryptographic support up to 56-bit in strength. In order to exploit cryptographic support greater than 56-bit (for example, 3DES, AES), either the System SSL Security Level 3 FMID JCPT451 must be ordered and installed or the CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement Feature 3863 must be installed. For more information about installing the Security Level 3 FMID, see the *z/OS Program Directory*.

System SSL parts shipped in the UNIX System Services file system

- /usr/lpp/gskssl/include

  Contains the header files, **gskssl.h**, **gsktypes.h** and **gskcms.h**, which declare structures and constants that are used by the System SSL and Certificate Management interfaces.
- /usr/lpp/gskssl/examples

  Contains sample client/server files including a display_certificate sample program.
- /usr/lpp/gskssl/lib

  Contains GSKSSL.x for APIs exported by the GSKSSL DLL, GSKSSL64.x for APIs exported by the GSKSSL64 DLL, GSKCMS31.x for APIs exported by the GSKCMS31 DLL, and GSKCMS64.x for APIs exported by the GSKCMS64 DLL. You use GSKSSL.x and GSKCMS31.x when you linkedit a 31-bit program that uses System SSL and you use GSKSSL64.x and GSKCMS64.x when you linkedit a 64-bit program that uses System SSL.
- /usr/lpp/gskssl/lib/nls/msg/En_US.IBM-1047

  Contains the English gskmsgs.cat message catalog file.
- /usr/lpp/gskssl/lib/nls/msg/Ja_JP.IBM-939

  Contains the Kanji gskmsgs.cat message catalog file.
- /usr/lpp/gskssl/bin

  Contains the **gskkyman** and **gsktrace** utilities.

System SSL parts shipped in PDS and PDSE

pdsename.**SIEALNKE** PDSE contains members **GSKSSL**, **GSKCMS31**, **GSKSRBRD**, **GSKSRBWT**, **GSKKYMAN**, **GSKSCTSS**, **GSKSRVR**, **GSKCMS64**, **GSKS31**, **GSKS64**, **GSKC31**, **GSKC64** and **GSKSSL64** when the base FMID HCPT450 is installed. When FMID JCPT451 is installed, members **GSKSUS31**, **GSKS31F**, **GSKS64F**, **GSKC31F**, **GSKC64F** and **GSKSUS64** are also in the PDSE.

pdsname.**SIEAHDR** PDS contains header files **GSKSSL**, **GSKCMS** and **GSKTYPES**.

pdsename.**SIEASID** PDS contains side files **GSKSSL**, **GSKCMS31**, **GSKSSL64 and GSKCMS64** when the base FMID HCPT450 is installed.

pdsname.**SGSKSAMP** PDS contains members **GSKMSGXT**, **GSKRACF**, **GSKSRVR** and **GSKWTR**.

pdsename.**SIEAMIGE** PDS contains member **GSKSCTFT**.

pdsname.**LPALIB** PDS contains member **GSKEOMRM** when the base FMID HCPT450 is installed.

pdsname and *pdsename* are the names determined during installation. You need to know the name of this PDS or PDSE when you identify the STEPLIB in the runtime steps. See *z/OS Program Directory* for information about installing the System SSL.

Note:

1. The DLLs are shipped in PDSE form so the DLLs can be called from UNIX System Services file-system-based or PDSE-based programs.
2. The DLLs are not placed in SYS1.LPALIB during installation. The DLLs cannot be added to an LPALSTxx member since PDSE data sets are not supported in LPALSTxx. The DLLs can be added to the dynamic LPA by adding them to a PROGxx member.
3. The DLLs cannot be added to the LPA if System SSL is to be used in FIPS mode.

System SSL is designed to meet the National Institute of Standards and Technology (NIST) FIPS 140-2 criteria. For more information about enabling applications and running System SSL FIPS enabled applications, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

Chapter 2. How System SSL works for secure socket communication

System SSL supports both the TLS (Transport Layer Security) and SSL (Secure Sockets Layer) protocols. Before you start writing your application, let's look at how System SSL works.

Note: The phrase *SSL* is used throughout to describe both the SSL and TLS protocols.

The SSL protocol begins with a "handshake". During the handshake, the client authenticates the server, the server optionally authenticates the client, and the client and server agree on how to encrypt and decrypt information. In addition to the "handshake", SSL also defines the format that is used to transmit encrypted data.

X.509 (V1, V2, or V3) certificates are used by both the client and server when securing communications using System SSL. The client must verify the server's certificate based on the certificate of the Certificate Authority (CA) that signed the certificate or based on a self-signed certificate from the server. The server must verify the client's certificate (if requested) using the certificate of the CA that signed the client's certificate. The client and the server then use the negotiated session keys and begin encrypted communications.

The SSL protocol runs above the TCP/IP and below higher-level protocols such as HTTP. It uses TCP/IP on behalf of the higher-level protocols.

The capabilities of SSL address several fundamental concerns about communication over the Internet and other TCP/IP networks:

SSL server authentication allows a client application to confirm the identity of the server application. The client application through SSL uses standard public-key cryptography to verify that the server's certificate and public key are valid and are signed by a trusted certificate authority (CA) that is known to the client application.

SSL client authentication allows a server application to confirm the identity of the client application. The server application through SSL uses standard public-key cryptography to verify that the client's certificate and public key are valid and are signed by a trusted certificate authority (CA) that is known to the server application.

An encrypted SSL connection requires all information that is sent between the client and server application to be encrypted. The sending application is responsible for encrypting the data and the receiving application is responsible for decrypting the data. In addition to encrypting the data, SSL provides message integrity. Message integrity provides a means to determine if the data has been tampered with since it was sent by the partner application.

Using System SSL on z/OS

System SSL provides programming interfaces to write both client and server applications. These programming interfaces provide functionality that is associated with either the SSL environment layer or secure socket connection layer. The SSL environment layer defines the general attributes of the environment, such as the key database file name, stash file name and session timeout. The secure socket connection layer defines the attributes that are associated with each secure connection being established, such as the file descriptor and certificate label. The SSL application program must first create the SSL environment layer. Once the environment is created, one or more instances of the secure socket connection layer can be associated with the SSL environment. Each of these secure socket connections can be established and closed independently of each other.

Each layer has four general function calls:

- open
- attribute_set

- initialize
- close

In addition, the secure socket connection layer has read and write function calls for reading and writing secure data between the two SSL enabled applications.

The open function calls return a handle (an environment handle or a secure socket connection handle) that must be passed back as a parameter on subsequent function calls. An instance of a secure socket connection handle is associated with an environment by passing the environment handle as a parameter on the **gsk_secure_socket_open()** call. The **gsk_secure_socket_open()** function is completely thread safe. Invocations to the **gsk_secure_socket open()** function can be issued from different threads within an environment. Read and write functions are full-duplex, so asynchronous read and write function calls can be performed from different threads for a given secure socket connection. However, there can only be one read and one write call in progress at one time for any secure socket connection handle.

For every *open*, there must be a corresponding *close*.

In addition to these functions, various **gsk_attribute_set ...()** and **gsk_attribute_get...()** functions exist to define and retrieve attributes values associated with either the environment or secure socket connection layers. The syntax of these function calls is the same for both the environment and the secure socket connection layers. The target for the set/get function is determined by the handle specified on the function call.

System SSL application overview

Figure 1 on page 8 describes the basic structure of the elements that are needed in your System SSL source program.

Whether writing a server or client applications, the initial steps are the same. First, an SSL environment must be established with these function calls:

gsk_environment_open()
: This is the first function call. It returns an environment handle that is used in all subsequent function calls. It also obtains storage and sets default values for all internal variables and picks up the values that are specified in system environment variables that override the built-in defaults.

gsk_attribute_set...()
: One or more of these function calls are issued to set attribute values for the environment.

gsk_environment_init()
: After you set all variables, issue this function call to complete the initialization of the SSL environment. When complete, you can open and close SSL connections.

Now, the client and server sides diverge. The server side sets up a listen environment. The listen environment is established by obtaining a socket descriptor through the **socket()** call and the activation of a connection through the **bind()**, **listen()** and **accept()** socket calls.When the listen environment is established, the server waits for notification that a secure socket connection is requested and issues these System SSL API function calls:

gsk_secure_socket_open()
: This function call reserves a handle in which to store information for initializing each secure socket. Default values for each SSL connection are set from the environment.

gsk_attribute_set...()
: This function call sets attribute values for this particular SSL connection. These values could include the socket file descriptor, ciphers, protocol, and application-supplied callback routines.

gsk_secure_socket_init()
: For each connection to be started, the application must issue this function call to complete the initialization of the SSL connection and to run the SSL handshake protocol. The SSL handshake is a function of the System SSL support.

gsk_secure_socket_read()
One or more read function calls is issued until the inbound data flow is complete. The number of calls is purely application-dependent.

gsk_secure_socket_write()
One or more write function calls is issued until all appropriate data is sent to the partner. Reads and writes may be alternated as defined by the application protocol until the data flow is complete.

gsk_secure_socket_close()
This function call frees all the resources that are used for the SSL connection.

All of the SSL API function calls are thread-safe. This is useful on the server side, since each connection can be run on its own thread, simplifying application design. See the sample client/server program that is shipped with z/OS System SSL, for an illustration of a multi-threaded application.

The client application then opens a connection to the server through the **socket()** and **connect()** calls and issues these System SSL API function calls:

gsk_secure_socket_open()
This function call reserves a handle in which to store information for initializing each secure socket.

gsk_attribute_set...()
This function call sets values for this particular SSL connection. These values could include the socket file descriptor, ciphers, protocol, and application-supplied callback routines.

gsk_secure_socket_init()
For each connection to be started, the application must issue this function call to complete the initialization of the SSL connection and to run the SSL handshake protocol. The SSL handshake is a function of the System SSL support.

gsk_secure_socket_write()
One or more write function calls are issued until the outbound data flow is complete. The number of calls is purely application-dependent.

gsk_secure_socket_read()
One or more read function calls are issued until all appropriate data is received from the partner. Writes and reads may be alternated as defined by the application protocol until the data flow is complete.

gsk_secure_socket_close()
This function call frees all the resources that are used for the SSL connection.

For both client and server applications, when the application is ready to end and all **gsk_secure_socket_close()** functions complete, destroy the sockets through the **close()** call and issue the **gsk_environment_close()** function call to close the SSL environment and return resources to the operating system.

Note: skread() and **skwrite()** are the routines responsible for sending and receiving data from the socket. They are invoked by the **gsk_secure_socket_init()**, **gsk_secure_socket_read()** and **gsk_secure_socket_write()** functions.

In addition to using the previous SSL programming interfaces in an application, an application is not complete until a key database is available for use by the SSL application. The key database contains certificate information and is a z/OS UNIX System Services file that is built and managed using the **gskkyman** utility. In addition to key database files, a GSKIT CMS V4 file, a PKCS #12 file, a SAF key ring, or a z/OS PKCS #11 token can be utilized for certificate information. For more information about key databases, see Chapter 10, "Certificate/Key management," on page 541.

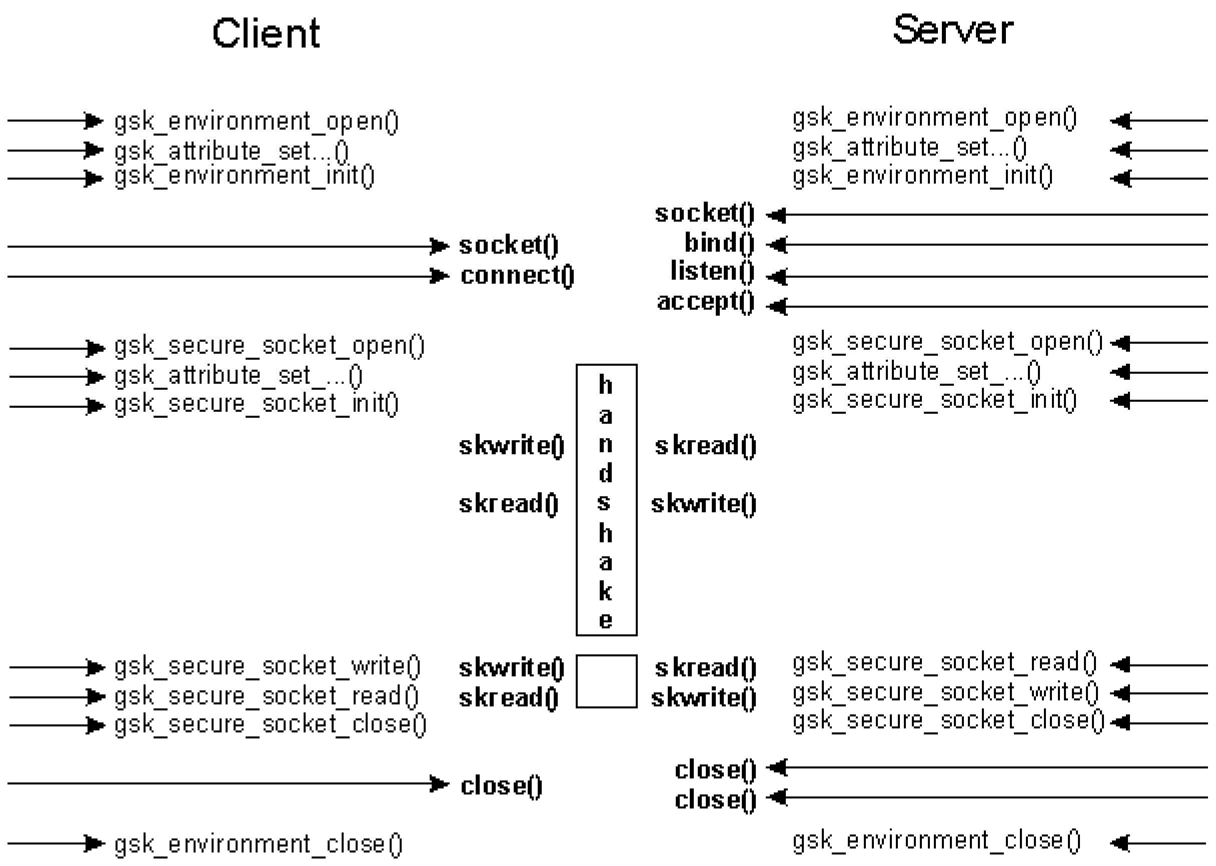

Figure 1. Sockets programming model using System SSL

Chapter 3. Using cryptographic features with System SSL

System SSL uses cryptographic features available on z/OS to offer a comprehensive range of cryptographic support. In addition to software cryptographic processing performed by System SSL, services offered by the Integrated Cryptographic Service Facility (ICSF) and the CP Assist for Cryptographic Function (CPACF) are employed to enhance System SSL with hardware cryptographic support for commonly used algorithms. ICSF also provides support for Elliptic Curve Cryptography (ECC).

In order for System SSL to use cryptographic support provided through ICSF, the ICSF started task must be running and the application user ID must be authorized for the appropriate resources in the RACF® CSFSERV class (when the class is active), either explicitly or through a generic resource profile. See "RACF CSFSERV resource requirements" on page 15 for further details. In addition to the CSFSERV class, the application user ID needs READ access to:

- RACF CSFKEYS class when SAF key rings are being used and the application's certificate keys are stored in ICSF'S PKDS. This access is not required if the CSFKEYS class is not active or the RACF resource is not defined.
- RACF resource USER.token-name within the CRYPTOZ class when either SAF key rings or PKCS #11 tokens are being used and the application's certificate keys are stored as secure keys in an ICSF PKCS #11 token. The CRYPTOZ class must be active and the RACF resource must exist, otherwise access is not granted.

For more information about access to CSFKEYS, see the RACDCERT command in *z/OS Security Server RACF Command Language Reference*. For more information about the CRYPTOZ class, see *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

Guidelines for using hardware cryptographic features

System SSL handshake processing uses the RSA and digital signature functions that are expensive functions when performed in software. For installations that have high volumes of SSL handshake processing, by using the capabilities of the hardware provides maximum performance and throughput. For example, having a Crypto Express Coprocessor or Accelerator or both results in the maximum clear key RSA and digital signature processing being done in hardware.

For installations that are more concerned with the transfer of encrypted data than with SSL handshakes, moving the encrypt/decrypt processing to hardware (CPACF) provides maximum performance. The encryption algorithm is determined by the SSL cipher value. To use hardware, the ciphers symmetric algorithm must be available in hardware. For example, an application encrypting or decrypting data using the symmetric algorithm 3DES or AES would benefit from the processing being done in the hardware.

For maximum performance and throughput, use hardware for both the SSL handshake and data encryption or decryption or both.

For information about the types of hardware cryptographic features that are supported by ICSF, see *z/OS Cryptographic Services ICSF Overview*. For information about configuring and using ICSF, see *z/OS Cryptographic Services ICSF Administrator's Guide* and *z/OS Cryptographic Services ICSF System Programmer's Guide*.

For products using System SSL, see the specific product publications for information about System SSL and ICSF considerations.

Access to ICSF cryptographic services can be controlled by the z/OS Security Server (RACF). For more information, see the topic about controlling who can use cryptographic keys and services in *z/OS Cryptographic Services ICSF Administrator's Guide*.

Overview of hardware cryptographic features and System SSL

System SSL might use ICSF or the CPACF for cryptographic hardware support, if they are available. Cryptographic hardware support provides performance benefits over software processing and might be used for particular cryptographic algorithms instead of the System SSL software algorithms. System SSL also uses ICSF for cryptographic algorithms that are not supported within the software of System SSL (for example, Elliptic Curve Cryptography). For algorithms for which System SSL has software versions, System SSL checks for hardware support during its runtime initialization and uses the support if available, unless the application specifies otherwise. See Appendix A, “Environment variables,” on page 763 for information about the GSK_HW_CRYPTO environment variable (which specifies whether the hardware cryptographic support is used).

When using a secure key (a key stored either in the ICSF PKDS or a PKCS #11 token) or an algorithm that is not supported within System SSL's software, System SSL always uses ICSF for the cryptographic operation. If ICSF is not available, the operation fails.

System SSL uses the CPACF directly for symmetric encryption algorithms DES, 3DES, and AES-CBC, and SHA based digest algorithms. If the algorithm is not available in the CPACF, System SSL uses its internal software implementation.

If during System SSL run-time initialization, System SSL detects the presence of a cryptographic card or cards, it will call ICSF to perform RSA signature and encryption operations supported by the available card or cards. If there are no cards or ICSF is not available, System SSL uses its internal software implementations. If a cryptographic card becomes available after System SSL's run-time initialization, System SSL will not detect the presence of the card.

If a severe ICSF error occurs during a clear key RSA operation, System SSL stops using the hardware support and reverts to using the software algorithms, when applicable. In this event, hardware failure notification is available through the SSL Started Task or SSL trace output, if either facility is enabled. The SSL Started Task outputs an error message to the console on the first occurrence of the hardware failure and to the system log on any subsequent events. A message showing the failing encryption algorithm appears in the system log only. Subsequent cryptographic operations for the current SSL application that attempt to use this algorithm is performed in software. When the severe problem with ICSF is resolved, System SSL functionality will attempt to utilize ICSF services again for clear key RSA operations.

When using a secure key (a key stored either in the ICSF PKDS or a PKCS #11 token) or an algorithm that is not supported within System SSL's software (ECC, AES-GCM, and ChaCha20), System SSL always uses ICSF for the cryptographic operation. If ICSF is not available when these algorithms are called upon, the operation fails. Clear key ECC, AES-GCM, and ChaCha20 operations use ICSF PKCS #11 support. For more information about ECC cryptographic support, see “Elliptic Curve Cryptography support” on page 12.

Note: System SSL can use secure key support for RSA, DSA, and ECC through ICSF. System SSL does not use secure symmetric keys except for the symmetric key that is used to encrypt the private key being encrypted by the **gsk_make_enveloped_private_key_msg()** API.

Table 1 on page 10 describes the hardware cryptographic functions that are used by System SSL or through ICSF under different hardware configurations when in non-FIPS mode. For FIPS mode hardware exploitation, see Algorithm support: FIPS and non-FIPS.

Table 1. Hardware cryptographic functions used by System SSL

	IBM z13/z13s				**IBM z14/z14 ZR1**				**IBM z15/IBM z15 T02**				**IBM z16**			
Algorithm	**CPACF**	**CEX5C**	**CEX5A**	**CEX5P**	**CPACF**	**CEX5C/ CEX6C**	**CEX5A / CEX6A**	**CEX5P/ CEX6P**	**CPACF**	**CEX5C/ CEX6C/ CEX7C**	**CEX5A / CEX6A / CEX7A**	**CEX5P/ CEX6P/ CEX7P**	**CPACF**	**CEX6C/ CEX7C/ CEX8C**	**CEX6A / CEX7A / CEX8A**	**CEX6P/ CEX7P/ CEX8P**
3DES	X				X				X				X			
AES 128-bit	X				X				X				X			
AES 256-bit	X				X				X				X			
AES-GCM 128-bit	X				X				X				X			

Table 1. Hardware cryptographic functions used by System SSL (continued)

	IBM z13/z13s				IBM z14/z14 ZR1				IBM z15/IBM z15 T02				IBM z16			
Algorithm	**CPACF**	**CEX5C**	**CEX5A**	**CEX5P**	**CPACF**	**CEX5C/ CEX6C**	**CEX5A / CEX6A**	**CEX5P/ CEX6P**	**CPACF**	**CEX5C/ CEX6C/ CEX7C**	**CEX5A / CEX6A / CEX7A**	**CEX5P/ CEX6P/ CEX7P**	**CPACF**	**CEX6C/ CEX7C/ CEX8C**	**CEX6A / CEX7A / CEX8A**	**CEX6P/ CEX7P/ CEX8P**
AES-GCM 256-bit	X				X				X				X			
DES	X				X				X				X			
ECC Key Generation									X				X			
ECDSA Digital Signature Generate (Secure Private key)		X		X		X		X		X		X		X		X
ECDSA Digital Signature Verification (Clear public key)		X				X			X	X			X	X		
Elliptic Curve Diffie-Hellman (ECDH/ ECDHE) key derivation		X		X		X		X	X	X		X	X	X		X
RSA Decrypt (Clear Private Key)		X	X			X	X			X	X			X	X	
RSA Decrypt (Secure Private Key)		X		X		X		X		X		X		X		X
RSA Digital Signature Generate (Clear or Secure Private key)		X		X Secure key only.		X		X Secure key only.		X		X Secure key only.		X		X Secure key only.
RSA Digital Signature Verification (Clear Public Key)		X	X			X	X			X	X			X	X	
RSA Encrypt (Clear Public Key)		X	X			X	X			X	X			X	X	
RSASSA-PSS signature generation (Clear RSA private key)		X	X			X	X			X	X			X	X	
RSASSA-PSS signature generation (Secure RSA private key)		X		X		X		X		X		X		X		X
RSASSA-PSS signature verification (Clear RSA public key)		X	X			X	X			X	X			X	X	
SHA-1	X				X				X				X			
SHA-2 (SHA-224)	X				X				X				X			

Table 1. Hardware cryptographic functions used by System SSL (continued)

	IBM z13/z13s				IBM z14/z14 ZR1				IBM z15/IBM z15 T02				IBM z16			
Algorithm	**CPACF**	**CEX5C**	**CEX5A**	**CEX5P**	**CPACF**	**CEX5C/ CEX6C**	**CEX5A / CEX6A**	**CEX5P/ CEX6P**	**CPACF**	**CEX5C/ CEX6C/ CEX7C**	**CEX5A / CEX6A / CEX7A**	**CEX5P/ CEX6P/ CEX7P**	**CPACF**	**CEX6C/ CEX7C/ CEX8C**	**CEX6A / CEX7A / CEX8A**	**CEX6P/ CEX7P/ CEX8P**
SHA-2 (SHA-256)	X				X				X				X			
SHA-2 (SHA-384)	X				X				X				X			
SHA-2 (SHA-512)	X				X				X				X			

Note: X = Algorithm supported.

Random byte generation support

System SSL supports the generation of random bytes. This support is performed either by calling the ICSF CSFPPRF callable service or through a software implementation within System SSL. If ICSF is available during System SSL's runtime initialization, System SSL calls ICSF. If unavailable, System SSL's software implementation is used. If ICSF terminates or access to the ICSF callable service CSFPPRF is protected by a CSFSERV class profile and the application user is not authorized, the generation of random bytes is performed in software. For more information about the CSFSERV resource class, see "RACF CSFSERV resource requirements" on page 15.

If the System SSL application is FIPS enabled, see "Random byte generation" on page 21 for more information about random bytes generation in FIPS mode.

Elliptic Curve Cryptography support

System SSL uses ICSF callable services for Elliptic Curve Cryptography (ECC) algorithm support. For ECC support through ICSF, ICSF must be initialized with PKCS #11 support. For more information, see *z/OS Cryptographic Services ICSF System Programmer's Guide*. In addition, the application user ID must be authorized for the appropriate resources in the RACF CSFSERV class, either explicitly or through a generic resource profile. See CSFSERV resources required for hardware support through ICSF callable services for the required CSFSERV resources for each ECC function.

If the ICSF started task is not running as required or ECC support is otherwise unavailable, System SSL will fail if an ECC-based operation is required. In this event, notification is available through return or status codes and System SSL trace output.

Current ICSF cryptographic support for ECC can be verified using the DISPLAY CRYPTO function of the SSL Started Task. See Chapter 11, "SSL started task," on page 611 for more information.

ECC public/private keys must be defined over prime finite fields (F_p type fields) only; characteristic two finite fields (F_{2m} type fields) are not supported. EC domain parameters may be defined using either the specifiedCurve format or the namedCurve format, as described in RFC 5480 (tools.ietf.org/html/rfc5480). If the EC domain parameters are defined using the specifiedCurve format, then they must match a supported named curve.

The following named curves are supported:

- NIST recommended curves
  - secp192r1 – {1.2.840.10045.3.1.1}
  - secp224r1 – {1.3.132.0.33}
  - secp256r1 – {1.2.840.10045.3.1.7}
  - secp384r1 – {1.3.132.0.34}
  - secp521r1 – {1.3.132.0.35}

- Brainpool defined curves
  - brainpoolP160r1 – {1.3.36.3.3.2.8.1.1.1}
  - brainpoolP192r1 – {1.3.36.3.3.2.8.1.1.3}
  - brainpoolP224r1 – {1.3.36.3.3.2.8.1.1.5}
  - brainpoolP256r1 – {1.3.36.3.3.2.8.1.1.7}
  - brainpoolP320r1 – {1.3.36.3.3.2.8.1.1.9}
  - brainpoolP384r1 – {1.3.36.3.3.2.8.1.1.11}
  - brainpoolP512r1 – {1.3.36.3.3.2.8.1.1.13}

For data signature generation and verification operations involving ECC-based algorithms, z/OS System SSL supports ECDSA with SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 digest algorithms. When creating signed certificates using the System SSL certificate management utility, **gskkyman**, or through CMS APIs that use a default digest algorithm, the recommended digest for the ECC key size of the signing private key is used (as specified in the following table).

Table 2. Recommended digest sizes for ECDSA signature key sizes

ECC curve type	ECDSA key sizes (bits)	Recommended digest algorithm	Signature algorithm type
x509_ecurve_brainpoolP160r1 **x509_ecurve_secp192r1** **x509_ecurve_brainpoolP192r1** **x509_ecurve_secp224r1** **x509_ecurve_brainpoolP224r1** **x509_ecurve_secp256r1** **x509_ecurve_brainpoolP256r1** **x509_ecurve_brainpoolP320r1**	160-383	SHA-256	x509_alg_ecdsaWithSha256
x509_ecurve_secp384r1 **x509_ecurve_brainpoolP384r1**	384-511	SHA-384	x509_alg_ecdsaWithSha384
x509_ecurve_brainpoolP512r1 **x509_ecurve_secp521r1**	512 and greater	SHA-512	x509_alg_ecdsaWithSha512

System SSL regards certain EC named curves to be the default curve for their key size. For CMS APIs that require ECC key generation and accept a key size parameter only, the default curve for the key size specified is used. These default EC named curves are outlined in the following table.

Table 3. Default EC named curves for specified key sizes

Key size (bits)	Default EC named curve	Named curve OID
224	secp224r1	1.3.132.0.33
256	secp256r1	1.2.840.10045.3.1.7
320	brainpoolP320r1	1.3.36.3.3.2.8.1.1.9
384	secp384r1	1.3.132.0.34
512	brainpoolP512r1	1.3.36.3.3.2.8.1.1.13
521	secp521r1	1.3.132.0.35

RSASSA-PSS signature support

System SSL uses ICSF callable services for RSASSA-PSS signature algorithm support. For more information, see *z/OS Cryptographic Services ICSF System Programmer's Guide*. In addition, the application user ID must be authorized for the appropriate resources in the RACF CSFSERV class, either explicitly or through a generic resource profile. See CSFSERV resources required for ICSF PKCS #11 callable services support for the required CSFSERV resources for RSASSA-PSS digital signature functional support.

If the ICSF started task is not running as required, System SSL fails if an RSASSA-PSS sign or RSASSA-PSS verify operation is required. In this event, notification is available through return or status codes and System SSL trace output.

System SSL only supports RSASSA-PSS signing and RSASSA-PSS verifying for the following conditions:

1. Key algorithm type x509_alg_rsaEncryption {1.2.840.113549.1.1.1}
2. Key sizes 2048 through 4096 inclusive
3. Digest algorithms
   a. x509_alg_sha256Digest {2.16.840.1.101.3.4.2.1}
   b. x509_alg_sha384Digest {2.16.840.1.101.3.4.2.2}
   c. x509_alg_sha512Digest {2.16.840.1.101.3.4.2.3}
4. Mask Generation Function algorithm
   a. x509_alg_rsaMgf1 {1.2.840.113549.1.1.8}
5. Salt length
   a. x509_alg_sha256Digest (32)
   b. x509_alg_sha384Digest (48)
   c. x509_alg_sha512Digest (64)

Table 4. RSASSA-PSS key algorithm recommendations

Key algorithm	Key Size	Recommended digest algorithm	Recommended mask generation function algorithm
x509_alg_rsaEncryption	>=2048 and <3072	x509_alg_sha256Digest	x509_alg_rsaMgf1
x509_alg_rsaEncryption	>=3072 and <4096	x509_alg_sha384Digest	x509_alg_rsaMgf1
x509_alg_rsaEncryption	4096	x509_alg_sha512Digest	x509_alg_rsaMgf1

If you are using the System SSL **gskkyman** utility to create new certificates (both self-signed and signed) with RSASSA-PSS based signatures, there are only two options for the RSA key size: 2048 and 4096. The signature digest algorithm used when creating a self-signed certificate is the recommended digest for the selected key size. When creating signed certificates, the recommended digest algorithm is used based on the signing RSA private key size (as specified in Table 4 on page 14).

When using System SSL CMS APIs to create self-signed and signed certificates, the caller selects both the signature algorithm and key size. RSA key sizes 2048 thru 4096 inclusive are all compatible with using x509_alg_mgf1Sha256WithRsaSsaPss, x509_alg_mgf1Sha384WithRsaSsaPss, and x509_alg_mgf1Sha512WithRsaSsaPss.

To perform signature processing using secure private keys stored in the PKDS, the following hardware is required:

- IBM z13 or higher.

- When using the Crypto Express5 Coprocessor card, October 2016 or later licensed internal code (LIC) is needed.

For more information about the cryptographic hardware support of RSASSA-PSS operations, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

RACF certificates with their RSA private key stored in the PKDS prior to V2R4 using the RSASSA-PSS signature support may fail due to the key not being protected by an ECC master key when it was created or added in the PKDS.

Diffie-Hellman key agreement

System SSL supports Diffie-Hellman (DH) key agreement group parameters as defined in PKCS #3 (Diffie-Hellman Key Agreement Standard) and RFC 2631 (tools.ietf.org/html/rfc2631). The Diffie-Hellman key agreement parameters are the prime P, the base G, and, in non-FIPS mode, the optional subprime Q, and subgroup factor J.

Diffie-Hellman key pairs are the private value X and the public value Y. The private value X is less than Q-1 if Q is present in the key parameters, otherwise, the private value X is less than P-1.

Multiple Diffie-Hellman key agreement keys can share domain group parameters (P and G). In addition, the Diffie-Hellman key agreement algorithm requires both parties to use the same group parameters when computing the secret value. An SSL client generates temporary Diffie-Hellman values if the group parameters in the client certificate are not the same as the group parameters in the server certificate. DSA keys may also share domain group parameters as Diffie-Hellman keys.

DH keys:

- Can be used only for end user certificates
- Can only be signed using a certificate that contains either an RSA or a DSA key
- Key size when in non-FIPS mode is between 512 and 2048 bits rounded up to a multiple of 64
- Key size in FIPS mode of 2048 bits
- Can only be used for connections where the cipher specification is a fixed Diffie-Hellman key exchange
- When used in fixed Diffie-Hellman key exchange must allow key agreement.

Only an RSA or DSA client certificate can be used in an ephemeral Diffie-Hellman key exchange.

RACF CSFSERV resource requirements

ICSF controls access to cryptographic services through the RACF CSFSERV resource class. An application using System SSL that requires cryptographic support from ICSF must be authorized for the appropriate resources in the class, either explicitly or through a generic resource profile. For more information, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

When the System SSL DLLs are loaded, System SSL determines what hardware is available by using the ICSF Query Algorithm callable service (CSFIQA). For this reason, make sure that the RACF user ID that starts the application can access the CSFIQA resource of the CSFSERV class. If the user ID that starts the SSL application cannot access the CSFIQA resource of the CSFSERV class, System SSL cannot retrieve information by using the CSFIQA callable service, and the informational message ICH408I (which indicates insufficient authorization) may be issued to the console. Although System SSL processing continues, System SSL might not be aware of all the hardware that is currently available.

The following tables summarize the CSFSERV resources required for each ICSF cryptographic function used by System SSL.

Table 5. CSFSERV resources required for hardware support through ICSF callable services

Function	ICSF callable services	CSFSERV resources required
ECC Digital Signature Generation (private key in the PKDS)	CSNDDSG	CSFDSG
PKA (RSA) Decrypt	CSNDPKB CSNDPKD	-- CSFPKD
PKA (RSA) Encrypt	CSNDPKB CSNDPKE	-- CSFPKE
RSA Digital Signature Generation	CSNDPKB CSNDPKI CSNDDSG	-- CSFPKI CSFDSG
RSA Digital Signature Verify	CSFDPKB CSNDDSV	-- CSFDSV
RSASSA-PSS Digital Signature Generation (private key in PKDS)	CSNDDSG	CSFDSG

Table 6. CSFSERV resources required for ICSF PKCS #11 callable services support

Function	ICSF PKCS #11 callable services	CSFSERV resources required
AES-GCM Secret Key Decrypt	CSFPSKD CSFPTRC	CSF1SKD CSF1TRC
AES-GCM Secret Key Encrypt	CSFPSKE CSFPTRC	CSF1SKE CSF1TRC
ChaCha20 Secret Key Decrypt	CSFPSKD CSFPTRC	CSF1SKD CSF1TRC
ChaCha20 Secret Key Encrypt	CSFPSKE CSFPTRC	CSF1SKE CSF1TRC
Diffie-Hellman in FIPS mode	CSFPTRC CSFPDVK CSFPGKP CSFPGSK CSFPGAV CSFPTRD	CSF1TRC CSF1DVK CSF1GKP CSF1GSK CSF1GAV CSF1TRD
ECC Digital Signature Generation	CSFPTRC CSFPPKS CSFPTRD	CSF1TRC CSF1PKS CSF1TRD

Table 6. CSFSERV resources required for ICSF PKCS #11 callable services support (continued)

Function	ICSF PKCS #11 callable services	CSFSERV resources required
ECC Digital Signature Verify	CSFPTRC CSFPPKV CSFPTRD	CSF1TRC CSF1PKV CSF1TRD
ECC Key Generation	CSFPGKP CSFPGAV CSFPTRD	CSF1GKP CSF1GAV CSF1TRD
ECDH Derive Key	CSFPTRC CSFPDVK CSFPGAV CSFPTRD	CSF1TRC CSF1DVK CSF1GAV CSF1TRD
PKA (RSA) Decrypt in FIPS mode	CSFPPD2	CSFPKD
PKA (RSA) Encrypt in FIPS mode	CSFPPE2	CSFPKE
Random Number Generation	CSFPPRF	CSFRNG
RSA Digital Signature Verify in FIPS mode	CSFPPV2	CSFDSV
RSA PKCS #11 Secure Key Decrypt	CSFPPKS	CSF1PKS
RSASSA-PSS Digital Signature Generate	CSFPOWH CSFPTRC CSFPTRD	CSFOWH CSF1TRC CSF1TRD
RSASSA-PSS Digital Signature Verify	CSFPOWH CSFPTRC CSFPTRD	CSFOWH CSF1TRC CSF1TRD
Secure PKCS #12 Private Key Export	CSFPGSK CSFPWPK CSFPTRC CSFPTRD	CSF1GSK CSF1WPK CSF1TRC CSF1TRD
Secure PKCS #7 Make Enveloped Data Message	CSFPTRC CSFPGSK CSFPWPK CSFPTRD	CSF1TRC CSF1GSK CSF1WPK CSF1TRD
Secure PKCS #7 Read Enveloped Data Message	CSFPPKS	CSF1PKS

PKCS #11 and setting CLEARKEY resource within CRYPTOZ class

The CLEARKEY.token-name resource within the CRYPTOZ class controls the ICSF policy for creating a clear key versus a secure key. When the resource is defined and set to NONE, System SSL's usage of the PKCS #11 callable services to generate keys is restricted to secure keys only. This causes functions within System SSL to fail. System SSL uses both explicit tokens and the SYSTOK-SESSION-ONLY omnipresent token.

The following are examples that can fail in this environment in System SSL:

- The **gskkyman** utility or CMS APIs that create ECC or DH (FIPS mode) keys or certificates.
- Ephemeral DH key exchanges during an SSL/TLS handshake.
- Ephemeral ECDH key exchanges during an SSL/TLS handshake.

PKCS #11 Cryptographic operations using ICSF handles

When executing cryptographic operations against PKCS #11 certificates and keys, System SSL uses the 44-byte handle as defined by ICSF. See Introducing PKCS #11 and using PKCS #11 callable services in *z/OS Cryptographic Services ICSF Application Programmer's Guide* for the definition of this handle. When using System SSL CMS APIs, the ICSF handle may be referred to as a label, for example, the input *private_key_label* to Certificate Management Services (CMS) API **gsk_make_enveloped_private_key_msg()** is the ICSF definition of a handle. The ICSF 44-byte handle is not the same as a PKCS #11 object handle, which is defined as PKCS #11 attribute CK_ULONG.

Chapter 4. System SSL and FIPS 140-2

National Institute of Standards and Technology (NIST) is the US federal technology agency that works with industry to develop and apply technology, measurements, and standards. One of the standards that are published by NIST is the Federal Information Processing Standard Security Requirements for Cryptographic Modules referred to as ‘FIPS 140-2’. FIPS 140-2 provides a standard that can be required by organizations that specify that cryptographic-based security systems are to be used to provide protection for sensitive or valuable data.

The objective of System SSL is to provide the capability to execute securely in a mode that is designed to meet the NIST FIPS 140-2 Level 1 criteria. System SSL can be executed in either ‘FIPS mode’ or ‘non-FIPS mode’. System SSL by default runs in ‘non-FIPS mode’ mode. Applications wanting to execute in FIPS mode must code to the gsk_fips_state_set() API. For additional information, see “Application changes” on page 27.

To meet the FIPS 140-2 Level 1 criteria, System SSL, when executing in FIPS mode, is more restrictive concerning cryptographic algorithms, protocols, and key sizes that can be supported. Original FIPS mode implementation enforced 80-bit security strength.

NIST Special Publication 800-131A Revision 1 (SP800-131Ar1) details a more secure implementation of FIPS mode support. SP800-131Ar1 increases the algorithm and cryptographic key strength from 80 bit to 112 or higher bit key strength.

To meet SP800-131Ar1 criteria, additional enumeration values have been added to **gsk_fips_state_set()**. When GSK_FIPS_STATE_LEVEL1 is used to set FIPS mode state, this is equivalent to GSK_FIPS_STATE_ON and enforces 80-bit security strength for all operations. GSK_FIPS_STATE_LEVEL2 uses 112-bit security strength when generating new keys, digital signatures, and RSA encryption. However, GSK_FIPS_STATE_LEVEL2 allows for 80-bit security when performing digital signature verification, RSA decryption, and Triple DES decryption when processing already protected information. GSK_FIPS_STATE_LEVEL3 enforces FIPS mode where 112 bit or higher security strength size is acceptable for all operations and 80-bit strength size is not allowed for any operation.

The meaning of level in the preceding enumerator values do not correspond to the different FIPS 140-2 Validation Levels. Level is being used by System SSL to indicate different security key enforcements that are performed by System for FIPS 140-2 Level 1 (Pre and Post NIST SP800-131Ar1).

Algorithms and key sizes

When executing in FIPS mode, System SSL continues to take advantage of the CP Assist for Cryptographic Function (CPACF) when available either directly or through ICSF. Hardware cryptographic card functions allowed in FIPS mode support clear keys (requires a cryptographic card to be defined as an accelerator and online prior to the startup of ICSF) and secure PKCS #11 keys. Secure keys stored in the PKDS are not supported.

Table 7 on page 19 summarizes the differences between FIPS mode and non-FIPS mode algorithm support. Hardware availability depends on the processor and CPACF feature installed. See Chapter 3, “Using cryptographic features with System SSL,” on page 9 for more information about processors, CPACF algorithm availability, and cryptographic card support.

Table 7. Algorithm support: FIPS and non-FIPS

	Non-FIPS				FIPS						
Algorithm	**Sizes**	**System SSL software**	**Direct calls to CPACF**	**Support through ICSF**[1]	**Sizes**	**System SSL software**	**Direct calls to CPACF**	**Support through ICSF (IBM z13, IBM z13s, IBM z14, IBM z15, IBM z16)**			
								Software	**CPACF**	**CEXnA**	**CEXnP**
3DES	168	X	X		168	X	X				

Table 7. Algorithm support: FIPS and non-FIPS (continued)

	Non-FIPS				FIPS						
Algorithm	**Sizes**	**System SSL software**	**Direct calls to CPACF**	**Support through ICSF**[1]	**Sizes**	**System SSL software**	**Direct calls to CPACF**	**Support through ICSF (IBM z13, IBM z13s, IBM z14, IBM z15, IBM z16)**			
								Software	**CPACF**	**CEXnA**	**CEXnP**
AES	128 and 256	X	X		128 and 256	X	X				
AES-GCM	128 and 256			X	128 and 256				X		
Brainpool Curves - ECC, ECDH, ECDHE	160-512			X							
DES	56	X	X								
DH, DHE	512–2048	X			2048			X		X - Key agreement	
DSA	512–2048	X			1024-2048	X					
MD5	48	X									
NIST Curves - ECC, ECDSA, ECDH, ECDHE	192-521			X	192-521			X			X - ECDSA signature generate, ECDH/ ECDHE key agreement
RC2	40 and 128	X									
RC4	40 and 128	X									
RSA	512–4096	X		X	1024–4096	X				X – Encrypt, Decrypt, Signature Verify	X – Encrypt, Decrypt, Signature Generate
RSASSA-PSS	2048 – 4096			X	2048 – 4096			X			
SHA-1	160	X	X		160	X	X				
SHA-2	224, 256, 384, and 512	X	X		160	X	X				

Notes:

- [1] For information on usage of ICSF in non-FIPS mode, see Table 1 on page 10.
- In FIPS mode, only NIST ECC recommended curves are currently supported. Curves under 224 bits are not recommended. Enforcement is the responsibility of the calling application or the system administrator.
- NIST SP800-131 recommended transition algorithm key sizes of RSA >= 2048, DSA >=2048, NIST ECC recommended curves >= 224, and the disallowment of SHA-1 for digital signature generation. Enforcement is the responsibility of the calling application or the system administrator.

  Brainpool ECC curves are not supported in FIPS mode.

Table 8 on page 21 summarizes the differences between FIPS modes ON and LEVEL1 thru LEVEL3 algorithm support.

Table 8. Algorithm support sizes: FIPS States ON and LEVEL1 thru LEVEL3

Algorithm	**ON or LEVEL1**	**LEVEL2**	**LEVEL3**
3DES	168		
AES	128 and 256		
Digital Signature Generation functions[3, 4]	SHA-1 thru SHA-512	SHA-224 thru SHA-512	
Digital Signature Verification functions[3, 4]	SHA-1 thru SHA-512		SHA-224 thru SHA-512
HMAC	80 bits and higher		112 bits and higher
DSA[1]	1024 thru 2048	2048	
DH	2048		
ECC	NIST ECC 192 thru 521		NIST ECC 224 thru 521
RSA[2]	1024 thru 4096	2048-4096	

Footnotes for Table 8 on page 21:

1

For DSA keys, when functioning at GSK_FIPS_STATE_LEVEL2 or GSK_FIPS_STATE_LEVEL3, generating new keys and digital signatures are enforced at the 112 bit security strength. When performing digital signature verification, GSK_FIPS_STATE_ON (GSK_FIPS_STATE_LEVEL1) and GSK_FIPS_STATE_LEVEL2 80 bit security is allowed. Key sizes 1024 or less are associated with 80 bit security strength. Keys sizes 2048 or higher are associated with 112 bit security strength.

2

For RSA keys, when functioning at GSK_FIPS_STATE_LEVEL2 or GSK_FIPS_STATE_LEVEL3, generating new keys and digital signatures are enforced at the 112 bit security strength. When performing digital signature verification, GSK_FIPS_STATE_ON (GSK_FIPS_STATE_LEVEL1) and GSK_FIPS_STATE_LEVEL2 80 bit security is allowed. Key sizes 1024 or less are associated with 80 bit security strength. Keys sizes 2048 or higher are associated with 112 bit security strength.

3

For Digital Signature Generation and Digital Signature Verification using RSASSA-PSS, digest sizes SHA-1 and SHA-224 are not supported, only digest sizes SHA-256, SHA-384, and SHA-512 are supported.

4

Digital Signature Generation and Digital Signature Verification using SHA-1 when used by the TLS protocol is allowed for all settings.

System SSL RSASSA-PSS only supports digest algorithms SHA-256, SHA-384, and SHA-512. FIPS LEVEL support for RSASSA-PSS signatures:

- GSK_FIPS_STATE_LEVEL2 and GSK_FIPS_STATE_LEVEL3 signature generation requires the digest algorithm size to be SHA-256, SHA-384, or SHA-512.
- GSK_FIPS_STATE_LEVEL2 and GSK_FIPS_STATE_LEVEL3 signature verification does not tolerate digest SHA-1 and SHA-224 for already created objects.

Random byte generation

When executing in FIPS mode, System SSL supports the generation of random bytes. System SSL generates random bytes by using ICSF's CSFPPRF callable service. In order for System SSL to call this service, ICSF must be available before System SSL's run time that is being initialized by the application. If access to the CSFPPRF callable service is protected by a CSFSERV class profile, the application's user

ID must be authorized to use the service. For more information about the CSFSERV resource class, see "RACF CSFSERV resource requirements" on page 15.

RSA digital signature generation, signature verification, encryption, and decryption

When running in FIPS mode, if System SSL detects during runtime initialization that ICSF is available and that a cryptographic card is available that is defined as an accelerator, System SSL attempts to call the ICSF callable services to perform clear key RSA digital signature verification, encryption, and decryption. The accelerator card must be online before the startup of ICSF. If ICSF is unable to perform the RSA cryptographic operation, System SSL performs the cryptographic operation in its software implementation. If access to the ICSF services is being protected by the CSFSERV class profile, the application user ID must be authorized to the CSFPPD2, CSFPPE2, and CSFPPV2 services. For more information about the CSFSERV resource class, see "RACF CSFSERV resource requirements" on page 15.

RSASSA-PSS signature support

System SSL uses ICSF callable services for RSASSA-PSS signature algorithm support. The application user ID must be authorized for the appropriate resources in the RACF CSFSERV class, either explicitly or through a generic resource profile. See Table 6 on page 16 for the required CSFSERV resources for RSASSA-PSS digital signature functional support.

If the ICSF started task is not running as required, System SSL fails if an RSASSA-PSS sign or RSASSA-PSS verify operation is required.

System SSL supports RSASSA-PSS signing and RSASSA-PSS verifying for the following conditions:

1. Key algorithm type x509_alg_rsaEncryption {1.2.840.113549.1.1.1}
2. Minimum key size is 2048.
3. Digest algorithms.
   a. x509_alg_sha256Digest {2.16.840.1.101.3.4.2.1}
   b. x509_alg_sha384Digest {2.16.840.1.101.3.4.2.2}
   c. x509_alg_sha512Digest {2.16.840.1.101.3.4.2.3}

Elliptic Curve Cryptography support

System SSL uses ICSF callable services for Elliptic Curve Cryptography (ECC) algorithm support. For ECC support through ICSF, ICSF must be initialized with PKCS #11 support. For more information, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*. For more information, see z/OS Cryptographic Services ICSF System Programmer's Guide. In addition, the application user ID must be authorized for the appropriate resources in the RACF CSFSERV class, either explicitly or through a generic resource profile. See Table 6 on page 16 for the required CSFSERV resources for each ECC function.

If the ICSF started task is not running as required or ECC support is otherwise unavailable, System SSL will fail if an ECC-based operation is required.

The following NIST recommended named curves are supported:

- secp192r1 – {1.2.840.10045.3.1.1}
- secp224r1 – {1.3.132.0.33}
- secp256r1 – {1.2.840.10045.3.1.7}
- secp384r1 – {1.3.132.0.34}
- secp521r1 – {1.3.132.0.35}

Notes:

- Brainpool curves are not supported in FIPS mode.

- NIST curves under 224 bits are not recommended according to NIST SP800-131. Enforcement is the responsibility of the calling application or the system administrator.

Diffie-Hellman key agreement

When executing in FIPS mode, System SSL uses ICSF's Diffie-Hellman support as documented in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*. In order for System SSL to be able to use ICSF, ICSF must be available before System SSL's run time is being initialized by the application. If access to the ICSF services is being protected by the CSFSERV class profile, the application user ID must be authorized. For more information about the CSFSERV resource class, see "RACF CSFSERV resource requirements" on page 15.

In FIPS mode, the only Diffie-Hellman key agreement parameters used are the prime P and the base G.

Diffie-Hellman key size in FIPS mode is 2048 bits.

Certificates

When executing in FIPS mode, System SSL can only use certificates that use the algorithms and key sizes shown in Algorithm support: FIPS and non-FIPS. During X.509 certificate validation (including CA certificates from untrusted data sources, that is, certificates flowing during the SSL/TLS handshake), if an algorithm that is incompatible with FIPS mode is encountered, then the certificate cannot be used and is treated as not valid.

SSL/TLS protocol

When executing in FIPS mode, applications are allowed to use the TLS V1.0, TLS V1.1, and TLS V1.2 protocols. SSL V2, SSL V3, and TLS V1.3 are not supported. The specification of SSL V2 and SSL V3 during setup of the SSL/TLS application is ignored. If the TLS V1.3 protocol is attempted to be enabled in FIPS mode, the enablement will fail. When executing in FIPS mode, the default 2-character cipher specifications string is:

3538392F3233

When executing in FIPS mode, if GSK_V3_CIPHERS is set to GSK_V3_CIPHERS_CHAR4 and a cipher specification is not set in GSK_V3_CIPHER_SPECS_EXPANDED, the default cipher specification is set as follows:

003500380039002F00320033

If non-FIPS mode ciphers are specified, they are ignored during the TLS handshake processing.

For the complete list of ciphers supported in FIPS mode and their 2-character or 4-character values, see Appendix C, "Cipher suite definitions," on page 795.

System SSL module verification setup

System SSL requires Security Level 3 FMID JCPT451 to be installed in order for enabled applications to execute in FIPS mode. Application enablement requires applications to invoke the **gsk_fips_state_set()** API. For more information about the FIPS enablement API, see "gsk_fips_state_set()" on page 323.

The System SSL modules that support FIPS 140-2 are signed using an IBM key during the build process. If any of your installation's System SSL applications are enabled for FIPS and you intend on using System SSL functions in compliance with the FIPS 140-2 standard, then, in accordance with that standard, the integrity of each System SSL module when being loaded into storage must be verified. The verification of the modules requires additional configuration steps prior to the execution of a FIPS enabled System SSL application.

These steps involve:

- Defining specific RACF profiles to enable the verification of the System SSL module signature (added during the IBM module build process) when loaded by the z/OS loader.
- Defining specific RACF profiles and identifying which System SSL modules require signature verification.

Signature verification provides a method to ensure that the System SSL modules remain unchanged from the time they were built, installed onto the system, and loaded into storage to be used by a FIPS enabled System SSL application.

The IBM key used to sign the System SSL modules is an RSA private key that belongs to an X.509 certificate signed by the STG Code Signing CA - G2 certificate. This certificate is shipped as a default CERTAUTH certificate in the RACF database under the label 'STG Code Signing CA - G2'.

Note: A sample clist, GSKRACF, is shipped in *pdsename*.SGSKSAMP to assist you with the RACF commands needed to enable signature verification.

The following steps need to be followed by the system administrator to enable signature validation of the System SSL modules:

1. Mark the IBM root CA as TRUSTed if not already TRUSTed

   ```
   RACDCERT CERTAUTH LIST(LABEL('STG Code Signing CA - G2'))
   RACDCERT CERTAUTH ALTER (LABEL('STG Code Signing CA - G2')) TRUST
   ```

2. Create a key ring to hold the STG Code Signing CA - G2 certificate and connect the certificate to the key ring.

   The key ring needs to be owned by a valid RACF ID and the key ring must be defined in uppercase. Make sure that the ID is an ID of a security administrator. In our example the security administrator ID is RACFADM.

   There can only be one designated signature verification key ring active at one time. If already active, add the CA certificate to the key ring. If not already active create the key ring. The suggested key ring name is CODE.SIGNATURE.VERIFICATION.KEYRING.

   - Determine if signature verification key ring is already active:

     ```
     RLIST FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION
     ```

     The key ring is present in the APPLICATION DATA field
   - Create key ring if needed and connect CA certificate:

     ```
     RACDCERT ID(RACFADM) ADDRING(CODE.SIGNATURE.VERIFICATION.KEYRING)
     ```

     ```
     RACDCERT ID(RACFADM) CONNECT(RING(CODE.SIGNATURE.VERIFICATION.KEYRING)
     CERTAUTH LABEL('STG Code Signing CA - G2') USAGE(CERTAUTH))
     ```
   - If a key ring exists, verify that the CA certificate is connected to the key ring. If not connected, connect the certificate:

     ```
     RACDCERT ID(RACFADM) LISTRING(CODE.SIGNATURE.VERIFICATION.KEYRING)
     ```

     ```
     RACDCERT ID(RACFADM) CONNECT(RING(CODE.SIGNATURE.VERIFICATION.KEYRING)
     CERTAUTH LABEL('STG Code Signing CA - G2') USAGE(CERTAUTH))
     ```

3. Create the FACILITY class profile that tells RACF the key ring to use for module signature verification if it is not already defined.

   Note: Because of space constraints, the second command example appears on two lines. However, the command should be entered completely (on one line) on your system.

   ```
   RLIST FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION
   ```

   ```
   RDEFINE FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION APPLDATA('RACFADM/
   CODE.SIGNATURE.VERIFICATION.KEYRING')
   ```

4. Activate your profile changes in the FACILITY, DIGTCERT or DIGTRING class, or both the DIGTCERT and DIGTRING classes, if RACLISTed.

```
SETROPTS RACLIST(FACILITY) REFRESH
SETROPTS RACLIST(DIGTCERT, DIGTRING) REFRESH
```

5. Activate PROGRAM control, if not already active.

   ```
   SETROPTS WHEN(PROGRAM)
   ```

   Note: Installations that have not previously turned on program control, may encounter problems after issuing `SETROPTS WHEN(PROGRAM)`. Program control is necessary for signature verification, hence installations must evaluate the impact of enabling program control for the first time.

6. Create the PROGRAM class profile that protects the program verification module IRRPVERS and specify its signature verification options.

   Note: Because of space constraints, the command appears on two lines. However, the command should be entered completely (on one line) on your system.

   ```
   RDEFINE PROGRAM IRRPVERS ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
   SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))
   ```

7. Refresh the PROGRAM class.

   ```
   SETROPTS WHEN(PROGRAM) REFRESH
   ```

8. Contact your system programmer to complete this step.

   a. Notify your system programmer to initialize program signature verification by running the IRRVERLD program which loads and verifies the program verification module IRRPVERS. For programming information, see *z/OS Security Server RACF System Programmer's Guide*.

   b. Check with your system programmer to ensure that IRRVERLD executed successfully. If it did not execute successfully, work with your system programmer to check error messages. Correct any setup errors and retry.

   c. Do **not** define PROGRAM profiles for the System SSL modules until IRRVERLD executes successfully.

9. Create the PROGRAM class profiles to indicate that the System SSL modules must be signed. The load should fail if the signature cannot be verified and auditing should occur for failure only. If your installation requires event logging for the signature verification, see the RALTER and RDEFINE commands in the *z/OS Security Server RACF Command Language Reference* for customizing the SIGAUDIT operand within the SIGVER segment.

   Note: Because of space constraints, the command examples appear on two lines. However, the command should be entered completely (on one line) on your system.

   ```
   RDEFINE PROGRAM GSKSSL ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
   SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

   RDEFINE PROGRAM GSKSSL64 ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
   SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

   RDEFINE PROGRAM GSKS31F ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
   SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

   RDEFINE PROGRAM GSKS64F ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
   SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

   RDEFINE PROGRAM GSKCMS31 ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
   SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

   RDEFINE PROGRAM GSKCMS64 ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
   SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

   RDEFINE PROGRAM GSKC31F ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
   ```

```
SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

RDEFINE PROGRAM GSKC64F ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

RDEFINE PROGRAM GSKSRVR ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

RDEFINE PROGRAM GSKKYMAN ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

RDEFINE PROGRAM GSKSRBRD ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

RDEFINE PROGRAM GSKSRBWT ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))
```

10. Refresh the PROGRAM class.

    ```
    SETROPTS WHEN(PROGRAM) REFRESH
    ```

Performance guideline

RACF can use virtual lookaside facility (VLF) to cache signature verification data in order to improve the performance of signature verification of signed program objects. This in turn can improve the load time of the signed System SSL program objects. For more information about using VLF see *VLF considerations for program signature verification* in *z/OS Security Server RACF System Programmer's Guide*.

Certificate stores

To use FIPS mode, certificates can be stored in either a SAF key ring, PKCS #11 token, PKCS #12 file, GSKIT CMS V4 key database, or a FIPS mode key database. All certificates in a certificate chain to be used by a FIPS enabled application must use algorithms and key sizes as specified in Algorithm support: FIPS and non-FIPS and Table 8 on page 21.

SAF key rings, PKCS #11 tokens, and GSKIT CMS V4 key database files

Provided a certificate and its signers chain use only valid algorithms and key sizes, then there are no changes that are required if using a SAF key ring, a PKCS #11 token, or a GSKIT key database. A SAF key ring, PKCS #11 token, or GSKIT CMS V4 key database may contain certificates with keys sizes or algorithms that are not supported in FIPS mode if those certificates are never used while executing in FIPS mode. While executing in FIPS mode, if an attempt to use a certificate with unsupported key size or algorithms is made, then the process fails. The corrective action is to either add/replace certificates with key sizes and algorithms that are valid in FIPS mode, or execute in non-FIPS mode.

The **gskkyman** utility runs in non-FIPS mode when managing PKCS #11 tokens. It is therefore possible to add certificates/keys with algorithms or key sizes that are not supported if the PKCS #11 token is later used while executing in FIPS mode.

Key database files

To use a key database in FIPS mode, it must be created as a FIPS mode database. Key databases that are created through **gskkyman** not explicitly specifying FIPS during creation, or created through an application not executing in FIPS mode, cannot be used by an application executing in FIPS mode. To create a FIPS mode key database using the **gskkyman** utility, see "Creating, opening, and deleting a key database file" on page 559. To create a FIPS mode key database using the Certificate Management Services API, the application must start in FIPS mode (see "gsk_fips_state_set()" on page 323).

The following are key points when using FIPS key databases:

- Certificates that meet any of the requirements for any FIPS state LEVEL (see Algorithm support: FIPS and non-FIPS) can be added to a FIPS key database.
- A FIPS key database may only be modified if executing in FIPS mode. When opening an existing FIPS key database, the **gskkyman** utility ensures that it is executing in FIPS mode. If an application modifies the key database by using the Certificate Management Services (CMS) APIs, then it too must ensure that it is executing in FIPS mode.
- A FIPS key database can be used in non-FIPS mode if it is opened for read only.
- A non-FIPS key database cannot be opened while executing in FIPS mode.

The **gskkyman** utility automatically detects when a FIPS mode key database is opened, and executes in FIPS mode. This ensures that only certificates or certificate requests that meet the FIPS mode requirements in Algorithm support: FIPS and non-FIPS may be added to the key database.

PKCS #12 files

To use a PKCS #12 file in FIPS mode, the file must be protected using 3DES. When creating a PKCS #12 file from certificates within a key database file, using the **gskkyman** utility, the certificates must be exported using strong encryption.

Provided a certificate and its signers chain use only valid algorithms and key sizes, there are no changes that are required if using a PKCS #12 file. A PKCS #12 file may contain certificates with keys sizes or algorithms that are not supported in FIPS mode. While executing in FIPS mode, if an attempt to use a certificate with unsupported key size or algorithms is made, the process fails. The corrective action is to either add or replace certificates with key sizes and algorithms that are valid in FIPS mode, or to execute in non-FIPS mode.

Application changes

To use System SSL in FIPS mode, application changes are required. By default, all applications that use System SSL execute in non-FIPS mode. The application must request that System SSL execute in FIPS mode in the very early stages of interaction with the System SSL API. The application does this by invoking the function **gsk_fips_state_set()** (see "gsk_fips_state_set()" on page 323). To set FIPS mode, **gsk_fips_state_set()** must be executed before all other System SSL functions except for **gsk_get_cms_vector()**, **gsk_get_ssl_vector()** and **gsk_fips_state_query()**. It is possible to switch to non-FIPS mode later. It is not possible to switch from non-FIPS mode to FIPS mode at any time.

When calling the **gsk_fips_state_set()** API to set FIPS mode, each cryptographic algorithm implemented through System SSL software goes through a validation test. The validation test, also known as a Known Answer Test, involves exercising the algorithm and comparing the results for correctness. See the **gsk_perform_kat()** API for the list of algorithms validated. If an issue is encountered during the validation, **gsk_fips_state_set()** will fail with return code CMSERR_KATPW_FAILED or CMSERR_KATPW_ICSF_FAILED.

The FIPS mode setting applies to the entire process. Once set, then all threads of the same process execute in FIPS mode. If any thread switches to non-FIPS mode, then all threads in the same process execute in non-FIPS mode.

When executing in FIPS mode and a severe cryptographic problem is encountered, one of the following return codes is returned from the API executing at the time of failure. These return codes should be treated as severe and the application should be terminated and restarted. If execution continues, all APIs except for **gsk_get_cms_vector()**, **gsk_get_ssl_vector()**, **gsk_fips_state_query()**, **gsk_query_crypto_level()**, and **gsk_strerror()** fails.

- CMSERR_FIPS_KEY_PAIR_CONSISTENCY - Failure when generating either an RSA or DSA key pair
- CMSERR_KATPW_FAILED - Failure was encountered by the **gsk_perform_kat()** API when performing known answer tests against the System SSL cryptographic algorithms.
- CMSERR_KATPW_ICSF_FAILED - Failure was encountered by the **gsk_perform_kat()** API when performing known answer tests using ICSF.

The sample files (see Appendix B, "Sample C++ SSL files," on page 791) client.cpp and server.cpp demonstrate the use of **gsk_fips_state_set()** to set the application to run in FIPS mode. In both cases, the **gsk_fips_state_set()** function is invoked before any other System SSL function.

SSL started task

The System SSL started task (GSKSRVR) executes in non-FIPS mode by default. In order for the GSKSRVR started task to execute in FIPS mode, environment variable GSK_FIPS_STATE must be specified and set to GSK_FIPS_STATE_ON in the envar file in the GSKSRVR home directory. If the GSKSRVR is unable to execute in FIPS mode (for example, the Level 3 FMID JCPT451 is not installed), it executes in non-FIPS mode after issuing message GSK01054E (see "SSL started task messages (GSK01nnn)" on page 741).

Sysplex session ID cache

GSKSRVR must be running in FIPS mode to maintain Sysplex Session ID cache entries for SSL server applications executing in FIPS mode. An SSL server application executing in FIPS mode caches its session in the Sysplex Session cache provided GSKSRVR is also executing in FIPS mode. An SSL server application executing in non-FIPS mode is able to cache its session in the Sysplex Session cache if GSKSRVR is executing in either FIPS mode or non-FIPS mode.

An SSL server application executing in FIPS mode is only able to resume a Sysplex Session cached session if it was for a session that executed in FIPS mode when the cache entry was created. Non-FIPS SSL server applications can resume FIPS and non-FIPS sessions that are cached in the Sysplex Session cache.

SSL servers executing in non-FIPS mode on systems with a back-level GSKSRVR are able to resume FIPS and non-FIPS sessions that are cached in the Sysplex Session cache by systems where the System SSL started task is executing in FIPS mode.

Chapter 5. Writing and building a z/OS System SSL application

This topic describes how to write, build, and run a secure socket layer (SSL) application that uses the System SSL programming interfaces. You can write both client and server applications using the System SSL (TLS/SSL) programming interfaces.

In Version 1 Release 2 of z/OS, a new set of functions were added that superseded some functions from previous System SSL releases. The functions that were superseded are referred to collectively as the deprecated SSL interface. It is suggested that new application programs do not use the deprecated SSL interface. For a complete list and descriptions of the suggested APIs, see Chapter 7, "API reference," on page 73. See Chapter 9, "Deprecated Secure Socket Layer (SSL) APIs," on page 513 for more information about deprecated APIs.

Note: When migrating from the deprecated SSL interface, the entire System SSL application must be migrated. The application must not contain a mixture of deprecated and superseding APIs.

In addition to writing the SSL applications, you must have a certificate repository available for the application. The certificate repository can be a key database file, GSKIT CMS V4 file, PKCS #12 file, PKCS #11 token, or SAF key ring. See Chapter 10, "Certificate/Key management," on page 541 for details about creating and managing key database files or PKCS #11 tokens. For SAF key rings, see the RACDCERT command information in *z/OS Security Server RACF Command Language Reference* for more information.

Sample programs using the new APIs are shipped in `/usr/lpp/gskssl/examples`.

For information on how to build the samples programs, see Appendix B, "Sample C++ SSL files," on page 791. An example of how to compile the sample programs **server, server_tls, client, client_tls,** and **display_certificate** in the z/OS shell with the **make** command is included. Another example of building the **display_certificate** program outside of the z/OS shell using JCL is also included.

Writing a System SSL source program

The first step in creating a System SSL application is to write the source program using the System SSL programming interfaces. See Chapter 7, "API reference," on page 73 for a description of the format of the System SSL programming interfaces.

Before establishing a secure connection, SIGPIPE signals should be set to be ignored or a signal handler should be defined. TCP/IP functions can cause SIGPIPE signals. When the signal is ignored, TCP/IP reflects the signal as an EPIPE error for the TCP/IP functions.

Create an SSL environment

For both the client and server System SSL programs, you must initialize the System SSL environment using the programming interfaces associated with the SSL environment layer.

gsk_environment_open()
Will define and obtain storage for the SSL environment and return an environment handle to be used on subsequent API invocations.

gsk_attribute_set...()
Sets environment attributes such as:

- The SSL protocol version to be used: SSL Version 2.0, SSL Version 3.0, TLS Version 1.0, TLS Version 1.1, TLS Version 1.2, or TLS Version 1.3.
- The key database to be used. (key database file, PKCS #12 file, SAF key ring or z/OS PKCS #11 token)

- The password for the key database. This can be specified directly by the application or by using a stashed password file. See Chapter 10, “Certificate/Key management,” on page 541 for details about creating a stashed password file.

  Note: When using SAF key rings or z/OS PKCS #11 tokens, the password and stash file must not be specified. When using a PKCS #12 file, a stash file must not be specified.
- The amount of time the SSL session identifier information is valid. By using already negotiated and agreed to SSL session identifier information, System SSL can reduce the amount of data exchanged during the SSL handshake that occurs during the **gsk_secure_socket_init()** call.

gsk_environment_init()
  Initializes the SSL environment.

This example code illustrates how to call the environment layer programming interface from a client or server System SSL program. In this example, TLS Version 1.2 support is requested, /keyring/key.kdb is the key database that is used, the password for the key database is "password", and default values are taken for the remaining SSL environment variable attributes.

```
gsk_handle env_handle;
int rc;
/* create the SSL environment */
rc = gsk_environment_open(&env_handle);
/* set environment attributes */
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV2, GSK_PROTOCOL_SSLV2_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV3, GSK_PROTOCOL_SSLV3_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1, GSK_PROTOCOL_TLSV1_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_1, GSK_PROTOCOL_TLSV1_1_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_2, GSK_PROTOCOL_TLSV1_2_ON);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_3, GSK_PROTOCOL_TLSV1_3_OFF);
rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_FILE, "/keyring/key.kdb",0);
rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_PW, "password",0);
/* initialize environment */
rc = gsk_environment_init(env_handle);
```

This example code illustrates how to create an SSL environment for a server System SSL program supporting TLS Version 1.0, TLS Version 1.1, and TLS Version 1.2.

```
gsk_handle env_handle;
int rc;
/* create the SSL environment */
rc = gsk_environment_open(&env_handle);
/* set environment attributes */
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV2, GSK_PROTOCOL_SSLV2_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV3, GSK_PROTOCOL_SSLV3_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1, GSK_PROTOCOL_TLSV1_ON);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_1, GSK_PROTOCOL_TLSV1_1_ON);
/* By default, TLS V1.1 protocol is set off */
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_2, GSK_PROTOCOL_TLSV1_2_ON);
/* By default, TLS V1.2 protocol is set off */
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_3, GSK_PROTOCOL_TLSV1_3_OFF);
/* By default, TLS V1.3 protocol is set off */
rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_FILE, "/keyring/key.kdb",0);
rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_PW, "password",0);
/* initialize environment */
rc = gsk_environment_init(env_handle);
```

This example code illustrates how to create an SSL environment for a server System SSL program supporting TLS Version 1.3. TLS Version 1.3 programs must use 4-character cipher specifications and specify the TLS V1.3 cipher specifications in the GSK_V3_CIPHER_SPECS_EXPANDED. The GSK_SERVER_TLS_KEY_SHARES setting must be specified for TLS Version 1.3 server programs.

```
gsk_handle env_handle;
int rc;
/* create the SSL environment */
rc = gsk_environment_open(&env_handle);
/* set environment attributes */
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV2, GSK_PROTOCOL_SSLV2_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV3, GSK_PROTOCOL_SSLV3_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1, GSK_PROTOCOL_TLSV1_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_1, GSK_PROTOCOL_TLSV1_1_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_2, GSK_PROTOCOL_TLSV1_2_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_3, GSK_PROTOCOL_TLSV1_3_ON);
```

```
/* By default, TLS V1.3 protocol is set off */
rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_FILE, "/keyring/key.kdb",0);
rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_PW, "password",0);
rc = gsk_attribute_set_buffer(env_handle, GSK_SERVER_TLS_KEY_SHARES, "00230024", 0);
rc = gsk_attribute_set_enum(env_handle, GSK_V3_CIPHERS, GSK_V3_CIPHERS_CHAR4);
rc = gsk_attribute_set_buffer(env_handle, GSK_V3_CIPHER_SPECS_EXPANDED, "130313021301", 0);
/* initialize environment */
rc = gsk_environment_init(env_handle);
```

This example code illustrates how to create an SSL environment for a client System SSL program supporting TLS Version 1.2 and TLS Version 1.3. TLS Version 1.3 programs must use 4-character cipher specifications and specify the TLS V1.3 cipher specifications in the GSK_V3_CIPHER_SPECS_EXPANDED. The GSK_CLIENT_TLS_KEY_SHARES setting must be specified for TLS Version 1.3 client programs. When enabled for other protocols in addition to TLS Version 1.3, the cipher specifications must include at least one cipher that is allowed with those protocols.

```
gsk_handle env_handle;
int rc;
/* create the SSL environment */
rc = gsk_environment_open(&env_handle);
/* set environment attributes */
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV2, GSK_PROTOCOL_SSLV2_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_SSLV3, GSK_PROTOCOL_SSLV3_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1, GSK_PROTOCOL_TLSV1_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_1, GSK_PROTOCOL_TLSV1_1_OFF);
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_2, GSK_PROTOCOL_TLSV1_2_ON);
/* By default, TLS V1.2 protocol is set off */
rc = gsk_attribute_set_enum(env_handle, GSK_PROTOCOL_TLSV1_3, GSK_PROTOCOL_TLSV1_3_ON);
/* By default, TLS V1.3 protocol is set off */
rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_FILE, "/keyring/key.kdb",0);
rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_PW, "password",0);
rc = gsk_attribute_set_buffer(env_handle, GSK_CLIENT_TLS_KEY_SHARES, "0023", 0);
rc = gsk_attribute_set_enum(env_handle, GSK_V3_CIPHERS, GSK_V3_CIPHERS_CHAR4);
rc = gsk_attribute_set_buffer(env_handle, GSK_V3_CIPHER_SPECS_EXPANDED, "130313021301002F0035",
0);
/* initialize environment */
rc = gsk_environment_init(env_handle);
```

Note: When the environment is initialized, the environment attributes cannot be changed unless they are also attributes of the secure socket connection. In this case, they can be changed only for that connection. If changes are necessary to the environment, a new SSL environment can be created within the same process.

When the System SSL program successfully creates the SSL environment, it must now perform the steps that are needed to allow the program to communicate with a peer program. The exact sockets and System SSL calls required to allow the program to communicate differ depending on whether the program is a client or a server.

System SSL server program

You can use these sockets and System SSL calls to enable a server program to communicate with a client program.

To create a stream socket to which client programs can connect, use this function call:

```
    int server_sock;

    server_sock = socket(AF_INET, SOCK_STREAM, 0);
```

Now that the server program socket is created, bind the socket to a port (for example, 1234) that is known to the client program using this function call:

```
    int rc;
    int namelength;
    struct sockaddr_in name;

    nameLength = sizeof(name);
    memset(&name, ’\0’, nameLength);
    name.sin_family = AF_INET;
    name.sin_port = 1234;
    name.sin_addr.s_addr = INADDR_ANY;
```

```
rc = bind(server_sock, (struct sockaddr *)&name, nameLength);
```

To make the server program socket ready to listen for incoming connection request, use this function call:

```
int rc;

rc = listen(server_sock, 5);  /* allow max of 5 connections */
```

The server program is now ready to begin accepting connections from client programs. To accept connections, use these function calls:

```
int client_sock;
int incomingNameLength;
struct sockaddr_in incomingName;

client_sock = accept(server_sock, (struct sockaddr *)incomingName, &incomingNameLength);
```

After successfully accepting a connection from a client program, the server program must establish the secure socket connection which will result in the SSL handshake being performed. Once the handshake is completed, secure transfer of application data can be done. The secure socket connection is established with these attribute values:

- The socket descriptor over which the communication is to occur.
- Certificate with label "ServerCertLabel"
- The type of handshake (for example, server) to be performed.
- The set of SSL protocol cipher specifications to be allowed for the secure session specified using 4-character cipher specifications. (For example, ciphers utilizing a RSA key exchange with either AES 128/256 or 3DES encryption.) The cipher is selected by the System SSL server program according to the server's order of usage preference.
- The 4-character cipher specification list in GSK_V3_CIPHER_SPECS_EXPANDED is used.
- The address of a routine to be called by System SSL to read data from the socket for the secure session.
- The address of a routine to be called by System SSL to write data on the socket for the secure session.

```
gsk_handle  soc_handle;
int rc;
gsk_iocallback local_io = {secureSocRecv, secureSocSend, NULL, NULL, NULL, NULL};

rc = gsk_secure_socket_open(env_handle, &soc_handle);

rc = gsk_attribute_set_numeric_value(soc_handle, GSK_FD, client_sock);
rc = gsk_attribute_set_buffer(soc_handle, GSK_KEYRING_LABEL, "ServerCertLabel",0);
rc = gsk_attribute_set_enum(soc_handle, GSK_SESSION_TYPE, GSK_SERVER_SESSION);
rc = gsk_attribute_set_buffer(soc_handle, GSK_V3_CIPHER_SPECS_EXPANDED, "0035002F000A",0);
rc = gsk_attribute_set_enum(soc_handle, GSK_V3_CIPHERS, GSK_V3_CIPHERS_CHAR4);
rc = gsk_attribute_set_callback(soc_handle, GSK_IO_CALLBACK, &local_io);

rc = gsk_secure_socket_init(soc_handle);
```

The System SSL program should provide the function to send and receive data over the application socket. For more information, see "I/O routine replacement" on page 41. Use these function calls, **send()** and **recv()**, to send and receive the application data.

```
int secureSocRecv(int fd, void *data, int len, char *user_data) {
    return( recv( fd, data, len,0 ));
}

int secureSocSend(int fd, void *data, int len, char *user_data) {
    return( send( fd, data, len,0 ));
}
```

After the server program successfully calls **gsk_secure_socket_init()**, it can now read and write data securely over the application socket. To read application data from the application socket, use this code:

```
int rc;
int buffer_length;
int length_read;
char *data_buffer;
```

```
    rc = gsk_secure_socket_read(soc_handle, data_buffer, buffer_length, &length_read);
```

To write application data over the application socket, use this code:

```
    int rc;
    int buffer_length;
    int length_written;
    char *data_buffer;

    rc = gsk_secure_socket_write(soc_handle, data_buffer, buffer_length, &length_written);
```

Once the server program is finished using the application socket to securely send and receive data, it must free all of the System SSL resources for the SSL session and close the socket. To free the System SSL resource for the SSL session, use the **gsk_secure_socket_close()** call:

```
    gsk_secure_socket_close(&soc_handle);
```

To free the resources used by the SSL environment, use the **gsk_environment_close()** call:

```
    gsk_environment_close(&env_handle);
```

Finally, to close the application socket, use this function call:

```
    int rc;
    rc = close(client_sock);
```

System SSL client program

The socket and System SSL API calls used by the client program are very similar to the calls used by the server program. Rather than accepting connections like a server program, a client program connects to the server program.

To create a stream socket that the client program can use to connect to the server, use this function call:

```
    int sock;

    sock = socket(AF_INET, SOCK_STREAM,0);
```

Now that the client program socket is created, connect the socket to the server program port using this function call:

```
    int rc;
    int namelength;
    struct sockaddr_in name;
    char *ServeHostName;

    nameLength = sizeof(name);
    memset(&name, '\0', nameLength);
    name.sin_family = AF_INET;
    name.sin_port = 1234;
    name.sin_addr.s_addr = ServerHostName;
    rc = connect(sock, (struct sockaddr *)&name, nameLength);
```

After successfully connecting to the server program, the client program must establish the secure socket connection. This connection causes the SSL handshake to be performed. Once the handshake is complete, secure communication of the application data can be done. This example code establishes the connection using these attribute values:

- The socket descriptor over which the communication is to occur.
- Certificate with label "THELABEL"
- The type of handshake (client) to be performed.
- The set of SSL protocol cipher specifications to be allowed for the secure session in client-preferred order specified using 4-character cipher specifications. (For example, ciphers utilizing a RSA key exchange with either AES 128/256 or 3DES encryption.)

Note: Although the client is allowed to specify a preference order, an SSL server might not accept the preference.

- The 4-character cipher specification list in GSK_V3_CIPHER_SPECS_EXPANDED is used.
- The address of a routine to be called by System SSL to read data from the socket for the secure session.
- The address of a routine to be called by System SSL to write data on the socket for the secure session.

```
int rc;
gsk_handle  soc_handle;
gsk_iocallback local_io = {secureSocRecv, secureSocSend, NULL, NULL, NULL, NULL};

rc = gsk_secure_socket_open(env_handle, &soc_handle);
rc = gsk_attribute_set_numeric_value(soc_handle, GSK_FD, sock);
rc = gsk_attribute_set_buffer(soc_handle, GSK_KEYRING_LABEL, "THELABEL",0);
rc = gsk_attribute_set_enum(soc_handle, GSK_SESSION_TYPE, GSK_CLIENT_SESSION);
rc = gsk_attribute_set_buffer(soc_handle, GSK_V3_CIPHER_SPECS_EXPANDED, "0035002F000A",0);
rc = gsk_attribute_set_enum(soc_handle, GSK_V3_CIPHERS, GSK_V3_CIPHERS_CHAR4);
rc = gsk_attribute_set_callback(soc_handle, GSK_IO_CALLBACK, &local_io);

rc = gsk_secure_socket_init(soc_handle);
```

The System SSL program should provide the function to send and receive data over the application socket. For more information, see "I/O routine replacement" on page 41. Use these function calls, **send()** and **recv()**, to send and receive the application data.

```
int secureSocRecv(int fd, void *data, int len, char *user_data) {
   return( recv( fd, data, len,0 ));
}

int secureSocSend(int fd, void *data, int len, char *user_data) {
   return(send( fd, data, len,0 ));
}
```

After the client program successfully calls **gsk_secure_socket_init()**, it can now read and write data securely over the application socket. To read application data from the application socket, use this code:

```
int rc;
int buffer_length;
int length_read;
char *data_buffer;

rc = gsk_secure_socket_read(soc_handle, data_buffer, buffer_length, &length_read);
```

To write application data over the application socket, use this code:

```
int rc;
int buffer_length;
int length_written;
char *data_buffer;

rc = gsk_secure_socket_write(soc_handle, data_buffer, buffer_length, &length_written);
```

Once the client program is finished using the application socket to securely send and receive data, it must free all of the System SSL resources for the SSL session and close the socket.

To free the System SSL resource for the SSL session, use the **gsk_secure_socket_close()** call:

```
gsk_secure_socket_close(&soc_handle);
```

To free the resources used by the SSL environment, use the **gsk_environment_close()** call:

```
gsk_environment_close(&env_handle);
```

Finally, to close the application socket, use this function call:

```
int rc;
rc=close(sock);
```

Building a z/OS System SSL application

1. Write the System SSL source program (see "Writing a System SSL source program" on page 29).
2. Compile your System SSL source program using the DLL compiler option.
3. Include the `/usr/lib/GSKSSL.x` or `/usr/lib/GSKSSL64.x` sidedeck in the prelink or bind step input.

   If using the Certificate Management APIs, include either the `/usr/lib/GSKCMS31.x` or `/usr/lib/GSKCMS64.x` sidedeck in the prelink or bind step input.
4. Build a key database file or z/OS PKCS #11 token using the **gskkyman** utility, create a SAF key ring or PKCS #11 token using the RACDCERT command, add or utilize an existing GSKIT CMS V4 key database file, or utilize an existing PKCS #12 file. The name of the key database file, PKCS #12 file, z/OS PKCS #11 token, or SAF key ring must match the name you specified as the GSK_KEYRING_FILE on the **gsk_attribute_set_buffer()** API. For key database files, you need to specify either the password associated with the key file or the stash file name. For PKCS #12 files, you need to specify the password associated with the file. The password must match the password specified on GSK_KEYRING_PW on the **gsk_attribute_set_buffer()** API or must be set to NULL if using a SAF key ring or z/OS PKCS #11 token. Note that the password is case-sensitive. See Chapter 10, "Certificate/Key management," on page 541 for information about how to create a key database file, SAF key ring, or z/OS PKCS #11 token.

Running a z/OS System SSL application

After successfully writing and building the System SSL application and creating the certificate repository, you can run the System SSL application. To run the application follow these steps:

1. Ensure that pdsename.SIEALNKE, the PDSE that contains the System SSL DLLs, is in the MVS™ search order. If it is not in the linklist or LPA, you can use the STEPLIB DD statement in your JCL or the STEPLIB environment variable in the shell. For example, in the z/OS shell, issue this command:

   ```
   export STEPLIB=$STEPLIB:pdsename.SIEALNKE
   ```
2. Ensure that the key database file, GSKIT CMS V4 file, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token is accessible to the System SSL application.
3. System SSL uses malloc() and free() operations and may benefit from the enablement of heap pools. For information on using heap pools and tuning of heap pools, see 'Stack and heap storage' in *z/OS Language Environment Programming Guide*.
4. Run the System SSL application.

Note:

1. SSL applications must be run from within a POSIX environment.
2. Once SSL applications call **gsk_initialize()** or **gsk_environment_open()**, they cannot destroy the LE environment.
3. SSL applications must call SSL APIs from a C program, as they are C APIs.

Chapter 6. System SSL application programming considerations

When designing a System SSL SSL/TLS application, there are several key functional capabilities to consider:

Will the application need to communicate with other applications using non-blocking I/O?
The socket connections used for communication between System SSL applications are, by default, blocking. An application attempting to read or write to a socket is blocked until all expected data is received. This might not be desirable, because no other processing can occur while the application is waiting for a read or write to complete. See "Non-Blocking I/O" on page 38 for additional information.

Will the application need to prompt the client user to select a certificate from a list during the client authentication process in the SSL handshake?
This behavior, if needed, can be accomplished using a registered callback routine that is invoked from inside the **gsk_secure_socket_init()** function call. See "Client authentication certificate selection" on page 40 for additional information.

Will the application need to override System SSLs default I/O callback routines to specify I/O behavior?
This can be accomplished by specifying your own callback routines for receiving and sending data. See "I/O routine replacement" on page 41 for additional information.

Will application-specific data need to be available to the SSL callback routines?
If needed, application-specific data can be made available using the **gsk_attribute_set_buffer()** and **gsk_attribute_get_buffer()** function calls. See "Use of user data" on page 41 for additional information.

Considering both security and performance benefits, how long should SSL sessions be allowed to remain active?
Security conscious applications should keep the session timeout values very low to ensure keys are generated frequently to avoid security breaches. Applications that are more performance conscious than security conscious should have longer session timeout values and a larger cache size. See "Session ID (SID) and session ticket cache" on page 42 for additional information.

Will the application need to initiate session renegotiation for a SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 connection?
If needed, the application can call the **gsk_secure_socket_misc** API to renegotiate the communications session to establish a new session key or have the session cipher reset. Notification callback routines allow the application to take specific actions during a session renegotiation. See "Session renegotiation notification (SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2)" on page 46 for additional information.

Will the application need to add functionality to the Transport Layer Security (TLS) protocol?
Applications can define a TLS extension to the SSL environment or connection by calling the **gsk_attribute_set_tls_extension()** function. See "TLS extensions" on page 46 for additional information.

Will a "Suite B Compliant" TLS V1.2 session be required?
System SSL allows TLS client and server applications to specify a profile compliant with Suite B Cryptography as defined in *RFC 5430: Suite B Profile for Transport Layer Security (TLS)* (RFC 5430 (tools.ietf.org/html/rfc5430)) or *RFC 6460: Suite B Profile for Transport Layer Security (TLS)* (RFC 6460 (tools.ietf.org/html/rfc6460)). This profile restricts the cryptographic algorithms used for the session to the set of algorithms supported by Suite B Cryptography. See "Suite B cryptography support" on page 49 for additional information.

Will certificate revocation be needed when validating the partner's certificate?
Applications can enable the usage of revocation information obtained through OCSP responses, HTTP CRLs, or LDAP CRLs. See "SSL/TLS partner certificate revocation checking" on page 52 for additional information.

Will the server application need to supply the OCSP response for the server's end entity certificate during the handshake process?
The server application can enable OCSP server stapling for this support. See "Enabling OCSP server stapling" on page 57 for additional information.

Will the server application need to provide more than one certificate label to communicate with more than one client?
More than one certificate label ensures that an appropriate certificate is available for the selected cipher. See "Using server multiple key label support" on page 59 for additional information.

Will the application need to enable TLS V1.3 protocol support?
System SSL allows client and server applications to enable for the TLS V1.3 protocol as defined in *RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3* RFC 8446 (tools.ietf.org/html/rfc8446). See "TLS V1.3 protocol support" on page 61 for additional information.

Does an existing application support SSL/TLS protocols prior to TLS V1.2?
An existing System SSL server or client application can be updated to support TLS V1.2. See "Upgrading to TLS V1.2 from earlier SSL and TLS protocols" on page 64 for additional information.

Does an existing application support TLS V1.2 and needs to be updated to also support TLS V1.3?
An existing System SSL server or client application that currently supports TLS V1.2 can be updated to support both TLS V1.2 and TLS V1.3. See "Upgrading from TLS V1.2 to TLS V1.2 and TLS V1.3 protocols" on page 67 for additional information.

Will the application want additional diagnostic information for peer certificate failure validations?
This can be accomplished through a registered callback routine that is invoked from inside the **gsk_secure_socket_init()** function when validation of the peer's certificate or certificate chain fails. For additional information, see "Certificate diagnostic callback routine" on page 41.

Will the application need to limit the elliptic curve utilized during the handshake key exchange process?
System SSL provides the capability for a client or server to limit the key exchange elliptic curves. For additional information, see "Limiting key exchange elliptic curves" on page 69.

Will server certificate domain name validation be needed by the SSL V3/TLS client?
TLS client applications can enable domain name validation using the server certificate's DNS entry for the subject alternative name or subject DN common name. For additional information, see "Server certificate domain-based validation" on page 70.

Non-Blocking I/O

Applications wanting to communicate securely to one another may establish a secure connection. Each application opens a socket and attempts to establish an SSL connection. After an SSL connection is established, the applications may now use the socket to exchange data securely. The default (blocking) mode of a socket requires an application attempting to read or write to the socket to block until all expected data is received. This blocking may not be desirable since no other processing may occur while the application is waiting for a read or write to complete. One solution to this problem is the use of non-blocking sockets.

When a socket is set up as non-blocking, reads and writes to the socket do not cause the application to block and wait. Instead the read or write function will read/write only the data currently available (if any). If the entire read/write is not completed, a status indicator is returned. The application might try read/write again later.

Non-Blocking socket primer

When a server wants to communicate with clients by using a socket, these routines are used:

Table 9. Server communicating with clients by way of a socket

Routine	Purpose
1. socket()	Create a socket
2. bind()	Register the socket
3. listen()	Indicate willingness to accept connections
4. accept()	Accept a connection request
5. Read request	
6. Write response	
7. Return to step 4	

Once the **accept()** routine is called, the server blocks until data is available for the socket. Problems arise when the server wants to monitor multiple sockets simultaneously or if the server wants to perform other tasks until data is available on the socket. However, by configuring the socket as non-blocking, these problems may be avoided. For more information, see "Enable/disable non-blocking mode" on page 40. When using non-blocking sockets, the **select()** routine is used to instruct the system to notify the server application when data is available on a particular socket.

Table 10. Using the select() routine

Routine	Purpose
1. socket()	Create a socket
2. bind()	Register the socket
3. listen()	Indicate willingness to accept connections
4. Set socket as non-blocking	See "Enable/disable non-blocking mode" on page 40
5. select()	Monitor a number of sockets
6. accept()	Accept a connection request
7. Read request	If unable to read all data, return to step 5
8. Write response	If unable to write all data, return to step 5
9. Return to step 4	

Affected SSL functions

These functions are affected by the use of non-blocking sockets with SSL.

gsk_secure_socket_init()
During the SSL handshake, the **io_setsocketoptions()** routine is called by the **gsk_secure_socket_init()** routine before initiating the SSL handshake (GSK_SET_SOCKET_STATE_FOR_HANDSHAKE) and again upon completion of the SSL handshake (GSK_SET_SOCKET_STATE_FOR_READ_WRITE). The default **io_setsocketoptions()** routine puts the socket into blocking mode for GSK_SET_SOCKET_STATE_FOR_HANDSHAKE and restores the original mode for GSK_SET_SOCKET_STATE_FOR_READ_WRITE. In order to perform a non-blocking SSL handshake, an application supplied **io_setsocketoptions()** callback must be provided to control the state of the socket. When the socket is in non-blocking mode, **gsk_secure_socket_init()** may return GSK_WOULD_BLOCK_READ or GSK_WOULD_BLOCK_WRITE. This error indicates that System SSL was unable to read or write the entire message. When this occurs, the application should call **select()** and then call **gsk_secure_socket_init()** again.

gsk_secure_socket_read()
Once the socket is configured as non-blocking, any calls to **gsk_secure_socket_read()** can potentially return GSK_WOULD_BLOCK. When this occurs, the application should call **select()** and then call **gsk_secure_socket_read()** again.

gsk_secure_socket_write()
Once the socket is configured as non-blocking, any calls to **gsk_secure_socket_write()** can potentially return GSK_WOULD_BLOCK. When this occurs, the application should call **select()** and then call **gsk_secure_socket_write()** again.

Enable/disable non-blocking mode

Once a socket is created using the **socket()** call, it may be set to non-blocking as follows:

```
#include "sys/ioctl.h"
int on =1;
int off =0;

//Enable non-blocking
ioctl (mySocket, FIONBIO, &(on));
//Disable non-blocking
ioctl (mySocket, FIONBIO, (char *) &(off));
```

Differences in SSL and unsecured non-blocking mode

Partial Data
An unsecured socket in non-blocking mode returns the partial data received or written. Since System SSL processes encrypted data, it is not possible to decrypt a message until the entire message is received, making it impossible to return partial data.

Error Indicator
When non-blocking mode is used on a non-secure socket, the status indicator is generally found by checking the *errno* variable, which is normally EWOULDBLOCK. System SSL does **not** set the *errno* variable. Instead the value returned from **gsk_secure_socket_read()** or **gsk_secure_socket_write()** is set to GSK_WOULD_BLOCK. **gsk_secure_socket_init()** returns either GSK_WOULD_BLOCK_READ or GSK_WOULD_BLOCK_WRITE.

Client authentication certificate selection

SSL enables the application to prompt the client user to select a certificate from a list during the client authentication process in the SSL handshake.

This is accomplished with a registered callback routine that is invoked from inside the **gsk_secure_socket_init()** function call. This topic provides an overview of that code.

The client application code must provide these functions:

- Register a standard C linkage callback routine using the **gsk_attribute_set_callback()** function call.
- Implement the callback routine that performs these functions:
  - Get the list of available certificates using the **gsk_attribute_get_data()** function call with the GSK_DATA_ID_SUPPORTED_KEYS option. This returns a list of labels from the key data base file, SAF key ring, or z/OS PKCS #11 token.
  - Display the list of labels to the user.
  - Prompt the user to select the label from the list
  - Set the label to be used with a **gsk_attribute_set_buffer()** function call with the GSK_KEYRING_LABEL option.
  - Return to SSL with the return value set to indicate use client authentication.
  - If the user elects to not use any of the certificates in the list, return with the value set to skip client authentication. A certificate is not sent to the partner, but the SSL handshake completes. The server decides whether to continue or close the connection.

- Optionally, the application can display certificate information using the **gsk_get_cert_by_label()** function call.
- Optionally, the application can use the **gsk_attribute_get_data()** function call with the GSK_DATA_ID_SERVER_ISSUERS option to display a list of server signer certificates.

I/O routine replacement

Callback routine for I/O

SSL allows applications to specify how I/O is to take place. This is done by specifying callback routines for receiving and sending data. The contents of this routine can be very unique per application. SSL has an internally defined default routine which is used if **gsk_attribute_set_callback()** is not used to override I/O routines. The default assumes that TCP/IP is being used. For reading it executes a **recv()** and for write a **send()**. If not using TCP/IP, applications should also consider the specification of the **getpeername** and **setsocketoptions** callback routine. It also depends on TCP/IP as being the transport layer protocol.

Note: Application provided I/O routines must use standard C linkage conventions.

Use of user data

Some complex applications require application-specific data to be available in the SSL callbacks. SSL enables this with the **gsk_attribute_set_buffer()** and **gsk_attribute_get_buffer()** function calls. In addition, the I/O callbacks and the certificate diagnostic callback pass a pointer to the user data.

These are the steps that need to be taken to effectively use the user data functions:

- Issue the **gsk_secure_socket_open()** function. This returns a soc_handle.
- To set the user data for a connection issue:
  - **gsk_attribute_set_buffer(***soc_handle***, GSK_USER_DATA,** *user_data, sizeof***(***user_data***));**
  - This function call copies the *user_data* into an area of storage owned by SSL.
- The address of the SSL copy of the user data is passed as a parameter to the user-specified **read**, **write**, **getpeername**, **set_socket_options**, and **certificate diagnostic** callbacks.
- Other callbacks pass the *soc_handle* as a parameter to the callback. To find the address of the copy of user data associated with a particular connection, issue:
  - **gsk_attribute_get_buffer(***soc_handle***, GSK_USER_DATA,** *&user_data_ptr***,** *&user_data_size***);**
  - You can modify the contents of the SSL copy of the user data, but you may not free or re-allocate the SSL user data. The SSL user data is freed when the connection is closed with the **gsk_secure_socket_close()** function call.

You can point to other application data from the SSL user data area. However, it is up to the application to free this other application data before the connection is closed.

Certificate diagnostic callback routine

Callback routine to be used to provide certificate validation information back to the application during the establishment of the SSL V3 or TLS secure connection, **gsk_secure_socket_init()** function. The provided diagnostic information is related to the validation of the peer's certificate and certificate chain and is defined by the gsk_diag_summary structure. The first certificate in the returned gsk_cert_diag_details is the peer certificate. Each subsequent certificate is the next certificate in the chain. The *failIndex* field identifies the index value of the failing certificate in the array (index value of 1 identifies the peer certificate). The *descText* field contains a brief description of the cause of the failure. This information is not translated and is meant to be used only for diagnostics.

Each gsk_cert_diag_detail structure contains fields that provide basic information about the certificate, such as the *subjectDN* and *issuerDN*. The gsk_cert_diag_detail structure also provides the DER-encoded certificate in the *derCert* field. Additional details about the certificate can be made available by decoding the contents of the *derCert* field into an X.509 certificate using the **gsk_decode_certificate()** API.

Additionally, the **gsk_cert_diagnostic_callback** routine is passed a diagnostic string. This string is not translated and is only intended to serve as a consolidated subset form of the diagnostic information contained in the gsk_diag_summary structure. This string contains information for a single certificate, which represents the failing certificate (or the peer's end-entity certificate when successful). The diagnostic string includes the SSL return code, the CMS return code, the certificate's Subject and Issuer DNs (enclosed in angle brackets <> to allow for blanks to be present in the names and formatted using **gsk_name_to_dn()**), serial number, a descriptive text of the validation result or error that occurred, and the trusted source used for the validation process. If the validation was unsuccessful, the diagnostic string also contains the source from which the failing certificate was retrieved. The diagnostic string is intended for diagnostic display purposes only, and as such, the information contained in this string can vary from release to release. Therefore, it is recommended that exploiters only process and/or parse data contained in the gsk_diag_summary structure and avoid creating any dependencies on the diagnostic string.

```
void cert_diagnostic_callback (
                               gsk_handle                soc_handle,
                               gsk_diag_summary *        diag_summary,
                               char *                    diag_string,
                               char *                    user_data);
```

By default, the diagnostic callback routine is called only when certificate validation fails. To have the diagnostic callback routine called for successful certificate validations or both successful and failure certificate validations, GSK_CERT_DIAG_INFO must be set. For information, see description of GSK_CERT_DIAG_INFO in "gsk_attribute_set_enum()" on page 112 and Appendix A, "Environment variables," on page 763.

The callback routine must not modify the contents of the diag_summary structure. Once control is returned from the diagnostic callback routine, the diagnostic information will be cleared and freed. The callback routine is responsible for processing the information prior to returning.

If an out-of-storage condition is encountered during the **gsk_secure_socket_init()** routine processing of the peer certificate, the callback routine is not called.

Session ID (SID) and session ticket cache

The SSL and TLS protocols have a mechanism built in to allow for faster secure connections between a client/server pair. For the SSL and TLS protocols, there is a concept of an SSL/TLS session that allows this to happen.

For protocols prior to TLS V1.3, the first time a client and server connect, cryptographic characteristics of that connection are saved into a session cache entry on both the client and server sides. A session is identified by a Session ID (SID). On the client side, the session cache entry contains a mapping to the server and port number to the session ID. On the server side, the session cache entry contains the session ID along with the state of the initial full handshake. The cached cryptographic components (SID cache entry) allows for new bulk encryption keys to be generated with subsequent SSL handshakes between the same client/server pair. When the client attempts to resume an SSL V2 through TLS V1.2 handshake, it searches the session cache for the server's IP address and port number for the Session ID. If a Session ID is found, the client transmits it on the subsequent handshake. When the server receives the subsequent handshake, it looks in its cache for the Session ID specified by the client. If the server finds the Session ID in its cache, the server obtains the keys from the initial handshake in the cache. The subsequent handshakes will be abbreviated because much of the data used to generate keys is in the SID cache entry. This abbreviated handshake does not require public key encryption to take place.

For the TLS V1.3 protocol, the SSL/TLS sessions are stored and communicated in a different manner. After a successful TLS V1.3 handshake completes in **gsk_secure_socket_init()**, the server may send one or more session tickets to the client. The client application must call **gsk_secure_socket_read()** to read in any session tickets that the server may have sent after the handshake has completed. Client applications should use non-blocking sockets when calling the **gsk_secure_socket_read()** routine when reading in session tickets from the server. The use of a blocking socket may cause the **gsk_secure_socket_read()** routine to hang if the server does not send any session tickets. By default, the session tickets sent by a System SSL server application are encrypted with a secret key and contain information to allow the server to fully resume the prior established TLS V1.3 session. With these encrypted session tickets, the server no longer maintains session state information in its own session cache entry. If sysplex ticket caching is enabled in the System SSL server application with the GSK_SYSPLEX_SESSION_TICKET_CACHE option set to ON, the session and ticket information is stored in cache entries that reside in the System SSL started task, GSKSRVR. The GSKSRVR must be up and running to allow this to occur; otherwise, encrypted session tickets are sent by the System SSL server application. If the TLS V1.3 session is successfully cached in the GSKSRVR, each ticket contains a unique identifier which is used for cache lookups. On the client side, the session cache entry contains a mapping to the server and port number with all the session tickets associated with that TLS V1.3 session. If the client decides to resume a previous TLS V1.3 session to the same server, the client looks in its Session ID and session ticket cache for a session ticket. If a session ticket is found, it is removed from the cache to prevent ticket reuse. The client includes that session ticket and derives a pre-shared key from that session ticket to include in its subsequent handshake messages. The server parses the pre-shared key included in the client's handshake message and validates the session ticket that the client has included. If the session ticket is valid, the server indicates that the TLS V1.3 connection was successfully resumed by indicating that in the server's handshake messages. At that point, an abbreviated handshake is done by the server which avoids having public key encryption to take place.

Public key encryption is very time consuming so avoiding it, improves performance for clients and servers using SSL and TLS. A SID cache entry exists for a limited time. Take care when specifying how long an SSL/TLS session ID or session tickets are allowed to live. Setting the SID cache timeout or number of SID cache entries to ZERO turns off SID and session ticket caching and causes a full handshake to be completed for every connection. Applications need to be sensitive to both security and performance issues. Security conscious applications should keep the session timeout values very low to ensure keys are generated frequently to avoid security breaches. Applications that are more performance conscious than security conscious should have longer session timeouts and a larger cache size.

Session ID (SID) and session tickets

Session ID (SID) and session ticket caching for the client is done internally within the client's address space and each SSL environment has its own cache. The server can either cache session IDs for SSL V3 through TLS V1.2 sessions within its address space per SSL environment or externally through the GSKSRVR for sysplex session ID caching. For TLS V1.3 sessions, the server by default does not locally cache session tickets sent after a successful TLS V1.3 handshake. If the server is enabled for sysplex session ticket caching by setting the GSK_SYSPLEX_SESSION_TICKET_CACHE to ON, session tickets are allowed to be cached in the GSKSRVR. Full session information is stored pertaining to each uniquely issued session ticket.

Sysplex session ID and sysplex ticket caching allows session information to be shared among like-servers or processes on the same system or on different systems within a sysplex. These like-server applications have the same configurations. To take full advantage of sysplex session ticket caching of TLS V1.3 sessions, all like-server applications must be running on z/OS V2R5 systems with the PTFs for APAR OA63252 applied or later releases. If not, additional full TLS V1.3 handshakes may occur.

See "Sysplex session cache support" on page 615 and "Sysplex session ticket cache support" on page 618 for more information.

Modifying SSL session caching parameters can help tune the security performance characteristics of SSL and TLS enabled servers and clients. The contents of the internal client and server caches are controlled by the setting of an expiration lifetime for an SSL session ID entry and the number of entries that can reside concurrently in the cache. Separate caches exist for SSL V2 and SSL V3 (TLS) sessions. The internal SSL SID cache is fixed to a configurable number of entries defined when the SSL environment is being established.

By default, the SSL V2 cache size is 256 entries and can be modified through the GSK_V2_SIDCACHE_SIZE environment attribute. The default expiration (or timeout) is 100 seconds and can be modified through the GSK_V2_SESSION_TIMEOUT environment attribute. By default, the

SSL V3 (TLS) cache size is 512 entries and can be modified through the GSK_V3_SIDCACHE_SIZE environment attribute. The default expiration (or timeout) is 24 hours and can be modified through the GSK_V3_SESSION_TIMEOUT environment attribute. By default, the client stores session tickets received from a server after a successful TLS V1.3 handshake completes and allows for TLS V1.3 session resumption. This is controlled by the GSK_SESSION_TICKET_CLIENT_ENABLE environment attribute. The client can configure the maximum number of session tickets stored per TLS V1.3 session by setting the GSK_SESSION_TICKET_CLIENT_MAXCACHED, which defaults to 8. The GSK_SESSION_TICKET_CLIENT_MAXSIZE environment attribute indicates the maximum size in bytes of a session ticket that can be stored in the client's session ticket cache. If the session ticket size is greater than the specified GSK_SESSION_TICKET_CLIENT_MAXSIZE, the session ticket is not stored in the client cache. Each session ticket issued by the server has a defined lifetime of up to seven days. When the client receives and stores these tickets in its cache, it will take the more restrictive time setting of either the local GSK_V3_SESSION_TIMEOUT setting or the server's ticket lifetime setting. There is no way to remove entries for other connections except for repeated connections between the same client/server pair.

Each time a full handshake is performed and caching is active (cache size !=0), a SID cache entry is created and added to the cache. During the add process, detected expired SID entries and session tickets are removed. If the cache reaches its size limit, an entry is removed from the cache and the newly created SID cache entry is added.

After the server successfully completes a TLS V1.3 handshake, it may send session tickets to the client. These session tickets are used by the client to resume a previously established TLS V1.3 session. There are several System SSL environment attributes that can be used to tailor different aspects of the server issued session tickets.

GSK_SESSION_TICKET_SERVER_ENABLE
: This setting indicates if the server is enabled to send session tickets after successfully completing a TLS V1.3 handshake. By default, this option is set to ON.

GSK_SESSION_TICKET_SERVER_COUNT
: This setting determines the number of session tickets that should be sent after a full TLS V1.3 handshake completes. By default, this option is set to 2, but can be set to anywhere between 0 through 16 tickets. For each successfully resumed TLS V1.3 handshake, a single session ticket is sent to the client, provided the count is greater than zero. Alternatively, if this option is set to zero, the GSK_SEND_SESSION_TICKET option can be specified in a call to **gsk_secure_socket_misc()** after a successful TLS V1.3 handshake to send a single session ticket message per call.

GSK_SYSPLEX_SESSION_TICKET_CACHE
: This setting indicates if TLS V1.3 session and ticket information is cached within the System SSL started task, GSKSRVR, if it is running. By default, this option is set to OFF. When set to OFF, the session tickets sent by the server are encrypted and contain full session state to allow for TLS V1.3 session resumption to occur. When set to ON, the session tickets sent by the server contain a unique ticket identifier. This unique ticket identifier is used to perform cache lookups in the GSKSRVR session ticket cache when a client attempts to perform a TLS V1.3 session resumption.

The following server session ticket settings are relevant for encrypted session tickets sent by the server. By default, session tickets are encrypted by the server and contain full session information and is controlled by the GSK_SYSPLEX_SESSION_TICKET_CACHE environment attribute. The GSK_SESSION_TICKET_SERVER_ENABLE must be set to ON to allow for any session tickets to be sent. If the System SSL started task, GSKSRVR, is not running and GSK_SYSPLEX_SESSION_TICKET_CACHE is set to ON, encrypted session tickets are sent.

GSK_SESSION_TICKET_SERVER_ALGORITHM
: This setting indicates the algorithm used to encrypt session tickets.

GSK_SESSION_TICKET_SERVER_KEY_REFRESH
: This setting specifies the interval at which the encryption key used to encrypt session tickets is refreshed. The server retains the current and previous keys used to encrypt TLS V1.3 session tickets in the SSL environment handle. If an older session ticket is included by the client in a TLS V1.3 resumption attempt, this allows for an older session ticket to be decrypted by the server. If GSK_SESSION_TICKET_SERVER_KEY_REFRESH is set to 300 seconds (5 minutes), the current key can be used for 300 seconds to encrypt new session tickets. The current and previous keys can be

used to decrypt tickets included by a client while attempting to resume a TLS V1.3 session. Therefore, a TLS V1.3 session ticket can be successfully decrypted for up to 600 seconds (10 minutes) since the current and previous keys each live for 300 seconds.

The GSK_SESSION_TICKET_SERVER_TIMEOUT setting specifies the maximum time in seconds from the initial handshake that a session ticket will be accepted back from the client for a TLS V1.3 session resumption attempt. This setting can be any value from 0 through 604800 seconds (seven days). The default lifetime is 300 seconds (5 minutes). If the session information is successfully cached in the sysplex ticket cache in the GSKSRVR when sysplex ticket caching is enabled, the default lifetime is 600 seconds (10 minutes). The System SSL started task, GSKSRVR environment variable, GSK_SESSION_TICKET_CACHE_TIMEOUT, also plays a role in how long a sysplex ticket cache entry lives within the ticket cache. See “Sysplex session ticket cache support” on page 618 for more information about sysplex session ticket caching.

Note: Session resumption attempts are initiated by the client and as such, server attribute settings do not guarantee that resumption will occur.

Session ID cache replacement (SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2)

The list of options for extending SID caching functionality can become quite long so an external SID cache API was created for those who are more discriminating about managing SID cache data. There are several callbacks used for external SID cache access.

Note that there are probably few applications where using an external SID cache makes sense. Some suggested environments where it might be considered is in a server configuration where multiple instances of a server exist for workload balancing purposes. It might be desirable to have a single SID cache to be used by all of the processes which each server is running in. Usually this can be avoided by writing applications which are multi threaded. All threads would use the single internal SID cache buffer.

Format

```
typedef gsk_data_buffer *               (*ptgsk_getcache) (
                const unsigned char *   session_id,
                unsigned int            session_id_length,
                int                     ssl_version);

typedef gsk_data_buffer *               (*ptgsk_putcache) (
                gsk_data_buffer *       ssl_session_data,
                const unsigned char *   session_id,
                unsigned int            session_id_length,
                int                     ssl_version);

typedef void                            (*ptgsk_deletecache) (
                const unsigned char *   session_id,
                unsigned int            session_id_length,
                int                     ssl_version);

typedef void                            (*ptgsk_freecache) (
                gsk_data_buffer *       ssl_session_data);

typedef struct _gsk_sidcache_callback {
                ptgsk_getcache          Get;
                ptgsk_putcache          Put;
                ptgsk_deletecache       Delete;
                ptgsk_freecache         FreeDataBuffer;
} gsk_sidcache_callback;
```

Callbacks

Get
Specifies the routine System SSL calls to search the session ID cache for the entry that matches the passed values in **sessionID**, **sessionIDLen**, and **SSLVersion**. The value returned by this routine is a pointer to a malloc'ed **gsk_data_buffer** structure for the sslSessionData that contains the session id cache entry.

Put
Specifies the routine System SSL calls to add an entry to the session ID cache. The passed in values **sessionID**, **sessionIDLen**, **SSLVersion** and **sslSessionData** are used to define the entry. This routine is responsible for getting storage to hold the entry. The value returned by this routine is either NULL if unable to allocate storage or a pointer to a **gsk_data_buffer** structure containing the **sslSessionData** that was passed into the routine.

Delete
Specifies the routine System SSL calls to delete an entry from the session ID cache. **sessionID**, **sessionIDLen** and **SSLVersion** are used to determine which entry is deleted.

FreeDataBuffer
Specifies the routine that System SSL calls to free memory that was returned by the **Get** session id cache callback routine.

Parameters

sessionID
The buffer containing the Session data

sessionIDLen
The length of the entry for the SID cache buffer entry.

SSLVersion
The version of the SSL Protocol.

data
This is the buffer that is created by the external SID cache process to transfer the SID cache entry to SSL.

Session renegotiation notification (SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2)

SSL provides a mechanism to renegotiate the communications session to establish a new session key or have the session cipher reset. This can be initiated by either the SSL server or SSL client through the **gsk_secure_socket_misc()** API. System SSL allows applications to specify callback routines for receiving notifications when SSL is commencing and completing a session renegotiation. System SSL calls the specified routines and supply the connection handle for session identification, indicating that new session keys are being negotiated. This allows the user application to take specific actions during a session renegotiation, such as suspending application communications until the negotiation is complete.

To reset the session keys for a TLS V1.3 connection, see "Refreshing session keys" on page 64.

TLS extensions

System SSL allows applications to specify TLS extensions that add functionality to the Transport Layer Security (TLS) protocol. TLS extensions may be set by both TLS clients and servers. The use of TLS extensions is compatible with earlier versions: communication is possible between TLS clients that support TLS extensions and TLS servers that do not support TLS extensions, and vice versa.

To use TLS extensions in a TLS client/server session, the **gsk_attribute_set_tls_extension()** SSL API must be used to define the extensions that the TLS client or server supports. TLS extensions may be defined:

- After **gsk_environment_open()** is performed but before the **gsk_environment_init()** call
- After **gsk_secure_socket_open()** is performed but before the **gsk_secure_socket_init()** call

TLS extensions that are defined for an SSL environment applies to all connections within the environment. Each connection can define additional TLS extensions to be used for that connection, or may override TLS extension settings that are defined for the environment. System SSL currently provides support for the following TLS extensions:

Truncated HMAC

Truncates the HMAC used to authenticate record layer communications to 80 bits

Maximum Fragment Length

Allows the client to use a fragment length smaller than the TLS default of 16,384 bytes when transmitting messages

Server Name Indication

Allows the client to tell the server the name of the server it wants to connect to

Setting server side extensions

The following example illustrates how to define each of the supported System SSL TLS extensions for a TLS server. The extensions are defined at the environment level and are optional. Optional allows the TLS server to communicate with TLS clients that support the extensions, including TLS clients that do not support the extensions.

```
int                                  rc;
gsk_handle                           envHandle;

gsk_tls_extension                    tls_extn[3];
char                                 server1[] = "server1.ibm.com";
char                                 server2[] = "server2.ibm.com";
char                                 server3[] = "server3.ibm.com";
char                                 label1[] = "Server1 Certificate";
char                                 label2[] = "Server2 Certificate";
char                                 label3[] = "Server3 Certificate";
gsk_server_key_label                 serverLabelPairs[] = {{server1, label1},
                                                      {server2, label2},
                                                      {server3, label3}};

    /*
     * Open the SSL environment
     */
rc = gsk_environment_open(&envHandle);

    /*
     * Set truncated HMAC extension
     */
memset(&tls_extn[0], 0, sizeof(gsk_tls_extension));
tls_extn[0].extId = GSK_TLS_EXTID_TRUNCATED_HMAC;
tls_extn[0].required = FALSE; /* optional extension */
tls_extn[0].u.truncateHmac = TRUE; /* enable extension */
rc = gsk_attribute_set_tls_extension(envHandle, &tls_extn[0]);

    /*
     * Set maximum fragment length extension
     */
memset(&tls_extn[1], 0, sizeof(gsk_tls_extension));
tls_extn[1].extId = GSK_TLS_EXTID_SERVER_MFL;
tls_extn[1].required = FALSE;   /* optional extension */
tls_extn[1].u.maxFragmentLength = GSK_TLS_MFL_ON;
                                /* enable extension */
rc = gsk_attribute_set_tls_extension(envHandle, &tls_extn[1]);

    /*
     * Set server name indication extension
     */
memset(&tls_extn[2], 0, sizeof(gsk_tls_extension));
tls_extn[2].extId = GSK_TLS_EXTID_SNI_SERVER_LABELS;
tls_extn[2].required = FALSE;     /* optional extension */
tls_extn[2].u.serverLabels.setSni = TRUE;
                                   /* enable extension */
tls_extn[2].u.serverLabels.unrecognized_name_fatal = TRUE;
                                  /* unrecognized name is fatal */
tls_extn[2].u.serverLabels.count = 3;
tls_extn[2].u.serverLabels.serverKeyLabel = serverLabelPairs;
rc = gsk_attribute_set_tls_extension(envHandle, &tls_extn[2]);

    /*
     * Initialize the SSL environment
     */
rc = gsk_environment_init(envHandle);
```

Setting client side extensions

The following example illustrates how to define each of the supported System SSL TLS extensions for a TLS client. The HMAC and maximum fragment extensions are defined at the environment level. The server name indication extension is defined, while the HMAC extension is modified for a particular connection. The environment level extensions are being defined as required and connection level extensions as optional. Required extensions require that the partner TLS server support the specified TLS extensions. If it does not support the extensions, the TLS handshake fails.

```
int                                              rc;
gsk_handle                                       envHandle;
gsk_handle                                       conHandle;

gsk_tls_extension                                tls_extn_env[2];
gsk_tls_extension                                tls_extn_con[2];
char                                             server1[] = "server1.ibm.com";
char                                             server2[] = "server2.ibm.com";
char *                                           serverNames[] = {server1, server2};

    /*
     * Open the SSL environment
     */
rc = gsk_environment_open(&envHandle);

    /*
     * Set truncated HMAC extension
     */
memset(&tls_extn_env[0], 0, sizeof(gsk_tls_extension));
tls_extn_env[0].extId = GSK_TLS_EXTID_TRUNCATED_HMAC;
tls_extn_env[0].required = TRUE; /* required extension */
tls_extn_env[0].u.truncateHmac = TRUE; /* enable extension */
rc = gsk_attribute_set_tls_extension(envHandle,&tls_extn_env[0]);

    /*
     * Set maximum fragment length extension
     */
memset(&tls_extn_env[1], 0, sizeof(gsk_tls_extension));
tls_extn_env[1].extId = GSK_TLS_EXTID_CLIENT_MFL;
tls_extn_env[1].required = TRUE; /* required extension */
tls_extn_env[1].u.maxFragmentLength = GSK_TLS_MFL_4096;
                                 /* set 4096 bit fragment length */
rc = gsk_attribute_set_tls_extension(envHandle,&tls_extn_env[1]);

    /*
     * Initialize the SSL environment
     */
rc = gsk_environment_init(envHandle);

    /*
     * Open the SSL connection
     */
rc = gsk_secure_socket_open(envHandle, &conHandle);

    /*
     * Set server name indication extension
     */
memset(&tls_extn_con[0], 0, sizeof(gsk_tls_extension));
tls_extn_con[0].extId = GSK_TLS_EXTID_SNI_CLIENT_SNAMES;
tls_extn_con[0].required = FALSE;           /* optional extension */
tls_extn_con[0].u.clientSnameList.setSni = TRUE;
                                            /* enable extension */
tls_extn_con[0].u.clientSnameList.unrecognized_name_fatal = TRUE;
                                            /* unrecognized name is fatal */
tls_extn_con[0].u.clientSnameList.count = 2;
tls_extn_con[0].u.clientSnameList.serverNames = serverNames;
rc = gsk_attribute_set_tls_extension(envHandle,&tls_extn_con[0]);

    /*
     * Modify truncated HMAC extension
     */
memset(&tls_extn_con[1], 0, sizeof(gsk_tls_extension));
tls_extn_con[1].extId = GSK_TLS_EXTID_TRUNCATED_HMAC;
tls_extn_con[1].required = FALSE;         /* optional extension */
tls_extn_con[1].u.truncateHmac = TRUE;    /* enable extension */
rc = gsk_attribute_set_tls_extension(envHandle,&tls_extn_con[1]);

     /*
      * Initialize the SSL connection
```

```
          */
rc = gsk_secure_socket_init(conHandle);
```

Suite B cryptography support

System SSL allows TLS client and server applications to specify a profile compliant with Suite B Cryptography as defined in RFCs 5430 and 6460: *Suite B Profile for Transport Layer Security (TLS)*. The Suite B profile restricts the cryptographic algorithms that are used for the session to the set of algorithms that are supported by Suite B Cryptography. Communication is possible between TLS clients that require Suite B cryptography and TLS servers that do not explicitly support Suite B cryptography, and vice versa, provided the non-Suite B entity supports the Suite B compliant cryptographic algorithms.

Suite B cryptography does not define cryptographic algorithms. Instead, it specifies the cryptographic algorithms that can be used in a “Suite B Compliant” TLS V1.2 session. Suite B requires the key establishment and authentication algorithms that are used in TLS V1.2 sessions to be based on Elliptic Curve Cryptography, and the encryption algorithm to be AES-CBC or AES-GCM. Suite B is not supported with TLS V1.3. If TLS V1.3 is enabled in the application, it is ignored when Suite B is enabled.

The security levels that are defined in the Suite B profile are:

- 128-bit security level that corresponds to an elliptic curve size of 256 bits and 128-bit AES-CBC or 128-bit AES-GCM ciphers.
- 128-bit minimum security level that corresponds to an elliptic curve size of 256 bits and a 128-bit AES-GCM cipher or an elliptic curve size of 384 bits and a 256-bit AES-GCM cipher.
- 192-bit security level that corresponds to an elliptic curve size of 384 bits and 256-bit AES-CBC or 256-bit AES-GCM ciphers.
- 192-bit minimum security level that corresponds to an elliptic curve size of 384 bits and a 256-bit AES-GCM cipher.

The numbers in Table 11 on page 49 indicate the cipher preference order for the 128-bit, 128-bit minimum, 192-bit, 192-bit minimum, and All Suite B profiles. If a number is not present, then that cipher suite is not supported for that Suite B profile.

Table 11. Suite B supported cipher suites

Cipher Suite	128-bit security level	128-bit-minimum security level	192-bit security level	192-bit minimum security level	All
C02B	1	1			3
C023	2				4
C02C		2	1	1	1
C024			2		2

For more information about the cipher suites, see Appendix C, “Cipher suite definitions,” on page 795.

The Suite B standard specifies the elliptic curves that are allowed in a TLS connection. Table 12 on page 49 contain a list of the curves that are allowed for the 128-bit, 128-bit minimum, 192-bit, 192-bit minimum, and All Suite B profiles.

Table 12. Supported curves

Named Curve	128-bit security level	128-bit minimum security level	192-bit security level	192-bit minimum security level	All
secp256r1 – {1.2.840.10045.3.1.7}	X	X			X

Table 12. Supported curves (continued)

Named Curve	128-bit security level	128-bit minimum security level	192-bit security level	192-bit minimum security level	All
secp384r1 – {1.3.132.0.34}		X	X	X	X

Server and client certificates that are used to establish a Suite B-compliant connection must be signed with ECDSA.

- For certificates used at the 128-bit security level, the subject public key must use the secp256r1 curve and be signed with either the secp384r1 curve or the secp256r1 curve.
- For certificates used at the 128-bit minimum security level, the subject public key can use the secp256r1 curve and be signed with the secp256r1 curve, or the subject public key can use the secp384r1 curve and be signed with the secp384r1 curve.
- For certificates used at the 192-bit or 192-bit minimum security levels, the subject public key must use the secp384r1 curve and be signed with the secp384r1 curve.
- For certificates used at the All security level, the subject public key must use the secp256r1 curve and be signed with either the secp384r1 curve or the secp256r1 curve, or the subject public key must use the secp384r1 curve and be signed with the secp384r1 curve.

Suite B support also imposes limitations on the signature and hash algorithm pairs that can be employed when signing the SERVER-KEY-EXCHANGE and CERTIFICATE-VERIFY messages. The numbers in Table 13 on page 50 indicate the signature and hash algorithm preference order for the 128-bit, 128-bit minimum, 192-bit, 192-bit minimum, and All Suite B profiles. If a number is not present, then that signature and hash algorithm is not supported for that Suite B profile.

Table 13. Signature and hash algorithms

Signature and hash algorithm	128-bit security level	128-bit minimum security level	192-bit security level	192-bit minimum security level	All
ECDSA with SHA-256 (0403)	1	1			2
ECDSA with SHA-384 (0503)		2	1	1	1

Note: The signature and hash algorithm that is used to sign the SERVER-KEY-EXCHANGE and CERTIFICATE-VERIFY messages with the 128-bit minimum or All Suite B profiles depends upon the certificates that are in use. For example, if the server's certificate is a secp256r1 curve certificate and the Suite B profile is set to 128-bit minimum or All, then the ECDSA with SHA-256 signature and hash algorithm is used to sign the SERVER-KEY-EXCHANGE message.

When the 128-bit minimum and 192-bit minimum Suite B profiles are configured, certificate and CRL validation is done according to RFC 5820 - *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* and RFC 5759 – *Suite B Certificate and Certificate Revocation List (CRL) Profile* when doing the TLS handshake no matter the certificate validation mode setting. Therefore, the GSK_CERT_VALIDATION_MODE setting is ignored when the 128-bit minimum and 192-bit minimum Suite B profiles are enabled. RFC 5759 enhances the requirements of RFC 5280 certificate validation mode when checking the self-signed root CA certificates, intermediate CA certificates, and end entity certificates. In addition, certificate validation that is done for the 128-bit minimum and 192-bit minimum Suite B profiles ensure that the public keys certificates are in non-descending order of size from the end entity certificate to the trust anchor certificate. (Note: A trust anchor certificate can be used when the SAF key ring contains an intermediate CA certificate without the root CA certificate present on the key ring. The GSK_CERT_VALIDATE_KEYRING_ROOT environment variable or attribute identifier on the **gsk_attribute_set_enum()** routine controls how certificates in a keyring are validated.)

Note: OCSP response signatures are not automatically checked to verify that they are either ECDSA with SHA-256 or ECDSA with SHA-384 when using the Suite B 128-bit minimum or 192-bit minimum profiles. If this checking is wanted, specify the GSK_OCSP_RESPONSE_SIGALG_PAIRS environment variable or attribute type in the **gsk_attribute_set_buffer()** and verify that the list includes 0403 (ECDSA with SHA-256) and 0503 (ECDSA with SHA-384).

The following additional certificate validation is done for a peer's self-signed root CA certificate when using the Suite B 128-bit minimum or 192-bit minimum profiles. If the criteria is not met, a certificate validation error is returned.

- Must contain the subjectKeyIdentifier, keyUsage, and basicConstraints extensions.
- The keyUsage extension must be marked as critical. The certificate sign and CRL sign bits in the keyUsage extension must be set. The digital signature and nonRepudiation bits can be set. All other bits are not allowed to be set.
- The basicConstraints extension must be marked as critical. The cA boolean must be set to indicate that the certificate is a CA and the pathLenConstraint subfield must not be present.

The following additional certificate validation is done for a peer's intermediate CA certificates when using the Suite B 128-bit minimum or 192-bit minimum profiles. If the criteria is not met, a certificate validation error is returned.

- Must contain the authorityKeyIdentifier, keyUsage, and basicConstraints extensions.
- The keyUsage extension must be marked as critical. The certificate sign and CRL sign bits in the keyUsage extension must be set. The digital signature and nonRepudiation bits can be set. All other bits are not allowed to be set.
- The basicConstraints extension must be marked as critical. The cA boolean must be set to indicate that the certificate is a CA and the pathLenConstraint subfield is optional.
- If a policy is asserted, the certificatePolicies extension must be marked as non-critical and must contain the OIDs for the applicable certificate policies. It is recommended that the policyQualifiers option not be included, hence System SSL does not enforce this extension being present.
- The policyMappings, policyConstraints, and inhibitAnyPolicy extensions are allowed to be any criticality.

The following additional certificate validation is done for a peer's end entity certificate when using the Suite B 128-bit minimum or 192-bit minimum profiles. If the criteria is not met, a certificate validation error is returned.

- Must contain the authorityKeyIdentifier and keyUsage extensions. It is recommended that end entity certificates contain the subjectKeyIdentifier extension, hence System SSL does not enforce this extension being present.
- The keyUsage extension must be marked as critical. The digital signature bit must be set. The nonRepudiation bit can be set. All other bits must not be set.
- If a policy is asserted, the certificatePolicies extension must be marked as non-critical and must contain the OIDs for the applicable certificate policies. It is recommended that the policyQualifiers option not be included; however, System SSL does not enforce the option being present.

Notes:

- If end user general ECC or user or server self-signed certificates are created in gskkyman, they will not work when the Suite B profile is set to 128MIN or 192MIN because the keyUsage extension has the keyAgreement bit set, which is not allowed in RFC 5759.
- If the certificates are created with the RACDCERT command, ensure that the KEYUSAGE parameter is set appropriately when generating the root CA, intermediate CA, and end entity certificates based on the guidelines when the 128-bit minimum or 192-bit minimum Suite B profile is used. For more information about the RACDCERT command, see *z/OS Security Server RACF Command Language Reference*.

Whenever a Suite B-compliant client and a Suite B-compliant server establish a TLS V1.2 session, only Suite B algorithms are employed. For more information about the cipher suites, see Appendix C, "Cipher suite definitions," on page 795. In a Suite B-compliant session, the TLS V1.2 protocol must be used to establish an SSL connection. Therefore, when System SSL is configured to run in a Suite B-compliant mode, any non-TLS V1.2 protocols (including TLS V1.3) that are configured for the connection are ignored and the TLS V1.2 protocol is activated, if not already active.

Additionally, Suite B also places restrictions on which cipher suites, elliptical curves, and signature algorithms can be used in a Suite B-compliant session. When System SSL is running in a Suite B mode, any cipher suites, cipher format, elliptical curves, and signature algorithms that are configured are ignored. Only the cipher suites, cipher format, elliptical curves, and signature algorithms for the profile that is chosen for the Suite B session are used by System SSL to establish the connection.

SSL/TLS partner certificate revocation checking

When performing revocation information checking against a partner certificate being used for a secure connection, the SSL/TLS application can be configured to obtain revocation information from any combination of the following revocation sources:

- A dedicated OCSP responder or OCSP responders specified in the Authority Information Access (AIA) extension.
- An HTTP server specified in the CRL Distribution Points (CDP) extension.
- A dedicated LDAP server.

Enabling OCSP support

OCSP revocation information can be obtained through OCSP responders identified within a certificate AIA extension (URI value) or through a dedicated OCSP responder.

To enable the use of the certificate AIA extension, when present in the certificate being validated, the GSK_OCSP_ENABLE setting must be set to ON. Note that in order to use the AIA extension, the extension must contain at least one identifying OCSP responder entry. An OCSP responder entry contains an access method of OCSP and URI access location.

When processing the possible values in the AIA extension, an attempt to contact each OCSP responder is tried and processing stops with the first responder that is able to be successfully contacted. The OCSP response from that OCSP responder is used to determine the revocation status.

To enable the used of a dedicated OCSP responder, GSK_OCSP_URL must be set to the HTTP URL of the OCSP responder. Only one dedicated responder can be specified. For information about specifying a HTTP URL, see the description for GSK_OCSP_URL in "gsk_attribute_set_buffer()" on page 100.

If GSK_OCSP_ENABLE is set ON and the dedicated OCSP responder is specified (GSK_OCSP_URL), both the dedicated OCSP responder and the responders identified in the AIA extension may be used to obtain revocation status. By default, the dedicated OCSP responder is utilized first during validation of a certificate. The default order can be changed through the GSK_OCSP_URL_PRIORITY setting. To have the dedicated OCSP responder (GSK_OCSP_URL) used after the AIA extension, GSK_OCSP_URL_PRIORITY must be set to OFF.

In all cases, once an OCSP responder has returned a response, the response is used to determine the revocation state of the certificate being validated.

If the dedicated OCSP responder (GSK_OCSP_URL) requires the OCSP request to be digitally signed or your security policy requires signing, signing is enabled by specifying the certificate to be used for signing and optionally the signature algorithm. The signing certificate is identified by setting GSK_OCSP_REQUEST_SIGKEYLABEL to the label of the certificate. The optional signature algorithm is specified through GSK_OCSP_REQUEST_SIGALG and defaults to RSA with SHA-256 (0401).

All OCSP responses are digitally signed by the OCSP responder using a CA certificate. The response signature must be verified using the public key of that certificate. The OCSP signing certificate must be found in either the trusted certificate store, the untrusted certificates that are provided through the handshake messages, or the OCSP response. The certificate chain for the OCSP signing certificate must also be validated through the root certificate for the chain and the root certificate must be in the trusted certificate store. Note that revocation checking of the OCSP response signer certificate is not performed if the id-pkix-ocsp-nocheck extension is present in the signing certificate.

The following OCSP request and response characteristics can be tailored to your environment:

HTTP method
: The GSK_OCSP_RETRIEVE_VIA_GET setting is used to indicate whether the HTTP GET or the HTTP POST method is to be used when sending OCSP requests to the OCSP responder. By default, the HTTP POST method is used. The HTTP GET method allows for the enablement of HTTP caching on the OCSP responder and can only be used when the OCSP request is less than 255 bytes.

  Note: System SSL sends HTTP OCSP GET requests as described in RFC 2560, which includes URL encoding of the base-64 encoding of the DER encoding of the OCSPRequest. However, some HTTP servers are unable to properly handle the URL encoding and may return unexpected HTTP errors while parsing the OCSP GET request. If these problems persist and it is not possible to correct the HTTP server behavior, send the OCSP requests with the HTTP POST method.

HTTP proxy server
: To allow the OCSP request and OCSP response to be routed via a proxy HTTP server, GSK_OCSP_PROXY_SERVER_NAME and GSK_OCSP_PROXY_SERVER_PORT must be set.

Nonce
: The OCSP protocol allows for the OCSP request and corresponding OCSP response to have additional verifying information in the form of a nonce value. The nonce is intended to cryptographically bind the OCSP response to a specific OCSP request. A nonce is sent when GSK_OCSP_NONCE_GENERATION_ENABLE is set to ON and checked when GSK_OCSP_NONCE_CHECK_ENABLE is set to ON. The size of the nonce can be specified by GSK_OCSP_NONCE_SIZE setting. Enabling this option improves security, but may cause failures if the OCSP responder does not support nonces.

OCSP response caching
: OCSP responses received from an OCSP responder can be cached in the application's address space to save the processing time required to contact an OCSP responder. OCSP caching is enabled by default when OCSP revocation checking is enabled. The maximum number of OCSP responses that are allowed to be stored in the OCSP cache is indicated by the GSK_OCSP_CLIENT_CACHE_SIZE setting, which has a default of 256. To have the OCSP responder contacted for every validation, GSK_OCSP_CLIENT_CACHE_SIZE must be set to 0. With the OCSP cache, it is possible to limit the number of cached responses tied to a specific issuing CA certificate through the GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE setting. By default, this size is set to 0, which indicates there is no limit on the number of cached certificate statuses allowed for a specific issuing CA certificate other than the limit imposed by the overall GSK_OCSP_CLIENT_CACHE_SIZE setting. The value specified for the GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE setting cannot exceed the GSK_OCSP_CLIENT_CACHE_SIZE setting.

  OCSP responses are only cached when they contain a nextUpdate time. No nextUpdate time indicates that the OCSP response is only valid at the time the response was provided. The nextUpdate time is used as the expiration time of the OCSP response in the cache. If the HTTP response containing the OCSP response includes the max-age option in the Cache-Control directive, the expiration time of the OCSP response in the cache is either the nextUpdate time or max-age plus the current time, whichever is earlier.

  If the HTTP response contains the Expires directive and the max-age option in the Cache-Control directive is not specified, the expiration time of the OCSP response in the cache is either the nextUpdate time or the Expires time of the HTTP response, whichever is earlier.

  When a new OCSP response is being added to the OCSP cache and the cache reaches its size limit, all expired entries are removed from the cache and then the newly obtained OCSP response is added.

  If during the validation process, an attempt to use a cached OCSP response detects the entry is expired, the OCSP response is removed from the cache and the OCSP responder is contacted to obtain a new OCSP response.

Response size
Although OCSP responses should be small in size, a default maximum size of 20480 (20K) has been defined. The size can be overridden through the GSK_OCSP_MAX_RESPONSE_SIZE setting.

Response timeout
When communicating with the OCSP responder, by default, the amount of time to wait for the response is set to 15 seconds. The timeout can be overridden through the GSK_OCSP_RESPONSE_TIMEOUT setting.

Enabling HTTP CDP support

Certificate revocation can be obtained through HTTP servers identified within a certificate's CDP extension. The HTTP server provides the revocation information in the form of a CRL.

To enable the use of the certificate's CDP extension when the extension is present in the certificate being validated and for retrieving CRLs from an HTTP server, the GSK_HTTP_CDP_ENABLE setting must be set to ON. Note that in order to use the HTTP CDP extension, the extension must contain at least one identifying HTTP entry.

When processing the possible values in the CDP extension, an attempt to contact each HTTP server is tried and processing stops with the first server that is able to be successfully contacted. The HTTP response contains a CRL that is used to determine the revocation status of the certificate.

The following HTTP server and CRL characteristics can be tailored to your environment:

HTTP CDP CRL caching
CRLs retrieved via HTTP can be cached in the application's address space to save the processing time required to contact an HTTP server. HTTP CDP CRL caching is enabled by default when HTTP CDP CRL revocation checking is enabled. The maximum number of CRLs that are allowed to be stored in the HTTP CDP CRL cache is indicated by the GSK_HTTP_CDP_CACHE_SIZE setting, which has a default of 32. To have the HTTP server contacted for every validation, GSK_HTTP_CDP_CACHE_SIZE must be set to 0.

CRLs are only cached when they contain a nextUpdate time. No nextUpdate time indicates the CRL is only valid at the time it was provided. The nextUpdate time is used as the expiration time of the CRL in the cache. If the HTTP response includes the max-age option in the Cache-Control directive, the expiration time of the CRL from the cache is either the nextUpdate time or max-age plus the current time, whichever is earlier.

If the HTTP response contains the Expires directive and the max-age option in the Cache-Control directive is not specified, the expiration time of the OCSP response in the cache is either the nextUpdate time or the Expires time of the HTTP response, whichever is earlier.

When a new CRL is being added to the HTTP CDP CRL cache, expired CRLs are removed. If the cache reaches its size limit, an entry is removed from the cache and the newly obtained CRL is added. If during the validation process, an attempt to use a cached CRL detects the entry is expired, the CRL is removed from the cache and the HTTP server is contacted to obtain an updated CRL.

If storing large CRLs in the HTTP CDP CRL cache is a concern, the GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE can be used to limit the byte size of these CRLs. By default, the GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE setting is set to 0, which indicates there is no limit on the size of a CRL stored in the HTTP CDP CRL cache.

HTTP proxy server
To allow the HTTP request and response to be routed via a proxy HTTP server, GSK_HTTP_CDP_PROXY_SERVER_NAME and GSK_HTTP_CDP_PROXY_SERVER_PORT must be set.

Response size
The HTTP response consists of the HTTP response directives and the CRL. CRLs can vary in size from being very small to extremely large depending upon how many non-expired certificates have been revoke by the CA. The default maximum HTTP response size is 204800 (200K). The size can be overridden through the GSK_HTTP_CDP_MAX_RESPONE_SIZE setting.

Response timeout

When communicating with the HTTP server, by default, the amount of time to wait for the response is set to 15 seconds. The timeout can be overridden through the GSK_HTTP_CDP_RESPONSE_TIMEOUT setting.

Enabling LDAP CRL support

Certificate revocation can be obtained through an LDAP server. The LDAP server provides the revocation information in the form of a CRL.

To enable the use of an LDAP server, GSK_LDAP_SERVER must be set to the IP address or the hostname of the LDAP server and optionally, the GSK_LDAP_PORT must be set to the port that the LDAP server is listening to for requests. When no port is provided, port 389 is used.

When checking the revocation status of a certificate, the x.500 directory name within the CDP extension is used to identify the CRL to be retrieved from the LDAP server. If the CDP extension is not present or does not contain an x.500 directory name, the issuer name within the certificate is used to identify the CRL. The returned CRL is used to determine the certificate revocation status.

The following LDAP server and CRL characteristics can be tailored to your environment:

Response timeout

The time to wait in seconds for a response from the LDAP server can be overridden by specifying GSK_LDAP_RESPONSE_TIMEOUT.

The default is 15 seconds.

LDAP CRL caching

CRLs retrieved from LDAP can be cached in the application's address space to save the processing time required to contact the LDAP server. LDAP CRL caching is enabled by default when LDAP CRL revocation checking is enabled.

When LDAP CRL support is enabled, LDAP basic CRL caching is automatically enabled. LDAP basic CRL caching allows for:

- All valid retrieved CRLs to be placed into the cache.
- An unlimited number of CRLs can reside in the cache. GSK_CRL_CACHE_SIZE can be specified to limit the number of entries allowed in the cache.
- CRLs stay in the cache for a maximum of 24 hours. Every 24 hours after the first CRL is added to the cache, the entire cache is emptied. GSK_CRL_CACHE_TIMEOUT can be specified to change the number of hours that the CRLs remain in the cache.
- Temporary CRLs are placed into the cache when the LDAP server does not contain the CRL. GSK_CRL_CACHE_TEMP_CRL can be specified to disable temporary CRL caching.

LDAP extended CRL cache support provides the ability to enhance the CRL caching capabilities. To enable LDAP extended CRL caching support, GSK_CRL_CACHE_EXTENDED must be set to ON. LDAP extended caching support allows the following:

- Valid retrieved CRLs are eligible to be cached when they contain a nextUpdate or expiration time that is greater than the current time.
- CRLs are allowed to expire in the cache at different times because the nextUpdate or expiration time controls their expiration time. When the cache is updated with a new CRL, any expired CRLs are removed.
- The default for GSK_CRL_CACHE_SIZE is 32 CRLs in the cache at one time. If the size is about to be exceeded, the CRL with the time closest to expiration is removed from the cache.
- By default, temporary CRLs are not added to the cache when the LDAP server does not contain the CRL. GSK_CRL_CACHE_TEMP_CRL can be specified to enable temporary CRL caching. If enabled, the lifetime of these temporary CRL entries is controlled by the GSK_CRL_CACHE_TEMP_CRL_TIMEOUT setting, which defaults to 24 hours.

For both basic and LDAP extended CRL cache support, the GSK_CRL_CACHE_ENTRY_MAXSIZE indicates the maximum size in bytes of a CRL that is allowed to be stored in the cache. The default is 0, which means that the CRL size is unlimited.

If LDAP basic CRL cache support is enabled, LDAP CRL caching can be disabled by setting GSK_CRL_CACHE_SIZE to 0 or GSK_CRL_CACHE_TIMEOUT to 0. If LDAP extended CRL cache support is enabled, LDAP CRL caching can be disabled by setting GSK_CRL_CACHE_SIZE to 0.

Revocation source checking (order of precedence)

If multiple revocation sources are enabled (OCSP, HTTP CDP, and LDAP) in the System SSL environment, there is an order of precedence that is used when checking for certificate revocation information. Use the GSK_OCSP_URL_PRIORITY setting to specify the order that the dedicated OCSP responder and OCSP responders in the AIA extension are checked for certificate revocation information.

- To have the dedicated OCSP responder checked first, set GSK_OCSP_URL_PRIORITY to ON. This is the default.
- To have the AIA extension checked first, set GSK_OCSP_URL_PRIORITY to OFF.

The GSK_AIA_CDP_PRIORITY setting specifies the order that the AIA and CDP extensions are checked for certificate revocation information. To have the OCSP responders (dedicated OCSP responder or OCSP responders in the AIA extension) checked before the HTTP servers specified in the CDP extension, set GSK_AIA_CDP_PRIORITY to ON. To have the HTTP servers specified in the CDP extension checked before the OCSP responders, set GSK_AIA_CDP_PRIORITY to OFF. By default, this setting is set to ON.

Revocation security enforcement

The GSK_REVOCATION_SECURITY_LEVEL setting specifies the level of security to be used when contacting an OCSP responder or an HTTP server specified in the CDP extension. When set to LOW, it indicates that certificate validation does not fail if the OCSP responder or the HTTP server cannot be contacted. When set to MEDIUM, it indicates that certificate validation fails if all OCSP responders and HTTP servers are not contactable, or if they are contactable, a valid OCSP response or CRL must be returned. When set to HIGH, it indicates that certificate validation fails if revocation information cannot be contained from any of the specified sources.

If GSK_LDAP_SERVER is specified, it is always checked last for certificate revocation information if either OCSP or HTTP CDP is enabled. To enable fallback to LDAP (if the OCSP responders or HTTP servers cannot be contacted), the GSK_REVOCATION_SECURITY_LEVEL option must be set to LOW. This then allows contact to be made to the LDAP server specified in the GSK_LDAP_SERVER option. The GSK_CRL_SECURITY_LEVEL option specifies the level of security to be used when contacting an LDAP server and has three settings: LOW, MEDIUM, and HIGH. For more information about the LDAP security level, see the information about GSK_CRL_SECURITY_LEVEL in Appendix A, "Environment variables," on page 763.

Revocation examples

The following examples depict some typical environments that can be configured in System SSL and indicate how the revocation sources are checked for revocation information.

Example 1

Certificate validation uses revocation information from either a dedicated OCSP responder or OCSP responders specified in the certificate's AIA extension. The dedicated OCSP responder is attempted first and if not contactable, the OCSP responders in the AIA extension are used. If none of the specified OCSP responders can be contacted, certificate validation fails. The dedicated OCSP URL is 127.0.0.1

```
GSK_OCSP_URL=http://127.0.0.1
GSK_OCSP_ENABLE=ON
GSK_REVOCATION_SECURITY_LEVEL=MEDIUM
```

Example 2

Certificate validation uses revocation information provided through the AIA and CDP extensions within the certificate. The AIA contains any OCSP responders to be used and the CDP contains any HTTP servers to be used. The OCSP responders within the AIA extension are checked first. If the OCSP responders cannot be contacted, the HTTP servers within the CDP extension are used. If there are no OCSP responders and HTTP servers or none of them can be contacted, the certificate is considered not revoked.

```
GSK_OCSP_ENABLE=ON
GSK_HTTP_CDP_ENABLE=ON
GSK_AIA_CDP_PRIORITY=ON
GSK_REVOCATION_SECURITY_LEVEL=LOW
```

Example 3

Certificate validation uses revocation information provided through the dedicated OCSP responder and the AIA and CDP extensions within the certificate. The revocation providers are checked in the following order:

1. HTTP servers within the CDP extensions.
2. OCSP responders in the AIA extensions.
3. Dedicated OCSP responder.

If none of the OCSP responders or HTTP servers can be contacted, certificate validation fails.

```
GSK_OCSP_URL=http://127.0.0.1
GSK_OCSP_ENABLE=ON
GSK_OCSP_URL_PRIORITY=OFF
GSK_HTTP_CDP_ENABLE=ON
GSK_AIA_CDP_PRIORITY=OFF
GSK_REVOCATION_SECURITY_LEVEL=MEDIUM
```

Example 4

Certificate validation uses revocation information provided by the certificate's CDP extension and a LDAP server. The CDP extension is checked first and if the HTTP server cannot be contacted, the LDAP server is used. If the LDAP server is used, a CRL must be available. If no revocation information is retrieved, the certificate is considered revoked.

```
GSK_HTTP_CDP_ENABLE=ON
GSK_LDAP_SERVER=127.0.0.1
GSK_LDAP_USER=cn=admin
GSK_LDAP_PASSWORD=secret
GSK_CRL_SECURITY_LEVEL=HIGH
GSK_REVOCATION_SECURITY_LEVEL=LOW
```

Note: GSK_REVOCATION_SECURITY_LEVEL controls the processing characteristics of the CDP extension. LOW allows processing to continue to the LDAP server. GSK_CRL_SECURITY_LEVEL controls the processing characteristics of the LDAP server.

Enabling OCSP server stapling

System SSL allows server applications the ability to supply the OCSP response for the server's end entity certificate or the server's certificate chain if the client indicates support for this capability during the TLS handshake. This is commonly referred to as OCSP stapling as the OCSP responses for the server's end entity certificate or server's certificate chain are attached to a handshake message. The client then parses the OCSP responses to determine the revocation status of the server's certificates to determine if the handshake should continue.

The GSK_SERVER_OCSP_STAPLING attribute type or environment variable on the **gsk_attribute_set_enum()** routine enables the OCSP stapling support in the server. When enabled, OCSP stapling support allows the server to query the OCSP responder for the revocation status of the server's end entity certificate or the server's certificate chain. See the description of the

GSK_SERVER_OCSP_STAPLING attribute type in **gsk_attribute_set_enum()** or the environment variable in Appendix A, “Environment variables,” on page 763.

If the server application is enabled for OCSP stapling, OCSP support must be enabled with the GSK_OCSP_URL or the GSK_OCSP_ENABLE environment variables or attribute types so that certificate revocation information can be retrieved from the OCSP responder.

All existing OCSP-related configuration options play a role in the OCSP stapling support. For example, the GSK_OCSP_CLIENT_CACHE_SIZE attribute type or environment variable indicates the size of the OCSP cache for the OCSP responses for the server's end entity certificate or the server's certificate chain and also any client certificates that are received during the TLS handshake. The OCSP response that is retrieved for the server's end entity certificate or the server's certificate chain when OCSP stapling is enabled follows the same basic expiration rules that are described in “OCSP response caching” on page 53

System SSL accepts the following OCSP-related options on the server side when enabled for OCSP stapling:

- GSK_MAX_SOURCE_REV_EXT_LOC_VALUES
- GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES
- GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE
- GSK_OCSP_CLIENT_CACHE_SIZE
- GSK_OCSP_ENABLE
- GSK_OCSP_PROXY_SERVER_NAME
- GSK_OCSP_PROXY_SERVER_PORT
- GSK_OCSP_REQUEST_SIGALG
- GSK_OCSP_REQUEST_SIGKEYLABEL
- GSK_OCSP_RESPONSE_TIMEOUT
- GSK_OCSP_RETRIEVE_VIA_GET
- GSK_OCSP_URL
- GSK_OCSP_URL_PRIORITY

For example, the GSK_OCSP_CLIENT_CACHE_SIZE setting indicates the total number of cached entries that can be stored in the OCSP cache on the server side. In other words, the OCSP cache can contain the OCSP responses for the server's certificate chain and the OCSP responses for the client's certificates if enabled for client authentication.

The following OCSP-related options are ignored when OCSP stapling is enabled:

- GSK_OCSP_MAX_RESPONSE_SIZE
- GSK_OCSP_NONCE_CHECK_ENABLE
- GSK_OCSP_NONCE_GENERATION_ENABLE
- GSK_OCSP_NONCE_SIZE
- GSK_OCSP_RESPONSE_SIGALG_PAIRS
- GSK_REVOCATION_SECURITY_LEVEL

System SSL applications can query the GSK_TLSEXT_SERVER_OCSP_STAPLING attribute type on the **gsk_attribute_get_enum()** routine to determine if OCSP stapling support has been negotiated between the client and server. For more information, see “GSK_TLSEXT_SERVER_OCSP_STAPLING” on page 95.

If a TLS V1.2 and earlier handshake is negotiated, the OCSP response for the server's end entity certificate and the OCSP responses for the server's certificate chain are sent from the server to the client in the CERTIFICATE-STATUS message. If the server is unable to retrieve a valid OCSP response for the server's certificate from the OCSP responder, the OCSP response in the CERTIFICATE-STATUS message is empty. In this case, it is up to the client to determine whether to continue the handshake or not.

If a TLS V1.3 handshake is negotiated, the OCSP response for the server's end entity certificate is sent from the server to the client in the Certificate Status Request extension in the server's CERTIFICATE message. If the server is unable to retrieve a valid OCSP response for the server's certificate from the OCSP responder, the Certificate Status Request extension is not sent for the server's end entity certificate.

For OCSP server stapling, System SSL will not query an OCSP responder in the following cases:

- The server's certificate is a self-signed certificate. If the negotiated protocol is TLS V1.2 and earlier, a CERTIFICATE-STATUS message is not sent to the client. If the negotiated protocol is TLS V1.3, the Certificate Status Request extension is not sent for that certificate.
- The key repository does not contain the issuing certificate.
- If the negotiated protocol is TLS V1.2 and earlier and the server's key repository does not have the issuing certificate for the end entity certificate, the CERTIFICATE-STATUS message is not sent to the client. If the intermediate certificate authority's issuing certificate is missing, the OCSP responder is not contacted and the CERTIFICATE-STATUS message contains an empty OCSP response for that certificate.
- If the negotiated protocol is TLS V1.3 and the server's key repository does not have the issuing certificate for the end entity certificate, the Certificate Status Request extension is not sent for that certificate.

System SSL applications can query the GSK_SERVER_OCSP_STAPLING_CERTSTATUS attribute type on the **gsk_attribute_get_enum()** routine to determine whether a CERTIFICATE-STATUS message or a Certificate Status Request extension in the CERTIFICATE message containing the OCSP response is sent to the client. For more information, see "GSK_SERVER_OCSP_STAPLING_CERTSTATUS" on page 93.

Using server multiple key label support

System SSL allows multiple certificates to be specified for use by a server. The main purpose of multiple certificates is to enable Elliptic Curve Digital Signature Algorithm (ECDSA) certificates while still allowing RSA certificates to be used with clients that require RSA.

For maximum interoperability, a server must be able to negotiate with a wide range of clients with varying SSL capabilities which may require more than one type of certificate to be used. For example, when one client does not support TLS V1.2 or ECDSA certificates, the server must retain the ability to negotiate with an RSA certificate.

During each SSL handshake, the appropriate certificate is selected for use by the session based on the attributes for the session. The selection process uses the SSL attributes from both the client and server to make the decision. It is possible that no certificate is usable for a combination of attributes.

When creating a server environment in System SSL, you have the option of specifying a single label of a key to be used for each connection utilizing attribute GSK_KEYRING_LABEL,

```
rc = gsk_attribute_set_buffer(env_handle, GSK_KEYRING_LABEL,
     "ServerCertLabel",0);
```

or specifying multiple key labels utilizing attribute GSK_SERVER_KEYRING_LABEL_LIST,

```
rc = gsk_attribute_set_buffer(env_handle,  GSK_SERVER_KEYRING_LABEL_LIST,
     "ServerCertLabel1,ServerCertLabel2,ServerCertLabel3",0);
```

Utilizing multiple key label support allows for a single server to support a range of certificates and key exchange types and thereby allowing secure connections with a larger range of clients. For example, specifying:

```
rc = gsk_attribute_set_buffer(env_handle, GSK_SERVER_KEYRING_LABEL_LIST,
     "rsa1024Cert,ecc256Cert",0);
```

could allow connections to two different clients using completely different keys and key exchange types as long as the GSK_V3_CIPHER_SPECS or GSK_V3_CIPHER_SPECS_EXPANDED attribute contains an appropriate cipher for the protocol.

Utilizing multiple key label support also allows an administrator to specify certificates that will soon expire along with certificates that are not valid yet in order to avoid interruption of service by having to restart a server by specifying a new GSK_KEYRING_LABEL.

```
rc = gsk_attribute_set_buffer(env_handle, GSK_SERVER_KEYRING_LABEL_LIST,
      "rsaCertExpiresApril_1_2016,rsaCertValidApril_1_2016",0);
```

Selection of an appropriate certificate is based upon the following criteria:

- The highest protocol the client and server both support. The supported protocols are SSL Version 3.0, TLS Version 1.0, TLS Version 1.1, TLS Version 1.2, and TLS Version 1.3. Note that SSL Version 2.0 is not supported within multiple key label support.
- When the negotiated protocol is TLS V1.2 or earlier:
  - The server's supported cipher list based upon that protocol.
  - The client's supported cipher list.
- When the negotiated protocol is TLS V1.3:
  - Basic certificate validity checking.

This selection criteria identifies the first certificate in the list that matches the above parameters that both the server and client support.

When utilizing server multiple key label support, it is important to consider the following:

- Protocol SSL V2 is not supported with multiple key label support. If the selected protocol between a client the server is SSL Version 2.0, attribute GSK_KEYRING_LABEL must be set in order to provide the server with the key label to be used for the connection.
- Certificates should be placed in the list in order of preference.
- Each key label specified for GSK_SERVER_KEYRING_LABEL_LIST should be contained within the keystore pointed to by GSK_KEYRING_FILE. Otherwise, that key label is ignored for possible usage with a connection.
- Using multiple certificates of the same key type is not advantageous unless a certificate early in the preference order is expected to expire soon and a certificate later in the order is expected to be the replacement certificate.
- A certificate selected from the list is the first certificate that fulfills the previously stated criteria. Handshake processing may fail later due to an additional requirement placed upon the certificate during connection creation.

GSK_SERVER_KEYRING_LABEL_LIST can be set at either the environment level or for an individual connection. When set at the environment level, the provided list is used for all subsequent connections unless a new list is defined for an individual connection.

```
/* create the SSL environment */
rc = gsk_environment_open(&env_handle);

/* set environment attributes */
rc = gsk_attribute_set_buffer(env_handle, GSK_SERVER_KEYRING_LABEL_LIST,
 "ServerCertLabel1,ServerCertLabel2,ServerCertLabel3",0);

/* initialize environment */
rc = gsk_environment_init(env_handle);

rc = gsk_secure_socket_open(env_handle, &soc1_handle);
```

gsk_secure_socket_init() will attempt to select a certificate from ServerCertLabel1 through ServerCertLabel3.

```
rc = gsk_secure_socket_open(env_handle, &soc2_handle);

rc = gsk_attribute_set_buffer(soc2_handle, GSK_SERVER_KEYRING_LABEL_LIST,
 "ServerCertLabel4,ServerCertLabel5,ServerCertLabel6,ServerCetLabel7",0);
```

gsk_secure_socket_init() will attempt to select a certificate from ServerCertLabel4 through ServerCertLabel7.

Once a connection is made, use the **gsk_attribute_get_buffer()** routine to find out which certificate was selected to complete the connection.

```
rc = gsk_attribute_get_buffer(soc1_handle, GSK_KEYRING_LABEL, userBuffer,
     userBufferLength);

rc = gsk_attribute_get_buffer(soc2_handle, GSK_KEYRING_LABEL, userBuffer,
     userBufferLength);
```

A maximum of eight key labels are allowed for the list.

If no appropriate key label is selected, the connection fails even if GSK_KEYRING_LABEL is defined or the key database has a default key.

TLS V1.3 protocol support

The TLS Version 1.3 protocol is a major rewrite of the earlier TLS protocol standards. The intent of this protocol rewrite allows for more secure communications between peers and reduce the roundtrips necessary to complete a TLS V1.3 handshake. With these considerations in mind, there are certain things that you must be aware of when enabling a System SSL application for TLS Version 1.3.

Cipher specifications

The TLS V1.3 standard introduced ciphers which are no longer tied to the server's certificate and key exchange that is occurring. Therefore, the cipher specifications that are valid in TLS V1.2 and earlier protocols cannot be used in TLS V1.3. System SSL supports three TLS V1.3 cipher suites: 1301, 1302, and 1303. Applications that are enabled for TLS V1.3 must use 4-character cipher specifications and specify these cipher specifications in the GSK_V3_CIPHER_SPECS_EXPANDED setting. See Table 30 on page 804 for the list of the supported 4-character ciphers for TLS V1.3.

Key shares

One of the security improvements made in TLS V1.3 is that most of the handshake is now encrypted. This is done by having the client and server sides each specify a key share group list. When the client attempts a TLS V1.3 handshake, it generates a public/private key pair for each supported group and the generated public value or values are put into the client's initial handshake message. The server selects a group in common with the client's groups and then generates its own public/private key pair for the selected group. The server takes its private value and the client's public value to generate a shared secret, which is used to generate keys for encrypting and decrypting handshake messages. Likewise, the client generates the same shared secret with the server's public value that is transmitted in the server's initial handshake message.

It is important to properly configure the key share settings (GSK_SERVER_TLS_KEY_SHARES for the server side and GSK_CLIENT_TLS_KEY_SHARES for the client side) and the client ecurve list (GSK_CLIENT_ECURVE_LIST) for the appropriate encryption level for your application.

The GSK_CLIENT_TLS_KEY_SHARES setting indicates the list of groups or curves that are supported by the client when doing a TLS V1.3 handshake. For the GSK_CLIENT_TLS_KEY_SHARES setting, a minimum number of groups should be specified as public/private key pairs are generated for each one, which is computationally expensive and may impact performance. If running as a client, the GSK_CLIENT_TLS_KEY_SHARES setting must be specified when TLS V1.3 is enabled or TLS V1.3 handshakes will fail.

The client examines the GSK_CLIENT_TLS_KEY_SHARES and the GSK_CLIENT_ECURVE_LIST settings to find groups or curves that are in common between the two lists and sends them to the server in the order of the GSK_CLIENT_ECURVE_LIST setting. It is important that the group or curve order in the GSK_CLIENT_ECURVE_LIST is set appropriately because that is the order used to send the key share groups to the server. See Table 29 on page 804 for the supported key share groups.

The GSK_SERVER_TLS_KEY_SHARES setting indicates the list of groups or curves that are supported by the server when doing a TLS V1.3 handshake. For the GSK_SERVER_TLS_KEY_SHARES setting, list all desired supported groups to allow as many TLS V1.3 clients to successfully establish a secure connection. If running as a server, the GSK_SERVER_TLS_KEY_SHARES setting must be specified when TLS V1.3 is enabled or TLS V1.3 handshakes will fail.

The server receives the key shares from the client during the TLS V1.3 handshake. While parsing the key share extension groups, the server uses the client's preference order in selecting a group that is also specified in the GSK_SERVER_TLS_KEY_SHARES setting. The selected group is used to encrypt the remaining portions of the TLS V1.3 handshake.

If the server receives an empty key share list from the client, or if the client list does not contain any key shares supported by the server, the server will attempt to suggest a key share value to be used.

The System SSL client will only negotiate using the key share groups specified in the GSK_CLIENT_TLS_KEY_SHARES list. If a server does not accept the key shares specified, the handshake will fail.

The negotiated and selected key share group for a TLS V1.3 handshake can be obtained by calling the **gsk_attribute_get_buffer()** routine and passing in the GSK_CONNECT_KEY_SHARE attribute identifier.

Supported groups or curves

On the client side, the GSK_CLIENT_ECURVE_LIST indicates the list of supported groups or curves that are sent to the server. This list is used by the client to guide the server as to which elliptic curves are preferred and guide group selection for encryption and decryption of handshake messages.

When doing a TLS V1.3 handshake, this setting must not be set to an empty string. If the application is only enabled for TLS V1.3, groups and curves that are not allowed in TLS V1.3 (for example, 0019 and 0024) are not sent to the partner and are quietly ignored. If the application is not enabled for TLS V1.3, supported groups valid for only TLS V1.3 (such as 0029 and 0030) are ignored. See Table 29 on page 804 for the supported groups or curve specifications.

Supported certificate types

TLS V1.3 has limited the allowed certificate types and no longer supports DSA or DH certificates. As documented in *RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3* RFC 8446 (tools.ietf.org/html/rfc8446), System SSL only supports RSA key sizes 2048 bits and larger and ECC keys 256 bits and larger when attempting to do TLS V1. 3 handshakes. RSA keys with a PKCS #1 v1.5 signature or with an RSASSA-PSS signature are supported.

If the certificate to be used for the TLS connection is of type RSA with its private key stored in the PKDS and was created or added to the RACF database prior to z/OS V2R4, the certificate will not be usable for TLS V1.3 connections. The RSA key needs to be protected using the ECC master key. Ensure that the ECC master key is activated in the CCA coprocessor and either recreate or re-import (PKCS#12) the certificate into the RACF database or convert the existing private key to be protected under the ECC master key. See Translate and replace an RSA key for RSA PSS (community.ibm.com/community/user/ibmz-and-linuxone/blogs/bob-petti1/2021/03/10/translate-and-replace-an-rsa-key-for-rsa-pss?CommunityKey=6593e27b-caf6-4f6c-a8a8-10b62a02509c&tab=recentcommunityblogsdashboard) for information on how to convert the private key. To ensure that the key can continue to be used for RSA PKCS 1.5 signature generation (for example, used for TLS V1.2 and prior connections), the format restriction keyword FR-NONE should be used when converting the private key.

Signature and hash algorithms

The GSK_TLS_SIG_ALG_PAIRS setting indicates the list of signature and hash algorithms that are supported by the client or server consisting of one or more 4-character values in order of preference for use in digital signatures of X.509 certificates and TLS handshake messages. The client sends its supported list in the signature algorithms extension to the server which is used to guide selection of the server's certificate and internal hashing of TLS V1.3 handshake messages. If the server is enabled for client authentication, the server sends its supported list to the client which is used to guide selection of the client's certificate and internal hashing of TLS V1.3 handshake messages.

If the optional GSK_CERT_TLS_SIG_ALG_PAIRS setting is specified, it indicates the list of hash and signature algorithm pair specifications that are supported by the client or server as a string consisting of one or more 4-character values in order of preference for use in digital signatures of X.509 certificates. The client sends is supported list in the certificate signature algorithms extension to the server which is used to guide selection of the server's certificate. If the server is enabled for client authentication, the server sends its supported list to the client which is used to guide selection of the client's certificate. If specified, the GSK_TLS_CERT_SIG_ALG_PAIRS setting overrides the GSK_TLS_SIG_ALG_PAIRS setting when checking the digital signatures of the remote peer's X.509 certificates. Generally, it is not necessary to specify the GSK_CERT_TLS_SIG_ALG_PAIRS setting as GSK_TLS_SIG_ALG_PAIRS setting is used for the supported signature and hash algorithms for certificates and TLS handshake messages.

The hash and signature algorithm that is selected for signing the TLS V1.3 handshake messages are determined in the following manner:

- A list of the signature algorithms that are in common between the client and server lists is constructed. This common signature algorithm list is in the order of the remote partner's list.
- Based upon the local certificate that is used during the handshake, an appropriate signature is selected from the common list.
  - An ECC certificate uses an ECDSA signature algorithm.
  - An RSA certificate signed with an RSA or an RSASSA-PSS signature algorithm uses an RSASSA-PSS signature algorithm.
- If an appropriate signature algorithm is not found in the common list on the server side, the server will select one from its list. See Table 14 on page 63 for the preferred signature algorithms that is used.
- If the client sends a certificate to the server and an appropriate signature algorithm is not found in the common list, the TLS V1.3 handshake fails.

Table 14. TLS V1.3 preferred signature algorithm to use for signing handshake messages based upon certificate types

Certificate type	Preferred Signature Algorithm
RSA certificate with a signature algorithm of RSA with SHA-1 (Key sizes 2048 and larger)	0804 - SHA-256 with RSASSA-PSS
RSA certificate with a signature algorithm of RSA with SHA-256 (Key sizes 2048 and larger)	0804 - SHA-256 with RSASSA-PSS
RSA certificate with a signature algorithm of RSASSA-PSS with SHA-256 (Key sizes 2048 and larger)	0804 - SHA-256 with RSASSA-PSS
RSA certificate with a signature algorithm of RSA with SHA-384 (Key sizes 2048 and larger)	0805 - SHA-384 with RSASSA-PSS
RSA certificate with a signature algorithm of RSASSA-PSS with SHA-384 (Key sizes 2048 and larger)	0805 - SHA-384 with RSASSA-PSS
RSA certificate with a signature algorithm of RSA with SHA-512 (Key sizes 2048 and larger)	0806 - SHA-512 with RSASSA-PSS
RSA certificate with a signature algorithm of RSASSA-PSS with SHA-512 (Key sizes 2048 and larger)	0806 - SHA-512 with RSASSA-PSS
Any ECC secp256r1 certificate	0403 - SHA-256 with ECDSA
Any ECC secp384r1 certificate	0503 - SHA-384 with ECDSA

Table 14. TLS V1.3 preferred signature algorithm to use for signing handshake messages based upon certificate types (continued)

Certificate type	Preferred Signature Algorithm
Any ECC secp521r1 certificate	0603 - SHA-512 with ECDSA

Note: Any other certificate types that are not listed in Table 14 on page 63 are not supported with TLS V1.3. ECC certificates that have a key size smaller than 256 bits and RSA certificates that have a key size smaller than 2048 are not supported with TLS V1.3.

On the server side, certificates that use SHA-1 signature algorithms for RSA and ECDSA will be ignored unless the client has included SHA-1 with RSA and SHA-1 with ECDSA in its signature algorithm list or certificate signature algorithm list. The server uses these algorithms as a last resort.

If the application is only enabled for TLS V1.3, hash and signature algorithms that are not allowed in TLS V1.3 (for example, 0202) are not sent to the partner and are quietly ignored.

Middlebox compatibility mode

While TLS V1.3 was being developed, there were issues with some middleboxes (proxies) that were not able to properly parse pure TLS V1.3 handshake messages and were expecting other messages to be sent like they were in previous TLS protocols. To allow these middleboxes to continue work without fully updating to the TLS V1.3 handshake process and message format, the TLS V1.3 specification outlined a compatibility mode that results in the TLS V1.3 handshake process including messages that are in the same format as the earlier TLS protocols. System SSL provides middlebox compatibility mode with the GSK_MIDDLEBOX_COMPAT_MODE setting. By default, middlebox compatibility mode is not enabled.

When enabled on the client side, the initial TLS V1.3 handshake message sent to the server includes a random Session ID even if one has not been previously cached by the client. If the server receives an initial handshake message from the client containing a Session ID, it will echo the Session ID in its response message no matter the server's compatibility mode setting. The server will send an additional message if the client's initial handshake message includes a Session ID or compatibility mode is enabled on the server side.

Refreshing session keys

When the handshake processing has completed and communication has been established with the peer, it is possible to update the application data read and write keys without having to create a new connection. Using the **gsk_secure_socket_misc()** API, both the application data read and write keys can change by specifying GSK_MISC_ID set to GSK_RESET_CIPHER. To only have the write key changed, specify GSK_MISC_ID set to GSK_RESET_WRITE_CIPHER.

Session resumption and session ticket

For information about resumption and session tickets, see "Session ID (SID) and session ticket cache" on page 42.

Upgrading to TLS V1.2 from earlier SSL and TLS protocols

The TLS V1.2 protocol introduced several security related updates of the earlier TLS and SSL protocol standards. When updating System SSL applications from earlier SSL/TLS protocols to TLS V1.2, there are several required updates and other optional considerations that should be examined when updating System SSL applications.

Required updates for TLS V1.2

There are several cipher specifications that are supported with earlier TLS and SSL versions that are no longer supported under TLS V1.2. If any TLS/SSL connections currently negotiate any of the following cipher suites, ensure that the local System SSL application and the remote peer application configure an acceptable cipher suite that is supported by TLS V1.2. The following ciphers are no longer supported in TLS V1.2:

- 0003 (03) - TLS_RSA_EXPORT_WITH_RC4_40_MD5
- 0006 (06) - TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
- 0009 (09) - TLS_RSA_WITH_DES_CBC_SHA
- 000C (0C) - TLS_DH_DSS_WITH_DES_CBC_SHA
- 000F (0F) - TLS_DH_RSA_WITH_DES_CBC_SHA
- 0012 (12) - TLS_DHE_DSS_WITH_DES_CBC_SHA
- 0015 (15) - TLS_DHE_RSA_WITH_DES_CBC_SHA

Notes:

- The System SSL default cipher list does not use any of the unsupported cipher suites. Because all cipher suites in the default cipher list are supported by TLS V1.2, no cipher suite changes are required for System SSL applications that use the default list.
- The GSK_V3_CIPHER_SPECS setting is used if the System SSL application is using 2-character cipher specifications. If using 4-character cipher specifications, the GSK_V3_CIPHER_SPECS_EXPANDED setting defines the allowed cipher specifications in the System SSL application.
- See Table 27 on page 799 for the allowed TLS V1.2 ciphers.

The TLS V1.2 protocol does not allow MD2 digital signatures within certificates. If any certificates in the local certificate chain or the remote communication partners' certificate chain use MD2 in their digital signatures, those certificates are rejected during a TLS V1.2 handshake. In these cases, it is necessary to obtain and install new certificates with stronger digital signatures to allow a successful TLS V1.2 handshake.

Cipher suite considerations when upgrading to TLS V1.2

The TLS V1.2 protocol has introduced several stronger cipher suites versus those that are supported in earlier TLS and SSL protocols. These new cipher specifications include those that support ephemeral Elliptic Curve key exchange, AES-GCM mode encryption, and SHA-256 and SHA-384 based message integrity algorithms:

- C02B - TLS_ECDHE_ECDSA_WITH_AES_128_ GCM_SHA256
- C02C - TLS_ECDHE_ECDSA_WITH_AES_256_ GCM_SHA384
- C02F - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- C030 - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

When upgrading a System SSL application to support TLS V1.2, the configured cipher specification list should be examined closely to ensure that appropriate strong cipher suites are configured. Every organization has different requirements for securing connections depending upon the data that is being protected. These organizational requirements should play a very important role when configuring the cipher specification list within System SSL applications. The following should be considered when configuring cipher specifications assuming the appropriate support by the remote peer:

- AES based ciphers are more secure than the corresponding 3DES, DES, and RC4 based ciphers. AES-GCM ciphers are more secure than AES-CBC ciphers.
- Cipher specifications that use NULL encryption should only be used in cases where it is intentionally desired to have message integrity protection without encrypting the traffic. These cases tend to be very rare and should be reviewed very carefully.
- Cipher specifications that use SHA-256 or stronger message integrity are preferred over those that use SHA (SHA-1) and MD5.
- Cipher specifications that use ephemeral Diffie-Hellman key exchange (DHE and ECDHE) are more secure than their fixed counterparts (DH and ECDH). However, the ephemeral Diffie-Hellman key exchange ciphers require significantly more CPU cycles. If your application generates large numbers

of TLS handshakes, be aware that the use of DHE and ECDHE cipher suites can cause unusually high CPU consumption.

- Cipher suite 0000 (TLS_NULL_WITH_NULL_NULL) should not be used as it does not provide message integrity or encrypt traffic/payload data.

If using the default System SSL cipher specification list, ensure that those default ciphers are appropriate for your application. If the application has specified its own cipher specification list, it should be specified in order from strongest to weakest. If FIPS mode is enabled in the application, the allowed cipher suites are further limited. For more information about supported cipher specifications, see Table 26 on page 795 and Table 27 on page 799.

Digital certificate considerations when upgrading to TLS V1.2

When upgrading a System SSL application to TLS V1.2, it is a good time to examine the local certificate and remote peer certificate chains in use. Every organization has different requirements for the certificates that are to be used. These organizational requirements should play a very important role when selecting certificates. The following should be considered when selecting the allowed certificates assuming the appropriate support by the remote peer:

- RSA key sizes should be at least 2048 bits.
- ECC key sizes should be at least 256 bits.
- Certificates should be signed with a digital signature based on the SHA-2 (SHA-224, SHA-256, SHA-384, or SHA-512) hashing algorithm.

System SSL provides several settings that indicate a minimum key size for the remote peer's end entity certificate. These System SSL settings include GSK_PEER_DH_MIN_KEY_SIZE, GSK_PEER_DSA_MIN_KEY_SIZE, GSK_PEER_ECC_MIN_KEY_SIZE, and GSK_PEER_RSA_MIN_KEY_SIZE. See Key size and Diffie-Hellman group size support for more information.

The TLS V1.2 protocol allows the specification of digital signature algorithm pairs that are acceptable within the remote peer digital certificate chain. Each signature algorithm pair includes an asymmetric algorithm (RSA, ECDSA, and so on) and a hashing algorithm (SHA, SHA-256, SHA-384, and so on). In prior protocols, there was no way to limit the allowable algorithms so certificates using any signature algorithm pair, even very weak ones, were accepted. The GSK_TLS_SIG_ALG_PAIRS setting and the GSK_TLS_CERT_SIG_ALG_PAIRS setting, which is new in z/OS V2R4, are used to indicate the allowed signature algorithms. If the GSK_TLS_CERT_SIG_ALG_PAIRS setting is specified, it takes precedence over the GSK_TLS_SIG_ALG_PAIRS setting. By default, the GSK_TLS_SIG_ALG_PAIRS setting allows any signature algorithm pair other than RSA with MD5. This should allow for a smooth TLS V1.2 upgrade unless the remote peer's certificate chain includes a signature algorithm of RSA with MD5.

If the GSK_TLS_SIG_ALG_PAIRS or GSK_TLS_CERT_SIG_ALG_PAIRS settings are updated to exclude any default signature algorithms, such as RSA with SHA-1 (0201), it is important that any such changes are coordinated with the owners of any potential peer certificates. If the remote partner uses a digital certificate that uses any excluded algorithms, the certificate is rejected during the attempted TLS V1.2 handshake. See GSK_TLS_CERT_SIG_ALG_PAIRS and GSK_TLS_SIG_ALG_PAIRS for more information.

Elliptic Curve Cryptography considerations when upgrading to TLS V1.2

The use of Elliptic Curve Cryptography (ECC) was introduced to the TLS V1.0 and TLS V1.1 protocols with RFC 4492 and is in the base of the TLS V1.2 protocol. When using ECC-based key exchange, the TLS client may indicate the specific set of elliptic curves that it is willing to use with its communication partner. System SSL refers to this list as the client ecurve list or client eCurves. In System SSL client applications, the GSK_CLIENT_ECURVE_LIST specifies this list. See GSK_CLIENT_ECURVE_LIST for more information. System SSL supports the client eCurve specifications on all TLS protocols so there are no upgrade actions required when enabling TLS V1.2. Any client ecurve list will continue to work with TLS V1.2 as it did with previous TLS protocol versions.

Upgrading from TLS V1.2 to TLS V1.2 and TLS V1.3 protocols

While it is a worthwhile goal to migrate an application from supporting TLS V1.2 secure connections to TLS V1.3 secure connections, it may take some time for peer applications to fully migrate from TLS V1.2 to TLS V1.3. To allow these peers to continue successfully establishing TLS V1.2 secure connections with your application, it is a good idea to support both TLS V1.2 and TLS V1.3 protocols for a defined period of time until all possible peers fully support TLS V1.3. The following discusses considerations to think about when supporting both TLS V1.2 and TLS V1.3 in your System SSL application. See "TLS V1.3 protocol support" on page 61 for additional TLS V1.3 information.

Required updates to enable TLS V1.3 protocol support

There are a few required updates that must be done when updating a z/OS System SSL application to support TLS V1.3. Application updates can be done via environment variables (if supported) or calling System SSL **gsk_attribute_set_*()** routines. If application code is to be updated, there is sample application code in "Writing a System SSL source program" on page 29.

Enabling TLS V1.3 protocol support

Enable TLS V1.3 protocol support by setting the GSK_PROTOCOL_TLSV1_3 setting to ON.

Cipher specifications

Update the application to support 4-character ciphers because the TLS V1.3 ciphers are all 4-character ciphers. System SSL applications must programmatically enable support for 4-character ciphers by running **gsk_attribute_set_enum()** with GSK_V3_CIPHERS and GSK_V3_CIPHERS_CHAR4 values. If the application is using 2-character cipher specifications and has set the GSK_V3_CIPHER_SPECS setting, it is necessary to convert those 2-character cipher specifications to 4-character ciphers and use the GSK_V3_CIPHER_SPECS_EXPANDED setting. In addition, one or more of the supported TLS V1.3 cipher specifications must be specified when enabled for the TLS V1.3 protocol.

In this example configuration, the application is configured to use ciphers 3D, 39, 38, 37, and 35.

```
GSK_V3_CIPHER_SPECS=3D39383735
```

If the application has not set the GSK_V3_CIPHER_SPECS or GSK_V3_CIPHER_SPECS_EXPANDED settings, it is using the System SSL default ciphers. The TLS V1.3 cipher specifications have not been added to the System SSL default cipher list. In these cases, specify one or more TLS V1.3 ciphers along with the default cipher list values in the GSK_V3_CIPHER_SPECS_EXPANDED setting. See "SSL/TLS protocol" on page 23 for the current System SSL default cipher list.

For example, if the current System SSL default cipher specification list for the GSK_V3_CIPHER_SPECS_EXPANDED setting is:

```
GSK_V3_CIPHER_SPECS_EXPANDED=003500380039002F00320033
```

The application must be updated to specify the GSK_V3_CIPHER_SPECS_EXPANDED setting with one or more TLS V1.3 ciphers in the list. This example shows the addition of ciphers 1302 and 1303 to the beginning of the System SSL cipher specification list above:

```
GSK_V3_CIPHER_SPECS_EXPANDED=13021303003500380039002F00320033
```

See Table 26 on page 795 and Table 27 on page 799 for the supported cipher specifications.

Key shares

System SSL has two settings GSK_CLIENT_TLS_KEY_SHARES and GSK_SERVER_TLS_KEY_SHARES that specify the key share groups supported within the client and server respectively. The client or server side settings are required when TLS V1.3 handshakes are attempted in that application. If the key share settings are not specified, TLS V1.3 handshakes will fail.

The following is an example configuring secp256r1 (0023) and x25519 (0029) in a client application:

```
GSK_CLIENT_TLS_KEY_SHARES=00230029
```

The following is an example configuring secp256r1 (0023), secp521r1 (0025), and x25519 (0029) in a server application:

```
GSK_SERVER_TLS_KEY_SHARES=002300250029
```

See Table 29 on page 804 for the supported key share groups.

Supported certificate types

When updating an application to support TLS V1.3, this is a good time to examine the local and remote peer certificates that are in use. As documented in *RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3* RFC 8446 (tools.ietf.org/html/rfc8446), System SSL only supports RSA key sizes 2048 bits and larger and ECC keys 256 bits and larger when attempting to do a TLS V1.3 handshake. RSA keys with a PKCS #1 v1.5 signature or with an RSASSA-PSS signature are supported. TLS V1.3 no longer supports DSA or DH certificates.

If the certificates currently being used in your environment do not adhere to these TLS V1.3 requirements, new RSA or ECC certificates must be obtained from the certificate authority (CA) and installed in the certificate repository to allow for successful TLS V1.3 secure connections. These new certificates should be obtained from the same CA to avoid possibly having the peer needing to install CA certificates from another certificate authority.

If the certificate to be used for the TLS connection is of type RSA with its private key stored in the PKDS and was created or added to the RACF database prior to z/OS V2R4, the certificate will not be usable for TLS V1.3 connections. The RSA key needs to be protected using the ECC master key. Ensure that the ECC master key is activated in the CCA coprocessor and either recreate or re-import (PKCS#12) the certificate into the RACF database or convert the existing private key to be protected under the ECC master key. See Translate and replace an RSA key for RSA PSS (community.ibm.com/community/user/ibmz-and-linuxone/blogs/bob-petti1/2021/03/10/translate-and-replace-an-rsa-key-for-rsa-pss?CommunityKey=6593e27b-caf6-4f6c-a8a8-10b62a02509c&tab=recentcommunityblogsdashboard) for information on how to convert the private key. To ensure that the key can continue to be used for RSA PKCS 1.5 signature generation (for example, used for TLS V1.2 and prior connections), the format restriction keyword FR-NONE should be used when converting the private key.

Optional considerations for certain configurations

Based upon the current configuration of the application, the following may need to be considered when supporting both TLS V1.2 and TLS V1.3 protocols in your System SSL application.

Supported groups or curves

If the client application has set the GSK_CLIENT_ECURVE_LIST setting, it must be examined to ensure that any groups within the GSK_CLIENT_TLS_KEY_SHARES are also specified. If the client application has not set the GSK_CLIENT_ECURVE_LIST setting, it does not need to be set unless the GSK_CLIENT_TLS_KEY_SHARES setting includes x448 (0030) as the x448 group is not in the default GSK_CLIENT_ECURVE_LIST.

The groups specified in the GSK_CLIENT_TLS_KEY_SHARES setting are only sent to the server when present in the GSK_CLIENT_ECURVE_LIST setting. See "Key shares" on page 61 for more information.

For example, the existing client application has this specified for the GSK_CLIENT_ECURVE_LIST setting:

```
GSK_CLIENT_ECURVE_LIST=00240025
```

If the client application has set the GSK_CLIENT_TLS_KEY_SHARES setting to 00230029, the GSK_CLIENT_ECURVE_LIST must be updated to specify 0023 and 0029 along with 0024 and 0025.

```
GSK_CLIENT_ECURVE_LIST=0023002400250029
```

See Table 29 on page 804 for a list of valid 4-character elliptic curve and supported groups specifications.

Signature and hash algorithms

If the application has set a specific GSK_TLS_SIG_ALG_PAIRS setting, it must be examined to ensure that it works with a TLS V1.3 handshake. If an RSA end entity certificate is in use, at least one RSASSA-PSS signature and hash algorithm that is in common with the peer must be added to the list. The TLS V1.3 protocol requires TLS handshake messages to be signed with an RSASSA-PSS signature when an RSA certificate is being used.

For example, the existing application has this specified for the GSK_TLS_SIG_ALG_PAIRS setting:

```
GSK_TLS_SIG_ALG_PAIRS=0601050104010301
```

The application should be updated to specify the three RSASSA-PSS signature algorithms (0806, 0805, and 0804). This will allow for the TLS V1.3 handshake messages to be signed with any supported RSASSA-PSS signature algorithm.

```
GSK_TLS_SIG_ALG_PAIRS=0601050104010301080608050804
```

See Table 30 on page 804 for a list of valid 4-character signature algorithm pair specifications.

Limiting key exchange elliptic curves

When utilizing an ephemeral Elliptic Curve Diffie-Hellman cipher (TLS_ECDHE_*xxx*), each side of the connection being negotiated generates an elliptic curve key pair and exchanges the public key as part of the TLS V1.0, TLS V1.1, or TLS V1.2 handshake process. The elliptic curve is selected by the server using a list of supported elliptic curves provided by the client.

System SSL provides the capability to define the list of supported elliptic curves through either an environment variable or through an invocation to the **gsk_attribute_set_buffer()** routine.

When a client, the list of supported elliptic curves is defined using GSK_CLIENT_ECURVE_LIST. This list represents the elliptic curves supported by the client for the key exchange in the client's preferred order. This list also represents certificate elliptic curves supported when a server is using an elliptic curve public key certificate.

When a server, the list of allowed elliptic curves is defined using GSK_SERVER_ALLOWED_KEX_ECURVES. This list represents the allowed key exchange elliptic curves with no defined order.

When enabled for FIPS mode, elliptic curves x25519 and x448 are not supported and if specified, will be ignored. The specified list may also be tailored to meet the requirements of the FIPS level being utilized. For information about FIPS mode level support, see Chapter 4, "System SSL and FIPS 140-2," on page 19.

For information about default settings and supported elliptic curves, see Table 29 on page 804.

Example of how the elliptic curve is selected by the server using a list of supported elliptic curves provided by the client

The client supports secp256r1 (0023) and secp384r1 (0024), but prefers secp256r1 to be utilized (For the client, the list of supported elliptic curves is defined using GSK_CLIENT_ECURVE_LIST):

```
GSK_CLIENT_ECURVE_LIST=00230024
```

The server supports secp384r1 (0024), x25519 (0029), and secp256r1 (0023) (For the server, the list of allowed curves is defined using GSK_SERVER_ALLOWED_KEX_ECURVES):

```
GSK_SERVER_ALLOWED_KEX_ECURVES=002400290023
```

In the example above, the key exchange process would use secp256r1 (0023) because the elliptic curve selection involves the server looking through the client's provided list for a match in the server's list. The first matching curve is used.

Server certificate domain-based validation

System SSL supports the ability for SSL V3 and TLS client applications to validate the connecting server's certificate during the handshake process. This case-insensitive verification occurs between the client application's provided reference ID list and the server's subject alternative name DNS entry or subject DN common name.

Enabling server certificate domain-based validation

The client application defines reference lists to be compared against the server's certificate. The reference list can be defined at either the SSL environment or connection level when using the **gsk_attribute_set_buffer()** routine. Two types of reference lists can be used:

- GSK_REFERENCE_ID_CN defines the domain name IDs to be compared against the subject DN common name.
- GSK_REFERENCE_ID_DNS defines the domain name IDs to be compared against the DNS entry in the subject alternative name extension.

The values in the reference list must be fully qualified domain names containing at least three labels separated by either a comma or a space. If the values contain a space, comma, or backslash as a part of the value, it can be escaped with a backslash character '\'. There is no validation done to confirm the validity of the client's provided DNS list values, other than verifying the number of labels provided for each ID. The maximum number of characters for the reference list is 16384 characters. The reference values should be specified in the local code page. If any of the values end in a period, the period will be removed prior to comparison, including any values provided from the server's certificate.

The client application can specify the following settings:

- GSK_REFERENCE_ID_DNS
- GSK_REFERENCE_ID_DNS and GSK_REFERENCE_ID_CN
- GSK_REFERENCE_ID_CN

When only the GSK_REFERENCE_ID_DNS is provided, if the server certificate does not contain a subject alternative name DNS name value or none of the values match the server's subject alternative name DNS entry, a handshake failure occurs.

When the GSK_REFERENCE_ID_DNS and GSK_REFERENCE_ID_CN is provided, if the server's certificate contains a subject alternative name DNS name entry, the values in the GSK_REFERENCE_ID_DNS will be used for comparison. Otherwise, the GSK_REFERENCE_ID_CN values will not be used against the subject DN common name and a handshake failure occurs.

When only the GSK_REFERENCE_ID_CN is provided, if the server's certificate contains a subject alternative name DNS name entry or none of the values match the server's subject DN common name, a handshake failure occurs.

Wildcard validation

System SSL allows the domain-based server certificate validation process to accept an asterisk as the wildcard character to replace zero or more characters in the server's certificate subject alternative name DNS entry or subject DN common name. Wildcarding is enabled through the GSK_WILDCARD_VALIDATION_ENABLE setting. For more information, see Appendix A, "Environment variables," on page 763.

The wildcard character can only vouch for characters within the first label of the server certificate's DNS entry value or common name value. The first label is anything before the first period. The wildcard character will not be acceptable to replace zero or more characters within any other label. The wildcard character does not vouch for multiple labels (for example, '*.com' is not an acceptable match to 'example.mycompany.com'). The server certificate's value must contain at least the main domain name and the domain extension. If the wildcard validation is not enabled, the wildcard character will not replace zero or more characters within the server's certificate ID.

By default, wildcard comparison is disabled. If using the **gsk_attribute_set_enum()** routine, set the GSK_WILDCARD_VALIDATION_ENABLE attribute to GSK_WILDCARD_VALIDATION_ENABLE_ON to enable wildcard validation.

Note: GSK_REFERENCE_ID_CN, GSK_REFERENCE_ID_DNS and wildcard matching is based on RFC6125.

Example 1 – Subject alternative name DNS entry validation

The client provides a GSK_REFERENCE_ID_DNS containing 'example1.mycompany.com,example.mycompany.com' and the server certificate contains a subject alternative name DNS entry of 'example.mycompany.com'.

The reference list is parsed until a valid value is found. In this example, a match is found at 'example.mycompany.com'. The matching fully qualified domain name value 'example.mycompany.com' is returned in GSK_REFERENCE_LIST_VALUE_USED.

Because the match was made between the server's subject alternative name DNS value and the GSK_REFERENCE_ID_DNS, the GSK_REFERENCE_LIST_TYPE_USED enumeration variable will contain GSK_REFERENCE_LIST_USED_DNS.

Example 2 – Subject DN common name validation

The client provides GSK_REFERENCE_ID_CN containing 'example1.mycompany.com,example.mycompany.com' and the server's certificate contains no subject alternative name DNS values and the subject DN common name is 'example.mycompany.com'.

The reference list is parsed until a valid value is found. In this example, a match is found at 'example.mycompany.com'. The matching fully qualified domain name value 'example.mycompany.com' is returned in GSK_REFERENCE_LIST_VALUE_USED.

Because the match was made between the server's subject DN common name and the GSK_REFERNCE_ID_CN, the GSK_REFERENCE_LIST_TYPE_USED enumeration variable will contain GSK_REFERENCE_LIST_USED_CN.

Example 3 – Subject alternative name matching using wildcarding

The client provides GSK_REFERENCE_ID_DNS or GSK_REFERENCE_ID_CN containing 'example.mycompany.com'. If the server's certificate contains a subject alternative name DNS entry or subject DN Common Name as '*.mycompany.com', '*ample.mycompany.com', 'ex*.mycompany.com', a successful handshake will occur.

In the example, the wildcard character is replacing zero or more characters within the first label only. The wildcard validation functions the same for the comparison of a server's subject DN common name and the GSK_REFERENCE_ID_CN.

For more information on setting GSK_WILDCARD_VALIDATION_ENABLE , see Appendix A, "Environment variables," on page 763.

Chapter 7. API reference

This topic describes the set of application programming interfaces (APIs) that z/OS System SSL supports for performing secure sockets layer (SSL/TLS) communication.

These APIs were introduced in z/OS Version 1 Release 2 and beyond and supersede the APIs from prior releases. **Only** the APIs in this topic should be used for writing new application programs. Existing application programs should be recoded if possible to use the new APIs. See Appendix E, "Migrating from deprecated SSL interfaces," on page 809 for more information about updating your application programs.

The deprecated APIs included in Chapter 9, "Deprecated Secure Socket Layer (SSL) APIs," on page 513 are for **reference only**. When creating new application programs, you **must not** include any of the deprecated APIs; you should use **only** the APIs in this topic.

These provide more information about X.509 certificates and the Secure Sockets Layer protocol. System SSL only supports the PKCS versions that are indicated in the following list. Make sure that you select the appropriate version of the document on the website.

Note: Copies of ANSI standards can be purchased from the American National Standards Institute (ANSI) (www.ansi.org).

- *ANSI X9.31 - 1998 Digital Certificates Using Reversible Public Key Cryptography for the Financial Services Industry*
- *ANSI X9.62 - Elliptic Curve Digital Signature Algorithm*
- FIPS 186-2: Digital Signature Standard (DSS) (csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf) - less than 1024-bit
- FIPS 186-4: Digital Signature Standard (DSS) (nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf) - 1024-bit and greater
- *PKCS #1, Version 2.1: RSA Encryption Standard*
- *PKCS #3, Version 1.4: Diffie-Hellman Key Agreement Standard*
- *PKCS #5, Version 2.0: Password-based Encryption*
- *PKCS #7, Version 1.5 and 1.6: Cryptographic Message Syntax*
- *PKCS #8, Version 1.2: Private Key Information Syntax*
- *PKCS #10, Version 1.7: Certification Request*
- *PKCS #12, Version 1.0: Personal Information Exchange*
- *RFC 2246: The TLS Protocol Version 1.0* (RFC 2246 (tools.ietf.org/html/rfc2246))
- *RFC 2253: UTF-8 String Representation of Distinguished Names* (RFC 2253 (tools.ietf.org/html/rfc2253))
- *RFC 2279: UTF-8, a transformation format of ISO 10646* (RFC 2279 (tools.ietf.org/html/rfc2279))
- *RFC 2459: Internet x.509 Public Key Infrastructure Certificate and CRL Profile* (RFC 2459 (tools.ietf.org/html/rfc2459))
- *RFC 2560: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP* (RFC 2560 (tools.ietf.org/html/rfc2560))
- *RFC 2587: PKIX LDAP Version 2 Schema* (RFC 2587 (tools.ietf.org/html/rfc2587))
- *RFC 2631: Diffie-Hellman Key Agreement Method* (RFC 2631 (tools.ietf.org/html/rfc2631))
- *RFC 3268: Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)* (RFC 3268 (tools.ietf.org/html/rfc3268))
- *RFC 3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* (RFC 3280 (tools.ietf.org/html/rfc3280))
- *RFC 3852: Cryptographic Message Syntax (CMS)* (RFC 3852 (tools.ietf.org/html/rfc3852)) - Key transport only

- *RFC 4346: The Transport Layer Security (TLS) Protocol Version 1.1* (RFC 4346 (tools.ietf.org/html/rfc4346))
- *RFC 4366: Transport Layer Security (TLS) Extensions (Server Name Indication, Maximum Fragment Length and Truncated HMAC)* (RFC 4366 (tools.ietf.org/html/rfc4366))
- *RFC 4492: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)* (RFC 4492 (tools.ietf.org/html/rfc4492))
- *RFC 5116: An Interface and Algorithms for Authenticated Encryption* (RFC 5116 (tools.ietf.org/html/rfc5116))
- *RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2* (RFC 5246 (tools.ietf.org/html/rfc5246))
- *RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* (RFC 5280 (tools.ietf.org/html/rfc5280))
- *RFC 5288: AES Galois Counter Mode (GCM) Cipher Suites for TLS* (RFC 5288 (tools.ietf.org/html/rfc5288))
- *RFC 5289: TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM))* (RFC 5289 (tools.ietf.org/html/rfc5289))
- *RFC 5430: Suite B Profile for Transport Layer Security (TLS)* (RFC 5430 (tools.ietf.org/html/rfc5430))
- *RFC 5480: Elliptic Curve Cryptography Subject Public Key Information* (RFC 5480 (tools.ietf.org/html/rfc5480))
- *RFC 5746: Transport Layer Security (TLS) Renegotiation Indication Extension* (RFC 5746 (tools.ietf.org/html/rfc5746))
- *RFC 5759: Suite B Certificate and Certificate Revocation List (CRL) Profile* (RFC 5759 (tools.ietf.org/html/rfc5759))
- *RFC 5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF)* (RFC 5869 (tools.ietf.org/html/rfc5869))
- *RFC 6066: Transport Layer Security (TLS) Extensions: Extension Definitions* (RFC 6066 (tools.ietf.org/html/rfc6066))
- *RFC 6460: Suite B Profile for Transport Layer Security (TLS)* (RFC 6460 (tools.ietf.org/html/rfc6460))
- *RFC 6960: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP* (RFC 6960 (tools.ietf.org/html/rfc6960))
- *RFC 7507: TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks* (RFC 7507 (tools.ietf.org/html/rfc7507))
- *RFC 7627: Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension* (RFC 7627 (tools.ietf.org/html/rfc7627))
- *RFC 7748: Elliptic Curves for Security* (RFC 7748 (tools.ietf.org/html/rfc7748))
- *RFC 8017: PKCS #1: RSA Cryptography Specifications Version 2.2* (RFC 8017 (tools.ietf.org/html/rfc8017))
- *RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3* (RFC 8446 (tools.ietf.org/html/rfc8446))

This is a list of APIs. Use these APIs when creating new application programs. If possible, recode your existing application programs to use these APIs as well:

- **gsk_attribute_get_buffer()** (see "gsk_attribute_get_buffer()" on page 76)
- **gsk_attribute_get_cert_info()** (see "gsk_attribute_get_cert_info()" on page 81)
- **gsk_attribute_get_data()** (see "gsk_attribute_get_data()" on page 85)
- **gsk_attribute_get_enum()** (see "gsk_attribute_get_enum()" on page 87)
- **gsk_attribute_get_numeric_value()** (see "gsk_attribute_get_numeric_value()" on page 97)
- **gsk_attribute_set_buffer()** (see "gsk_attribute_set_buffer()" on page 100)
- **gsk_attribute_set_callback()** (see "gsk_attribute_set_callback()" on page 107)

- **gsk_attribute_set_enum()** (see “gsk_attribute_set_enum()” on page 112)
- **gsk_attribute_set_numeric_value()** (see “gsk_attribute_set_numeric_value()” on page 124)
- **gsk_attribute_set_tls_extensions()** (see “gsk_attribute_set_tls_extension()” on page 130)
- **gsk_environment_close()** (see “gsk_environment_close()” on page 133)
- **gsk_environment_init()** (see “gsk_environment_init()” on page 134)
- **gsk_environment_open()** (see “gsk_environment_open()” on page 136)
- **gsk_free_cert_data()** (see “gsk_free_cert_data()” on page 145)
- **gsk_get_all_cipher_suites()** (see “gsk_get_all_cipher_suites()” on page 146)
- **gsk_get_cert_by_label()** (see “gsk_get_cert_by_label()” on page 148)
- **gsk_get_cipher_suites()** (see “gsk_get_cipher_suites()” on page 152)
- **gsk_get_ssl_vector()** (see “gsk_get_ssl_vector()” on page 153)
- **gsk_get_update()** (see “gsk_get_update()” on page 154)
- **gsk_list_free()** (see “gsk_list_free()” on page 155)
- **gsk_secure_socket_close()** (see “gsk_secure_socket_close()” on page 156)
- **gsk_secure_socket_init()** (see “gsk_secure_socket_init()” on page 157)
- **gsk_secure_socket_misc()** (see “gsk_secure_socket_misc()” on page 170)
- **gsk_secure_socket_open()** (see “gsk_secure_socket_open()” on page 173)
- **gsk_secure_socket_read()** (see “gsk_secure_socket_read()” on page 174)
- **gsk_secure_socket_shutdown()** (see “gsk_secure_socket_shutdown()” on page 177)
- **gsk_secure_socket_write()** (see “gsk_secure_socket_write()” on page 179)
- **gsk_strerror()** (see “gsk_strerror()” on page 181)

gsk_attribute_get_buffer()

Gets the value of an attribute buffer.

Format

```
#include <gskssl.h>

gsk_status gsk_attribute_get_buffer (
                                      gsk_handle          ssl_handle,
                                      GSK_BUF_ID          buffer_id,
                                      const char **       buffer_value,
                                      int *               buffer_length)
```

Parameters

ssl_handle
: Specifies an SSL environment handle returned by **gsk_environment_open()** or an SSL connection handle returned by **gsk_secure_socket_open()**.

buffer_id
: Specifies the buffer identifier.

buffer_value
: Returns the address of the buffer value. The buffer is in storage owned by the SSL run time and must not be modified or released by the application. The buffer returned for the GSK_USER_DATA identifier may be modified by the application but must not be released.

buffer_length
: Returns the length of the buffer value.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ATTRIBUTE_INVALID_ID]
: The buffer identifier is not valid or cannot be used with the specified handle.

[GSK_INSUFFICIENT_STORAGE]
: Insufficient storage is available.

[GSK_INVALID_HANDLE]
: The handle is not valid.

[GSK_INVALID_STATE]
: The handle is closed.

Usage

The **gsk_attribute_get_buffer()** routine will return a buffer value for an SSL environment or an SSL connection. The buffer is in storage owned by the SSL run time and must not be released by the application. The address remains valid until the SSL environment or connection is closed or until the application calls the **gsk_attribute_set_buffer()** routine to set a new buffer value.

These buffer identifiers are supported:

GSK_CLIENT_ECURVE_LIST
: Returns the list of elliptic curve specifications or groups supported by the client as a string consisting of 4-character decimal values. See Table 29 on page 804 for a list of valid 4-character elliptic curve or group specifications.

  GSK_CLIENT_ECURVE_LIST may be specified for an SSL environment or an SSL connection.

GSK_CLIENT_TLS_KEY_SHARES
Returns the list of the key share groups that are supported by the client during a TLS V1.3 handshake. See Table 29 on page 804 for a list of valid 4-character key share specifications.

GSK_CLIENT_TLS_KEY_SHARES may be specified for an SSL environment or SSL connection.

GSK_CONNECT_CIPHER_SPEC
Returns the cipher specification selected for an initialized connection. When using the SSL V2 protocol the cipher specification will be returned as a single character. For other protocols the cipher specification may be returned as either a 2-character or 4-character cipher depending on the setting in GSK_V3_CIPHERS. See Table 25 on page 795 for a list of valid SSL V2 cipher specifications. See Table 26 on page 795 and Table 27 on page 799 for a list of valid 2-character and 4-character cipher specifications for the SSL V3 and TLS protocols.

GSK_CONNECT_KEY_SHARE
Returns the selected key share group from a TLS V1.3 handshake. See Table 29 on page 804 for a list of valid 4-character elliptic curve or group specifications. GSK_CONNECT_KEY_SHARE may be specified only for an SSL connection.

GSK_CONNECT_SEC_TYPE
Returns the security protocol for an initialized connection. The value will be "SSLV2", "SSLV3", "TLSV1", "TLSV1.1", "TLSV1.2", or "TLSV1.3" depending upon the protocol selected during the SSL handshake. GSK_CONNECT_SEC_TYPE may be specified only for an SSL connection.

GSK_HTTP_CDP_PROXY_SERVER_NAME
Returns the DNS name or IP address of the HTTP proxy server for HTTP CDP CRL retrieval. GSK_HTTP_CDP_PROXY_SERVER_NAME may be specified only for an SSL environment

GSK_KEYRING_FILE
Returns the name of the key database file, PKCS #12 file, SAF key ring or z/OS PKCS #11 token. A key database or PKCS #12 file is used if a database password is defined using either an environment variable or the **gsk_attribute_set_buffer()** routine. When a stash file is defined, a key database file is used.

GSK_KEYRING_LABEL
Returns the label associated with the certificate being used by the SSL environment or connection. This will be the value set by the application if the environment or connection is not initialized. GSK_KEYRING_LABEL may be specified for an SSL environment or an SSL connection. When querying a server's SSL connection and GSK_SERVER_KEYRING_LABEL_LIST is specified, the returned label name is the one chosen for the connection.

GSK_KEYRING_PW
Returns the password for the key database or PKCS #12 file. A NULL address will be returned after the environment is initialized. GSK_KEYRING_PW may be specified only for an SSL environment.

GSK_KEYRING_STASH_FILE
Returns the name of the key database password stash file. GSK_KEYRING_STASH_FILE may be specified only for an SSL environment.

GSK_LDAP_SERVER
Returns the DNS name or IP address of the LDAP server. GSK_LDAP_SERVER may be specified only for an SSL environment.

GSK_LDAP_USER
Returns the distinguished name to use when connecting to the LDAP server. GSK_LDAP_USER may be specified only for an SSL environment.

GSK_LDAP_USER_PW
Returns the password to use when connecting to the LDAP server. GSK_LDAP_USER_PW may be specified only for an SSL environment.

GSK_OCSP_PROXY_SERVER_NAME
Returns the DNS name or IP address of the OCSP proxy server. GSK_OCSP_PROXY_SERVER_NAME may be specified only for an SSL environment.

GSK_OCSP_REQUEST_SIGALG
Returns the hash and signature algorithm pair to be used to sign OCSP requests as a string consisting of a 4-character value. See Table 31 on page 805 for a list of valid 4-character signature algorithm pair specifications. GSK_OCSP_REQUEST_SIGALG may be specified only for an SSL environment.

GSK_OCSP_REQUEST_SIGKEYLABEL
Returns the certificate label of the key used to sign OCSP requests. GSK_OCSP_REQUEST_SIGKEYLABEL may be specified only for an SSL environment.

GSK_OCSP_RESPONSE_SIGALG_PAIRS
Returns a preference ordered list of hash and signature algorithm pair specifications that are sent on the OCSP request and may be used by the OCSP responder to select an appropriate algorithm for signing the OCSP response.

GSK_OCSP_URL
Returns the URL of the OCSP responder. GSK_OCSP_URL may be specified only for an SSL environment.

GSK_PEER_ID
Returns the Base64-encoded version of the cached session peer ID. GSK_PEER_ID may be specified only for an SSL connection and is only applicable for a client SSL V3, or TLS V1.0 or higher connection when SSL environment enumerator GSK_ENABLE_CLIENT_SET_PEERID is set to ON.

When the SSL connection is not initialized, the GSK_PEER_ID returned is either the peer ID specified on a previous **gsk_attribute_set_buffer()** invocation or NULL.

When the SSL connection is initialized, the GSK_PEER_ID that is returned is either the peer ID data specified on a previous **gsk_attribute_set_buffer()** invocation or the Base64-encoded version of the peer ID data and consists of displayable characters.

When the SSL connection is initialized, the peer ID that is returned can be used as input to the **gsk_attribute_set_buffer()** function to identify the cached session information to be used for a subsequent connection.

For more information about using the GSK_PEER_ID, see Specifying a cached session in the **gsk_secure_socket_init()** usage section.

GSK_REFERENCE_ID_CN
Returns the value or values set to GSK_REFERENCE_ID_CN by the client for validation of the server certificate. GSK_REFERENCE_ID_CN may be specified for an SSL environment or SSL connection.

GSK_REFERENCE_ID_DNS
Returns the value or values set to GSK_REFERENCE_ID_DNS by the client for validation of the server certificate. GSK_REFERENCE_ID_DNS may be specified for an SSL environment or SSL connection.

GSK_REFERENCE_LIST_VALUE_USED
Returns the reference ID value that matched the server certificate’s DNS entry for the subject alternative name or subject DN common name for a successful handshake when reference ID validation was requested. GSK_REFERENCE_LIST_VALUE_USED may be specified only for an SSL connection.

GSK_SERVER_ALLOWED_KEX_ECURVES
Returns the list of elliptic curve specifications that are allowed by the server for the TLS V1.0, TLS V1.1, and TLS V1.2 server key exchange as a string consisting of 4-character decimal values. See Table 29 on page 804 for a list of valid 4-character elliptic curve or group specifications.

GSK_SERVER_ALLOWED_KEX_ECURVES may be specified for an SSL environment or an SSL connection.

GSK_SERVER_KEYRING_LABEL_LIST
Returns the key labels that are available to be used for an SSL server connection.

GSK_SERVER_TLS_KEY_SHARES
Returns the list of the key share groups that are supported by the server during a TLS V1.3 handshake. See Table 29 on page 804 for a list of valid 4-character key share specifications.

GSK_SERVER_TLS_KEY_SHARES may be specified for an SSL environment or SSL connection.

GSK_SID_VALUE

Returns the Base64-encoded version of the session identifier. GSK_SID_VALUE may be specified only for an SSL connection.

When the SSL connection is not initialized, the GSK_SID_VALUE that is returned is either the session ID specified on a previous **gsk_attribute_set_buffer()** invocation or NULL.

When the SSL connection is initialized, the GSK_SID_VALUE that is returned is either the session ID specified on a previous **gsk_attribute_set_buffer()** invocation or the Base64-encoded version of the session identifier and consists of displayable characters.

GSK_SID_VALUE can be used as input to the **gsk_attribute_set_buffer()** function to identify the session information to be used for subsequent server SSL V3, or TLS V1.0 or higher connections.

For more information about using the GSK_SID_VALUE, see Specifying a cached session in the **gsk_secure_socket_init()** usage section.

GSK_SNI_LIST

Returns the address of a list of server names passed to the server by the client for use during server name indication callback routine. Server name indication is an extension to TLS V1.0 or higher protocols which allow the client to pass server names to the server. The server can use the list of server names as an aid in selection of the certificate to be used by the server. GSK_SNI_LIST may be specified only for an SSL connection and only on the server side of the connection. When returned, the buffer contains a list of server names with each server name preceded by a 1-byte name type and a 2-byte field (in large endian format) containing the length of the server name. The name type always contains X'00' to indicate that it is a hostname; however, new name types may be introduced in the future. The server name content will be in UTF-8 format.

GSK_SUITE_B_CIPHER_SPECS

Returns the Suite B cipher specifications configured for the environment as a string consisting of 4-character values. GSK_SUITE_B_CIPHER_SPECS may be specified for an SSL environment after the environment has been initialized. See Table 11 on page 49 for a list of valid suite B cipher specifications.

GSK_TLS_CERT_SIG_ALG_PAIRS

Returns the list of hash and signature algorithm pair specifications that are supported by the client or server as a string consisting of one or more 4-character values in order of preference for use in digital signatures of X.509 certificates. Certificate signature algorithm pair specification only has relevance for TLS V1.2 client or TLS V1.3 client and server sessions. See Table 30 on page 804 for a list of valid 4-character certificate signature algorithm pair specifications.

GSK_TLS_CERT_SIG_ALG_PAIRS may be specified for an SSL environment or an SSL connection.

GSK_TLS_SIG_ALG_PAIRS

Returns the list of hash and signature algorithm pair specifications set by the client or server as a string consisting of one or more 4-character values in order of preference for use in digital signatures of X.509 certificates and TLS handshake messages. Signature algorithm pair specification only has relevance for sessions using TLS V1.2 and TLS V1.3. See Table 30 on page 804 for a list of valid 4-character signature algorithm pair specifications.

GSK_TLS_SIG_ALG_PAIRS may be specified for an SSL environment or an SSL connection.

GSK_USER_DATA

Returns the address of the user data to be passed to SSL exit routines. The application may alter the user data but may not free it. GSK_USER_DATA may be specified only for an SSL connection.

GSK_V2_CIPHER_SPECS

Returns the SSL V2 cipher specifications as a string consisting of 1-character values. GSK_V2_CIPHER_SPECS may be specified for an SSL environment or an SSL connection. See Table 25 on page 795 for a list of valid SSL v2 cipher specifications.

Note: If Suite B support is enabled in the SSL environment, the SSL V2 cipher specifications are ignored. The Suite B ciphers in use in the SSL environment can be retrieved by specifying the GSK_SUITE_B_CIPHER_SPECS buffer identifier.

GSK_V3_CIPHER_SPECS
Returns the SSL V3 and TLS cipher specifications as a string consisting of 2-character values. GSK_V3_CIPHER_SPECS may be specified for an SSL environment or an SSL connection. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 protocols. See Table 26 on page 795 for a list of valid 2-character cipher specifications.

Note: If Suite B support is enabled, the 2-character cipher specifications are ignored. The Suite B ciphers in use in the SSL environment can be retrieved by specifying the GSK_SUITE_B_CIPHER_SPECS buffer identifier.

GSK_V3_CIPHER_SPECS_EXPANDED
Returns the SSL V3 and TLS cipher specifications as a string consisting of 4-character values. GSK_V3_CIPHER_SPECS_EXPANDED may be specified for an SSL environment or an SSL connection. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 protocols. See Table 27 on page 799 for a list of valid 4-character cipher specifications.

Note: If Suite B support is enabled, the 4-character cipher specifications are ignored. The Suite B ciphers in use in the SSL environment can be retrieved by specifying the GSK_SUITE_B_CIPHER_SPECS buffer identifier.

Related topics

- "gsk_attribute_set_buffer()" on page 100
- "gsk_environment_open()" on page 136
- "gsk_secure_socket_open()" on page 173

gsk_attribute_get_cert_info()

Returns certificate information following an SSL handshake.

Format

```
#include <gskssl.h>

gsk_status gsk_attribute_get_cert_info (
                                         gsk_handle                 soc_handle,
                                         GSK_CERT_ID                cert_id,
                                         gsk_cert_data_elem **      cert_data,
                                         int *                      elem_count)
```

Parameters

soc_handle
Specifies the connection handle returned by the **gsk_secure_socket_open()** routine.

cert_id
Specifies the certificate identifier.

cert_data
Returns the certificate data array. The **gsk_free_cert_data()** routine should be called to release the array when the certificate information is no longer needed. A NULL address will be returned if no certificate information is available.

elem_count
Returns the number of elements in the array of gsk_cert_data_elem structures.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ATTRIBUTE_INVALID_ID]
The certificate identifier is not valid.

[GSK_ERR_ASN]
Unable to decode certificate.

[GSK_INSUFFICIENT_STORAGE]
Insufficient storage is available.

[GSK_INVALID_HANDLE]
The connection handle is not valid.

[GSK_INVALID_STATE]
The connection is not initialized.

Usage

The **gsk_attribute_get_cert_info()** routine returns information about certificates used in an SSL handshake. The connection must be in the initialized state. The certificate data address will be NULL if there is no certificate information available.

These certificate identifiers are supported:

GSK_LOCAL_CERT_INFO
Returns information about the local certificate.

GSK_PARTNER_CERT_INFO
Returns information about the partner certificate.

Each element of the certificate data array has an element identifier. The element identifiers used for a particular certificate depend upon the contents of the certificate. These element identifiers are currently provided:

CERT_BODY_BASE64
Certificate body in Base64-encoded format

CERT_BODY_DER
Certificate body in binary ASN.1 DER-encoded format

CERT_COMMON_NAME
Subject common name (CN)

CERT_COUNTRY
Subject country (C)

CERT_DN_DER
Subject distinguished name in binary ASN.1 DER-encoded format

CERT_DN_PRINTABLE
Subject distinguished name as a printable character string

These DN attribute names are recognized by the System SSL run time.

- C - Country
- CN - Common name
- DC - Domain component
- DNQUALIFIER - Distinguished name qualifier
- EMAIL - email address
- GENERATIONQUALIFIER - Generation qualifier
- GIVENNAME - Given name
- INITIALS - Initials
- L - Locality
- MAIL - RFC 822 style address
- NAME - Name
- O - Organization name
- OU - Organizational unit name
- PC - Postal code
- SERIALNUMBER - Serial number
- SN - Surname
- ST - State or province
- STREET - Street
- T - Title

CERT_DNQUALIFIER
Subject distinguished name qualifier (DNQUALIFIER)

CERT_DOMAIN_COMPONENT
Subject domain component (DC)

CERT_EMAIL
Subject email address (EMAIL)

CERT_GENERATIONQUALIFIER
Subject generation qualifier (GENERATIONQUALIFIER)

CERT_GIVENNAME
Subject given name (GIVENNAME)

CERT_INITIALS
Subject initials (INITIALS)

CERT_ISSUER_COMMON_NAME
Issuer common name (CN)

CERT_ISSUER_COUNTRY
Issuer country (C)

CERT_ISSUER_DN_DER
Issuer distinguished name in binary ASN.1 DER-encoded format

CERT_ISSUER_DN_PRINTABLE
Issuer distinguished name as a printable character string

These DN attribute names are recognized by the System SSL run time.

- C - Country
- CN - Common name
- DC - Domain component
- DNQUALIFIER - Distinguished name qualifier
- EMAIL - email address
- GENERATIONQUALIFIER - Generation qualifier
- GIVENNAME - Given name
- INITIALS - Initials
- L - Locality
- MAIL - RFC 822 style address
- NAME - Name
- O - Organization name
- OU - Organizational unit name
- PC - Postal code
- SERIALNUMBER - Serial number
- SN - Surname
- ST - State or province
- STREET - Street
- T - Title

CERT_ISSUER_DNQUALIFIER
Issuer distinguished name qualifier (DNQUALIFIER)

CERT_ISSUER_DOMAIN_COMPONENT
Issuer domain component (DC)

CERT_ISSUER_EMAIL
Issuer email address (EMAIL)

CERT_ISSUER_GENERATIONQUALIFIER
Issuer generation qualifier (GENERATIONQUALIFIER)

CERT_ISSUER_GIVENNAME
Issuer given name (GIVENNAME)

CERT_ISSUER_INITIALS
Issuer initials (INITIALS)

CERT_ISSUER_LOCALITY
Issuer locality (L)

CERT_ISSUER_MAIL
Issuer RFC 822 style address (MAIL)

CERT_ISSUER_NAME
Issuer name (NAME)

CERT_ISSUER_ORG
Issuer organization (O)

CERT_ISSUER_ORG_UNIT
Issuer organizational unit (OU)

CERT_ISSUER_POSTAL_CODE
Issuer postal code (PC)

CERT_ISSUER_SERIALNUMBER
Issuer serial number (SERIALNUMBER)

CERT_ISSUER_STATE_OR_PROVINCE
Issuer state or province (ST)

CERT_ISSUER_STREET
Issuer street (STREET)

CERT_ISSUER_SURNAME
Issuer surname (SN)

CERT_ISSUER_TITLE
Issuer title (T)

CERT_LOCALITY
Subject locality (L)

CERT_MAIL
Subject RFC 822 style address (MAIL)

CERT_NAME
Subject name (NAME)

CERT_ORG
Subject organization (O)

CERT_ORG_UNIT
Subject organizational unit (OU)

CERT_POSTAL_CODE
Subject postal code (PC)

CERT_SERIAL_NUMBER
Certificate serial number

CERT_SERIALNUMBER
Subject serial number (SERIALNUMBER)

CERT_STATE_OR_PROVINCE
Subject state or province (ST)

CERT_STREET
Subject street (STREET)

CERT_SURNAME
Subject surname (SN)

CERT_TITLE
Subject title (T)

The CERT_BODY_DER, CERT_DN_DER, and CERT_ISSUER_DN_DER elements are not null-terminated and the 'cert_data' field must be used to get the element length. All of the other elements are null-terminated character strings and the 'cert_data' field is the length of the string excluding the end-of-string delimiter.

Related topics

- "gsk_secure_socket_init()" on page 157
- "gsk_free_cert_data()" on page 145

gsk_attribute_get_data()

Returns information related to a certificate request.

Format

```
#include <gskssl.h>

gsk_status gsk_attribute_get_data (
                                    gsk_handle      soc_handle,
                                    GSK_DATA_ID     data_id,
                                    void **         data_ptr)
```

Parameters

soc_handle
Specifies the connection handle returned by the **gsk_secure_socket_open()** routine.

data_id
Specifies the data identifier.

data_ptr
Returns the address of the requested data. The address will be NULL if the requested data is not available.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ATTRIBUTE_INVALID_ID]
The data identifier is not valid.

[GSK_ERR_ASN]
Unable to decode certification authority name.

[GSK_ERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[GSK_ERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[GSK_ERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[GSK_ERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[GSK_ERR_MISSING_REQUEST_SIGALGS]
Missing required certificate request signature algorithms.

[GSK_INSUFFICIENT_STORAGE]
Insufficient storage is available.

[GSK_INVALID_HANDLE]
The connection handle is not valid.

[GSK_INVALID_STATE]
The connection is not initialized.

Usage

The **gsk_attribute_get_data()** routine returns information related to a certificate request. The server sends a certificate request to the client as part of the client authentication portion of the SSL handshake. The connection must be in the initialized state.

These data identifiers are supported:

GSK_DATA_ID_SUPPORTED_KEYS
Returns a list of labels for certificates signed by a certification authority that is in the list that is provided by the server. A database entry is included in the list only if it has both a certificate and a private key. Certificates that are expired or are not yet valid are not included in the list. The labels are associated with certificates that are read in from a key database, PKCS #12 file, SAF key ring, or PKCS #12 token when the SSL environment was established (**gsk_environment_init()**). If executing in FIPS mode, the list only includes labels that can be used in FIPS mode. If using the TLS V1.2 protocol, the list includes only those certificates that use the key and signature algorithms supported by the server. If using the TLS V1.3 protocol, the list includes only TLS V1.3 compliant certificates that use the signature algorithms supported by the server. The **gsk_list_free()** routine should be called to release the list when it is no longer needed.

GSK_DATA_ID_SERVER_ISSUERS
Returns a list of distinguished names of certification authorities provided by the server in the certificate request. The **gsk_list_free()** routine should be called to release the list when it is no longer needed.

Related topics

- "gsk_list_free()" on page 155

gsk_attribute_get_enum()

Gets an enumerated value.

Format

```
#include <gskssl.h>

gsk_status gsk_attribute_get_enum (
                                   gsk_handle          ssl_handle,
                                   GSK_ENUM_ID         enum_id,
                                   GSK_ENUM_VALUE *    enum_value)
```

Parameters

ssl_handle
Specifies an SSL environment handle that is returned by **gsk_environment_open()** or an SSL connection handle that is returned by **gsk_secure_socket_open()**.

enum_id
Specifies the enumeration identifier.

enum_value
Returns the enumeration value.

Results

The function return value is 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ATTRIBUTE_INVALID_ID]
The enumeration identifier is not valid or cannot be used with the specified handle.

[GSK_INVALID_HANDLE]
The handle is not valid.

[GSK_INVALID_STATE]
The environment is closed or the SSL connection is established.

Usage

The **gsk_attribute_get_enum()** routine returns an enumerated value for an SSL environment or an SSL connection.

These enumeration identifiers are supported:

GSK_3DES_KEYCHECK
Returns GSK_3DES_KEYCHECK_ON when key parts are compared for uniqueness.

Returns GSK_3DES_KEYCHECK_OFF when key parts are not compared for uniqueness.

GSK_3DES_KEYCHECK can be specified only for an SSL environment.

GSK_AIA_CDP_PRIORITY
Returns GSK_AIA_CDP_PRIORITY_ON to indicate that the AIA extension and GSK_OCSP_URL is queried before examining the CDP extension.

Returns GSK_AIA_CDP_PRIORITY_OFF to indicate that the HTTP URI values specified in the CDP extension is contacted before attempting to contact the OCSP responders in the AIA extension or the OCSP responder that are specified in GSK_OCSP_URL.

GSK_AIA_CDP_PRIORITY can be specified only for an SSL environment.

GSK_CERT_DIAG_INFO

Returns GSK_CERT_DIAG_INFO_FAILURE_ONLY if certificate diagnostic information is only provided for peer validation failures.

Returns GSK_CERT_DIAG_INFO_SUCCESS_ONLY if certificate diagnostic information is only provided for successful peer validations.

Returns GSK_CERT_DIAG_INFO_SUCCESS_OR_FAILURE if certificate diagnostic information is provided for both successful and failing peer validations.

GSK_CERT_DIAG_INFO can be specified for an SSL environment or an SSL connection.

GSK_CERT_VALIDATE_KEYRING_ROOT

Returns GSK_CERT_VALIDATE_KEYRING_ROOT_ON if SAF key ring certificates must be validated to the root CA certificate.

Returns GSK_CERT_VALIDATE_KEYRING_ROOT_OFF if SAF key ring certificates are only validated to the trust anchor certificate.

If a sole intermediate certificate is found in a SAF key ring and the next issuer is not found in the same SAF key ring, the intermediate certificate acts as a trust anchor and the certificate chain is considered complete.

GSK_CERT_VALIDATE_KEYRING_ROOT can only be specified for an SSL environment.

GSK_CERT_VALIDATION_MODE

Returns GSK_CERT_VALIDATION_MODE_2459 if certificate validation is based on the RFC 2459 method, GSK_CERT_VALIDATION_MODE_3280 if certificate validation is based on the RFC 3280 method, and GSK_CERT_VALIDATION_MODE_5280 if certificate validation is based on the RFC 5280 method.

Returns GSK_CERT_VALIDATION_MODE_ANY if certificate validation can use any supported X.509 certificate validation method.

GSK_CERT_VALIDATION_MODE can only be specified for an SSL environment.

GSK_CLIENT_AUTH_ALERT

Returns GSK_CLIENT_AUTH_NOCERT_ALERT_OFF if the SSL server application is configured to allow client connections where client authentication is requested and the client failed to supply an X.509 certificate.

Returns GSK_CLIENT_AUTH_NOCERT_ALERT_ON if the SSL server application is configured to terminate client connections where client authentication is requested and the client failed to supply an X.509 certificate.

GSK_CLIENT_AUTH_ALERT can be specified only for an SSL environment.

GSK_CLIENT_AUTH_TYPE

Returns GSK_CLIENT_AUTH_FULL_TYPE if received certificates are validated by the System SSL runtime and GSK_CLIENT_AUTH_PASSTHRU_TYPE otherwise.

GSK_CLIENT_AUTH_TYPE can be specified only for an SSL environment.

GSK_CLIENT_EPHEMERAL_DH_GROUP_SIZE

Returns GSK_CLIENT_EPHEMERAL_DH_GROUP_SIZE_2048 when a client application wants to enforce a minimum group size of 2048 for each ephemeral DH server handshake.

Returns GSK_CLIENT_EPHEMERAL_DH_GROUP_SIZE_LEGACY when a client application wants to enforce a minimum group size of 1024 for each new server handshake in non-FIPS mode and group size 2048 when operating in FIPS mode.

GSK_CLIENT_EPHEMERAL_DH_GROUP_SIZE can be specified only for an SSL environment.

GSK_CLIENT_EXTENDED_MASTER_SECRET

Returns GSK_CLIENT_EXTENDED_MASTER_SECRET_OFF if the TLS client does not send the extended master secret extension to the server.

Returns GSK_CLIENT_EXTENDED_MASTER_SECRET_ON if the TLS client sends the extended master secret extension to the server, but does not require the server to support the extension.

Returns GSK_CLIENT_EXTENDED_MASTER_SECRET_REQUIRED if the TLS client sends the extended master secret extension to the server and requires the server to support the extension.

GSK_CLIENT_EXTENDED_MASTER_SECRET can be specified for an SSL environment or an SSL connection. This option is only applicable for TLS V1.0, TLS V1.1, and TLS V1.2 handshakes.

GSK_CRL_CACHE_TEMP_CRL

Returns GSK_CRL_CACHE_TEMP_CRL_ON if a temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not exist on the LDAP server.

Returns GSK_CRL_CACHE_TEMP_CRL_OFF if a temporary LDAP CRL cache entry is not added to the LDAP CRL cache when the CRL does not exist on the LDAP server.

GSK_CRL_CACHE_TEMP_CRL can be specified only for an SSL environment.

GSK_CRL_CACHE_EXTENDED

Returns GSK_CRL_CACHE_EXTENDED_ON to indicate that LDAP extended CRL cache support is enabled.

Returns GSK_CRL_CACHE_EXTENDED_OFF to indicate that LDAP basic CRL cache support is enabled.

GSK_CRL_CACHE_EXTENDED can be specified only for an SSL environment.

GSK_CRL_SECURITY_LEVEL

Returns GSK_CRL_SECURITY_LEVEL_LOW if certificate validation does not fail if the LDAP server cannot be contacted.

Returns GSK_CRL_SECURITY_LEVEL_MEDIUM if certificate validation requires the LDAP server to be contactable, but does not require a CRL to be defined.

Returns GSK_CRL_SECURITY_LEVEL_HIGH if certificate validation requires that CRL revocation information is provided from the LDAP server.

GSK_ENABLE_CLIENT_SET_PEERID

Returns GSK_ENABLE_CLIENT_SET_PEERID_ON if the use of a cached peer ID was enabled for the current SSL environment by a call to **gsk_attribute_set_enum()** for this enum.

Returns GSK_ENABLE_CLIENT_SET_PEERID_OFF if the use of a cached peer ID was not enabled for the current SSL environment.

GSK_ENABLE_CLIENT_SET_PEERID is only valid for an SSL V3, TLS V1.0, or higher connections and is only meaningful if specified for a client connection within an SSL environment.

GSK_EXTENDED_MASTER_SECRET_USED

Returns GSK_EXTENDED_MASTER_SECRET_USED_OFF if the extended master secret TLS extension is not negotiated during TLS V1.0, TLS V1.1, or TLS V1.2 handshakes.

Returns GSK_EXTENDED_MASTER_SECRET_USED_ON if the extended master secret TLS extension is negotiated during TLS V1.0, TLS V1.1, or TLS V1.2 handshakes. If a TLS V1.3 handshake is negotiated, GSK_EXTENDED_MASTER_SECRET_USED_ON is returned because the TLS V1.3 protocol incorporates aspects of the extended master secret processing.

GSK_EXTENDED_MASTER_SECRET_USED can be specified only for an SSL connection.

GSK_EXTENDED_RENEGOTIATION_INDICATOR

Returns GSK_EXTENDED_RENEGOTIATION_INDICATOR_OPTIONAL if renegotiation indication is not required during the initial SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 handshake. This is the default.

Returns GSK_EXTENDED_RENEGOTIATION_INDICATOR_CLIENT if the client initial handshake is allowed to proceed only if the server indicates support for RFC 5746 renegotiation.

Returns GSK_EXTENDED_RENEGOTIATION_INDICATOR_SERVER if the server initial handshake is allowed to proceed only if the client indicates support for RFC 5746 renegotiation.

Returns GSK_EXTENDED_RENEGOTIATION_INDICATOR_BOTH if the server and client initial handshakes are allowed to proceed only if the partner indicates support for RFC 5746 renegotiation.

GSK_EXTENDED_RENEGOTIATION_INDICATOR can be specified only for an SSL environment.

GSK_HTTP_CDP_ENABLE

Returns GSK_HTTP_CDP_ENABLE_ON if certificate revocation checking is using the HTTP URI values in the CDP extension to locate an HTTP server.

Returns GSK_HTTP_CDP_ENABLE_OFF if certificate revocation checking is not using the HTTP URI values in the CDP extension.

GSK_HTTP_CDP_ENABLE can be specified only for an SSL environment.

GSK_MIDDLEBOX_COMPAT_MODE

Returns GSK_MIDDLEBOX_COMPAT_MODE_ON if the TLS V1.3 handshake process ought to use or tolerate handshake messages in a manner compliant with earlier TLS protocols to alleviate possible issues with middleboxes or proxies.

Returns GSK_MIDDLEBOX_COMPAT_MODE_OFF if the TLS V1.3 handshake process ought to use the pure TLS V1.3 handshake message format.

GSK_MIDDLEBOX_COMPAT_MODE can be specified only for an SSL environment.

GSK_OCSP_ENABLE

Returns GSK_OCSP_ENABLE_ON if certificate revocation checking is using the HTTP URI values in the AIA extension to locate an OCSP responder.

Returns GSK_OCSP_ENABLE_OFF if certificate revocation checking is not using the HTTP URI values in the AIA extension.

GSK_OCSP_ENABLE can be specified only for an SSL environment.

GSK_OCSP_NONCE_CHECK_ENABLE

Returns GSK_OCSP_NONCE_CHECK_ENABLE_ON if the nonce in the OCSP response is verified to ensure it matches the nonce sent in the OCSP request.

Returns GSK_OCSP_NONCE_CHECK_ENABLE_OFF if checking of the nonce in the OCSP response is disabled.

GSK_OCSP_NONCE_CHECK_ENABLE can be specified only for an SSL environment.

GSK_OCSP_NONCE_GENERATION_ENABLE

Returns GSK_OCSP_NONCE_GENERATION_ENABLE_ON if OCSP requests include a generated nonce.

Returns GSK_OCSP_NONCE_GENERATION_ENABLE_OFF if OCSP nonce generation is disabled.

GSK_OCSP_NONCE_GENERATION_ENABLE can be specified only for an SSL environment.

GSK_OCSP_RETRIEVE_VIA_GET

Returns GSK_OCSP_RETRIEVE_VIA_GET_ON if the HTTP GET request should be used when sending an OCSP request.

Returns GSK_OCSP_RETRIEVE_VIA_GET_OFF if the HTTP request will always be sent via an HTTP POST.

GSK_OCSP_RETRIEVE_VIA_GET can be specified only for an SSL environment.

GSK_OCSP_URL_PRIORITY

Returns GSK_OCSP_URL_PRIORITY_ON if the GSK_OCSP_URL defined responder will first be used and then the responders identified in the AIA extension.

Returns GSK_OCSP_URL_PRIORITY_OFF if the responders identified in the AIA extension will be used first and then the GSK_OCSP_URL defined responder.

GSK_OCSP_URL_PRIORITY can be specified only for an SSL environment.

GSK_PEER_CERT_MIN_VERSION

Returns GSK_PEER_CERT_MIN_VERSION_3 when the partner's X.509 end-entity certificate must be X.509 version 3.

Returns GSK_PEER_CERT_MIN_VERSION_ANY when the partner's X.509 end-entity certificate can be any supported System SSL X.509 version.

GSK_PEER_CERT_MIN_VERSION can be specified only for an SSL environment.

GSK_PROTOCOL_SSLV2

Returns GSK_PROTOCOL_SSLV2_ON if the SSL Version 2 protocol is enabled and GSK_PROTOCOL_SSLV2_OFF if the SSL Version 2 protocol is not enabled.

GSK_PROTOCOL_SSLV2 can be specified for an SSL environment or an SSL connection.

GSK_PROTOCOL_SSLV3

Returns GSK_PROTOCOL_SSLV3_ON if the SSL Version 3 protocol is enabled and GSK_PROTOCOL_SSLV3_OFF if the SSL Version 3 protocol is not enabled.

GSK_PROTOCOL_SSLV3 can be specified for an SSL environment or an SSL connection.

GSK_PROTOCOL_TLSV1

Returns GSK_PROTOCOL_TLSV1_ON if the TLS Version 1 protocol is enabled and GSK_PROTOCOL_TLSV1_OFF if the TLS Version 1 protocol is not enabled.

GSK_PROTOCOL_TLSV1 can be specified for an SSL environment or an SSL connection.

GSK_PROTOCOL_TLSV1_1

Returns GSK_PROTOCOL_TLSV1_1_ON if the TLS Version 1.1 protocol is enabled and GSK_PROTOCOL_TLSV1_1_OFF if the TLS Version 1.1 protocol is not enabled.

GSK_PROTOCOL_TLSV1_1 can be specified for an SSL environment or an SSL connection.

GSK_PROTOCOL_TLSV1_2

Returns GSK_PROTOCOL_TLSV1_2_ON if the TLS Version 1.2 protocol is enabled and GSK_PROTOCOL_TLSV1_2_OFF if the TLS Version 1.2 protocol is not enabled.

GSK_PROTOCOL_TLSV1_2 can be specified for an SSL environment or an SSL connection.

GSK_PROTOCOL_TLSV1_3

Returns GSK_PROTOCOL_TLSV1_3_ON if the TLS Version 1.3 protocol is enabled and GSK_PROTOCOL_TLSV1_3_OFF if the TLS Version 1.3 protocol is not enabled.

GSK_PROTOCOL_TLSV1_3 can be specified for an SSL environment or an SSL connection.

GSK_PROTOCOL_USED

Returns:

- GSK_PROTOCOL_USED_SSLV2 if the SSL Version 2 protocol was used to establish the connection.
- GSK_PROTOCOL_USED_SSLV3 if the SSL Version 3 protocol was used to establish the connection.
- GSK_PROTOCOL_USED_TLSV1 if the TLS Version 1.0 protocol was used to establish the connection.
- GSK_PROTOCOL_USED_TLSV1_1 if the TLS Version 1.1 protocol was used to establish the connection.
- GSK_PROTOCOL_USED_TLSV1_2 if the TLS Version 1.2 protocol was used to establish the connection.
- GSK_PROTOCOL_USED_TLSV1_3 if the TLS Version 1.3 protocol was used to establish the connection.
- GSK_NULL is returned if a connection is not established.

GSK_PROTOCOL_USED can be specified only for an SSL connection.

GSK_REFERENCE_LIST_TYPE_USED

Returns GSK_REFERENCE_LIST_TYPE_USED_CN if the reference ID validation finds a match between the common name reference list to the server certificate’s subject DN common name.

Returns GSK_REFERENCE_LIST_TYPE_USED_DNS if the reference ID validation finds a match between the subject alternative name DNS reference list and one of server certificate’s DNS entries for the subject alternative name.

Returns GSK_REFERENCE_LIST_TYPE_USED_NONE if no reference ID values were provided for the connection.

GSK_REFERENCE_LIST_TYPE_USED may be specified for an SSL connection.

GSK_RENEGOTIATION

Returns GSK_RENEGOTIATION_NONE if SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 handshake renegotiation as a server is disabled, while RFC 5746 renegotiation is allowed. This is the default.

Returns GSK_RENEGOTIATION_DISABLED if SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 handshake renegotiation, including RFC 5746 renegotiation, is disabled.

Returns GSK_RENEGOTIATION_ALL if SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 handshake renegotiation as a server is enabled.

Returns GSK_RENEGOTIATION_ABBREVIATED if SSL V3 and TLS abbreviated handshake renegotiation for resuming the current session only is permitted as a server. RFC 5746 renegotiation is also allowed.

GSK_RENEGOTIATION can only be specified for an SSL environment.

GSK_RENEGOTIATION_PEER_CERT_CHECK

Returns GSK_RENEGOTIATION_PEER_CERT_CHECK_OFF if an identity check against the peer's certificate is not performed during renegotiation. This is the default.

Returns GSK_RENEGOTIATION_PEER_CERT_CHECK_ON if a comparison is performed against the peer's certificate to ensure that certificate does not change during renegotiation.

GSK_RENEGOTIATION_PEER_CERT_CHECK can only be specified for an SSL environment.

GSK_REQ_CACHED_SESSION

Returns GSK_REQ_CACHED_SESSION_ON if a particular cached session peer ID is wanted for an upcoming SSL V3, or TLS V1.0 or higher connection.

Returns GSK_REQ_CACHED_SESSION_OFF if cached session reuse is not active.

GSK_REQ_CACHED_SESSION is only meaningful if specified for a client connection.

GSK_REVOCATION_SECURITY_LEVEL

Returns GSK_REVOCATION_SECURITY_LEVEL_LOW if certificate validation does not fail if the OCSP responder or HTTP server specified in the URI value of the CDP extension cannot be contacted.

Returns GSK_REVOCATION_SECURITY_LEVEL_MEDIUM if certificate validation requires the OCSP responder or the HTTP server specified in the URI value of the CDP extension to be contactable.

Returns GSK_REVOCATION_SECURITY_LEVEL_HIGH if certificate validation requires revocation information is provided by an OCSP responder or HTTP server.

GSK_REVOCATION_SECURITY_LEVEL can be specified only for an SSL environment.

GSK_SERVER_EPHEMERAL_DH_GROUP_SIZE

Returns GSK_SERVER_EPHEMERAL_DH_GROUP_SIZE_2048 when an application wants to enforce a minimum group size of 2048 for each ephemeral DH server handshake.

Returns GSK_SERVER_EPHEMERAL_DH_GROUP_SIZE_LEGACY when an application wants to enforce a minimum group size of 1024 for each new server handshake in non-FIPS mode and group size 2048 when operating in FIPS mode.

Returns GSK_SERVER_EPHEMERAL_DH_GROUP_SIZE_MATCH when an application wants to match the server group size strength to the strength of the server's certificate for each new server handshake.

GSK_SERVER_EPHEMERAL_DH_GROUP_SIZE can be specified only for an SSL environment.

GSK_SERVER_EXTENDED_MASTER_SECRET
Returns GSK_SERVER_EXTENDED_MASTER_SECRET_OFF if the TLS server does not support negotiating the extended master secret extension from clients.

Returns GSK_SERVER_EXTENDED_MASTER_SECRET_ON if the TLS server supports negotiating the extended master secret extension from clients, but does not require the extension.

Returns GSK_SERVER_EXTENDED_MASTER_SECRET_REQUIRED if the TLS server requires negotiating the extended master secret extension from clients.

GSK_SERVER_EXTENDED_MASTER_SECRET can be specified for an SSL environment or an SSL connection. This option is only applicable for TLS V1.0, TLS V1.1, and TLS V1.2 handshakes.

GSK_SERVER_FALLBACK_SCSV
Returns GSK_SERVER_FALLBACK_SCSV_ON to indicate that the server supports the TLS fallback Signaling Cipher Suite Value (SCSV) when included in the client's supported cipher list during an SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 handshake.

Returns GSK_SERVER_FALLBACK_SCSV_OFF to indicate that the server ignores the SCSV when included in the client's supported cipher list during an SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 handshake.

GSK_SERVER_FALLBACK_SCSV can be specified only for an SSL environment.

GSK_SERVER_OCSP_STAPLING
Returns GSK_SERVER_OCSP_STAPLING_OFF if the server is not enabled to contact the configured OCSP responders to retrieve the OCSP responses for the server's end entity certificate or the server's certificate chain.

Returns GSK_SERVER_OCSP_STAPLING_ENDENTITY if the server is enabled to contact the configured OCSP responders to retrieve the OCSP response for the server's end entity certificate.

Returns GSK_SERVER_OCSP_STAPLING_ANY if the server is enabled to contact the configured OCSP responders to retrieve the OCSP responses for the server's end entity certificate or the server's certificate chain.

GSK_SERVER_OCSP_STAPLING can be specified only for an SSL environment.

GSK_SERVER_OCSP_STAPLING_CERTSTATUS
Returns GSK_SERVER_OCSP_STAPLING_CERTSTATUS_ENDENTITY if the server is able to successfully retrieve the OCSP response for the server's end entity certificate and send it back to the client.

Returns GSK_SERVER_OCSP_STAPLING_CERTSTATUS_ANY if the server is able to successfully retrieve the OCSP responses for more than one of the server's certificates in its chain and sends them back to the client.

Returns GSK_SERVER_OCSP_STAPLING_CERTSTATUS_OFF if the server is not configured for OCSP stapling or is unable to successfully retrieve the OCSP responses for the server's end entity certificate or any certificates in the server's chain.

GSK_SERVER_OCSP_STAPLING_CERTSTATUS can be specified only for an SSL connection.

GSK_SESSION_TICKET_CLIENT_ENABLE
Returns GSK_SESSION_TICKET_CLIENT_ENABLE_ON if the client supports caching session tickets received from a server after a TLS V1.3 handshake has completed and supports TLS V1.3 resumption attempts to the server.

Returns GSK_SESSION_TICKET_CLIENT_ENABLE_OFF if the client does not support caching session tickets received from a server after a TLS V1.3 handshake has completed and does not support TLS V1.3 resumption attempts to the server.

GSK_SESSION_TICKET_CLIENT_ENABLE can be specified only for an SSL environment.

GSK_SESSION_TICKET_SERVER_ALGORITHM
Returns GSK_SESSION_TICKET_SERVER_ALGORITHM_AESCBC128 if the server is using 128-bit AES-CBC encryption for session tickets.

Returns GSK_SESSION_TICKET_SERVER_ALGORITHM_AESCBC256 if the server is using 256-bit AES-CBC encryption for session tickets.

GSK_SESSION_TICKET_SERVER_ALGORITHM can be specified only for an SSL environment.

GSK_SESSION_TICKET_SERVER_ENABLE

Returns GSK_SESSION_TICKET_SERVER_ENABLE_ON if the server supports sending session tickets after a TLS V1.3 handshake has completed and if it will accept resumption attempts from the client.

Returns GSK_SESSION_TICKET_SERVER_ENABLE_OFF if the server does not support sending tickets after a TLS V1.3 handshake has completed and will not accept resumption attempts from the client.

GSK_SESSION_TICKET_SERVER_ENABLE can be specified only for an SSL environment.

GSK_SESSION_TYPE

Returns GSK_CLIENT_SESSION if the SSL handshake is to be performed as a client, GSK_SERVER_SESSION if the SSL handshake is to be performed as a server, or GSK_SERVER_SESSION_WITH_CL_AUTH if the SSL handshake is to be performed as a server requiring client authentication.

GSK_SESSION_TYPE can be specified for an SSL environment or an SSL connection.

GSK_SID_FIRST

Returns GSK_SID_IS_FIRST if a full SSL/TLS handshake was performed to establish the connection or GSK_SID_NOT_FIRST if an existing session was used to establish the connection.

GSK_NULL is returned if a connection is not established.

GSK_SID_FIRST can be specified only for an SSL connection.

GSK_SUITE_B_PROFILE

Returns the Suite B for TLS profile setting. Returns:

- GSK_SUITE_B_PROFILE_128 if the 128-bit Suite B security profile is being applied by the SSL client or server to TLS sessions.
- GSK_SUITE_B_PROFILE_128MIN if the 128-bit minimum Suite B security profile is being applied by the SSL client or server to TLS sessions.
- GSK_SUITE_B_PROFILE_192 if the 192-bit Suite B security profile is being applied by the SSL client or server to TLS sessions.
- GSK_SUITE_B_PROFILE_192MIN if the 192-bit minimum Suite B security profile is being applied by the SSL client or server to TLS sessions.
- GSK_SUITE_B_PROFILE_ALL if either the 128-bit or 192-bit Suite B security profile is allowed by the SSL client or server for TLS sessions.
- GSK_SUITE_B_PROFILE_OFF if there is no Suite B security profile being applied by the SSL client or server to TLS sessions.

GSK_SUITE_B_PROFILE can be specified only for an SSL environment.

GSK_SYSPLEX_SESSION_TICKET_CACHE

Returns GSK_SYSPLEX_SESSION_TICKET_CACHE_ON if sysplex session ticket caching for TLS V1.3 sessions is enabled for this SSL environment within this server application.

Returns GSK_SYSPLEX_SESSION_TICKET_CACHE_OFF if sysplex session ticket caching for TLS V1.3 sessions is not enabled for this SSL environment within this server application.

GSK_SYSPLEX_SESSION_TICKET_CACHE can be specified only for an SSL environment.

GSK_SYSPLEX_SIDCACHE

Returns GSK_SYSPLEX_SIDCACHE_ON if sysplex session caching for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 sessions is enabled for this SSL environment within this server application.

Returns GSK_SYSPLEX_SIDCACHE_OFF if sysplex session caching for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 sessions is not enabled for this SSL environment within this server application.

GSK_SYSPLEX_SIDCACHE can be specified only for an SSL environment.

GSK_T61_AS_LATIN1

Returns GSK_T61_AS_LATIN1_ON if the ISO8859-1 character set is used when converting a string tagged as TELETEXSTRING or GSK_T61_AS_LATIN1_OFF if the T.61 character set is used.

GSK_T61_AS_LATIN1 can be specified only for an SSL environment.

The GSK_T61_AS_LATIN1 setting is global and applies to all SSL environments.

GSK_TLS_CBC_PROTECTION_METHOD

Returns GSK_TLS_CBC_PROTECTION_METHOD_NONE to indicate that no CBC protection is enabled.

Returns GSK_TLS_CBC_PROTECTION_METHOD_ZEROBYTEFRAGMENT to indicate that zero byte record fragmenting is enabled.

Returns GSK_TLS_CBC_PROTECTION_METHOD_ONEBYTEFRAGMENT to indicate that one byte record fragmenting is enabled.

GSK_TLS_CBC_PROTECTION_METHOD can be specified only for an SSL environment.

GSK_TLSEXT_MFL

Returns GSK_TLSEXT_MFL_OFF if the Maximum Fragment Length type TLS extension is not negotiated, and the SSL connection is therefore using the default fragment length (16384 bytes).

Returns GSK_TLSEXT_MFL_512, GSK_TLSEXT_MFL_1024, GSK_TLSEXT_MFL_2048 or GSK_TLSEXT_MFL_4096 if the Maximum Fragment Length type TLS extension is negotiated, where the returned value reflects the negotiated maximum fragment length.

GSK_TLSEXT_MFL can be specified only for an SSL connection.

GSK_TLSEXT_SERVER_OCSP_STAPLING

Returns GSK_TLSEXT_SERVER_OCSP_STAPLING_ON if at least one of the OCSP stapling TLS extensions Certificate Status Request or Multiple Certificate Status Request are negotiated and in use.

Returns GSK_TLSEXT_SERVER_OCSP_STAPLING_OFF if the OCSP stapling TLS extensions Certificate Status Request and Multiple Certificate Status Request are not negotiated.

GSK_TLSEXT_SERVER_OCSP_STAPLING can be specified only for an SSL connection.

GSK_TLSEXT_SNI

Returns GSK_TLSEXT_SNI_ON if the Server Name Indication type TLS extension is negotiated and is in use.

Returns GSK_TLSEXT_SNI_OFF if the Server Name Indication type TLS extension is not negotiated.

GSK_TLSEXT_SNI can be specified only for an SSL connection.

GSK_TLSEXT_THMAC

Returns GSK_TLSEXT_THMAC_ON if the Truncated HMAC type TLS extension is negotiated and is in use.

Returns GSK_TLSEXT_THMAC_OFF if the Truncated HMAC type TLS extension is not negotiated.

GSK_TLSEXT_THMAC can be specified only for an SSL connection.

GSK_V3_CIPHERS

Returns GSK_V3_CIPHERS_CHAR2 if two character V3 cipher specifications are in use.

Returns GSK_V3_CIPHERS_CHAR4 if four character V3 cipher specifications are in use.

GSK_V3_CIPHERS can be specified for an SSL environment or an SSL connection.

GSK_WILDCARD_VALIDATION_ENABLE

Returns GSK_WILDCARD_VALIDATION_ENABLE_ON if the client has indicated that the server validation using a reference list will accept an asterisk as the wildcard character. The wildcard character will be acceptable to replace zero or more characters in the server certificate's subject alternative name DNS or subject DN common name values.

Returns GSK_WILDCARD_VALIDATION_ENABLE_OFF if the client has indicated that the server validation using a reference list will not accept an asterisk as the wildcard character. The wildcard character will not replace zero or more characters in the server certificate's subject alternative name DNS or subject DN common name values.

GSK_WILDCARD_VALIDATION_ENABLE may be specified for an SSL environment or SSL connection.

Related topics

- "gsk_attribute_set_enum()" on page 112
- "gsk_environment_open()" on page 136
- "gsk_secure_socket_open()" on page 173

gsk_attribute_get_numeric_value()

Gets a numeric value.

Format

```
#include <gskssl.h>

gsk_status gsk_attribute_get_numeric_value (
                                             gsk_handle     ssl_handle,
                                             GSK_NUM_ID     num_id,
                                             int *          num_value)
```

Parameters

ssl_handle
: Specifies an SSL environment handle returned by **gsk_environment_open()** or an SSL connection handle returned by **gsk_secure_socket_open()**.

num_id
: Specifies the numeric identifier.

num_value
: Returns the numeric value.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ATTRIBUTE_INVALID_ID]
: The numeric identifier is not valid or cannot be used with the specified handle.

[GSK_INVALID_HANDLE]
: The handle is not valid.

[GSK_INVALID_STATE]
: The environment is closed.

Usage

The **gsk_attribute_get_numeric_value()** routine will return a numeric value for an SSL environment or an SSL connection.

These numeric identifiers are supported:

GSK_CRL_CACHE_ENTRY_MAXSIZE
: Returns the maximum size in bytes of a CRL that is allowed to be stored in the LDAP CRL cache. Any CRLs larger than this size are not cached. GSK_CRL_CACHE_ENTRY_MAXSIZE can be specified only for an SSL environment.

GSK_CRL_CACHE_SIZE
: Returns the maximum number of CRLs that are allowed to be stored in the LDAP CRL cache. GSK_CRL_CACHE_SIZE can be specified only for an SSL environment.

GSK_CRL_CACHE_TEMP_CRL_TIMEOUT
: Returns the time in hours that a temporary CRL cache entry resides in the LDAP extended CRL cache. GSK_CRL_CACHE_TEMP_CRL_TIMEOUT can be specified only for an SSL environment.

GSK_CRL_CACHE_TIMEOUT
: Returns the LDAP basic CRL cache timeout. GSK_CRL_CACHE_TIMEOUT can be specified only for an SSL environment.

GSK_FD
Returns the socket descriptor used for network operations. GSK_FD can be specified only for an SSL connection.

GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE
Returns the maximum size in bytes of a CRL that is allowed to be stored in the HTTP CDP CRL cache. GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE can be specified only for an SSL environment.

GSK_HTTP_CDP_CACHE_SIZE
Returns the maximum number of CRLs that are allowed to be stored in the HTTP CDP CRL cache. GSK_HTTP_CDP_CACHE_SIZE can be specified only for an SSL environment.

GSK_HTTP_CDP_MAX_RESPONSE_SIZE
Returns the maximum size in bytes accepted as a response from an HTTP server when retrieving a CRL. GSK_HTTP_CDP_MAX_RESPONSE_SIZE can be specified only for an SSL environment.

GSK_HTTP_CDP_PROXY_SERVER_PORT
Returns the HTTP proxy server port for HTTP CDP CRL retrieval. GSK_HTTP_CDP_PROXY_SERVER_PORT can be specified only for an SSL environment.

GSK_HTTP_CDP_RESPONSE_TIMEOUT
Returns the time in seconds to wait for a response from the HTTP server. GSK_HTTP_CDP_RESPONSE_TIMEOUT can be specified only for an SSL environment.

GSK_LDAP_RESPONSE_TIMEOUT
Returns the time in seconds to wait for a response from the LDAP server. GSK_LDAP_RESPONSE_TIMEOUT can be specified only for an SSL environment.

GSK_LDAP_SERVER_PORT
Returns the LDAP server port. GSK_LDAP_SERVER_PORT can be specified only for an SSL environment.

GSK_MAX_SOURCE_REV_EXT_LOC_VALUES
Returns the maximum number of location values that will be contacted for each revocation source when performing validation of a certificate. GSK_MAX_SOURCE_REV_EXT_LOC_VALUES can be specified only for an SSL environment.

GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES
Returns the maximum number of location values that will be contacted when performing validation of a certificate. GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES can be specified only for an SSL environment.

GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE
Returns the maximum number of OCSP responses or cached certificate statuses that are allowed to be kept in the OCSP response cache for an issuing CA certificate. GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE can be specified only for an SSL environment.

GSK_OCSP_CLIENT_CACHE_SIZE
Returns the maximum number of OCSP responses or cached certificate statuses to be kept in the OCSP response cache. GSK_OCSP_CLIENT_CACHE_SIZE can only be specified for an SSL environment.

GSK_OCSP_MAX_RESPONSE_SIZE
Returns the maximum size in bytes allowed in a response from an OCSP responder. GSK_OCSP_MAX_RESPONSE_SIZE can be specified only for an SSL environment.

GSK_OCSP_NONCE_SIZE
Returns the size in bytes of the nonce that will be sent in OCSP requests. GSK_OCSP_NONCE_SIZE can be specified only for an SSL environment.

GSK_OCSP_PROXY_SERVER_PORT
Returns the port for the OCSP responder proxy server. GSK_OCSP_PROXY_SERVER_PORT can be specified only for an SSL environment.

GSK_OCSP_RESPONSE_TIMEOUT
Returns the time in seconds to wait for a response from the OCSP responder. GSK_OCSP_RESPONSE_TIMEOUT can be specified only for an SSL environment.

GSK_PEER_DH_MIN_KEY_SIZE
Returns the X.509 certificate Diffie-Hellman minimum allowed key size for the peer end-entity certificate. GSK_PEER_DH_MIN_KEY_SIZE can be specified only for an SSL environment.

GSK_PEER_DSA_MIN_KEY_SIZE
Returns the X.509 certificate DSA minimum allowed key size for the peer end-entity certificate. GSK_PEER_DSA_MIN_KEY_SIZE can be specified only for an SSL environment.

GSK_PEER_ECC_MIN_KEY_SIZE
Returns the X.509 certificate ECC minimum allowed key size for the peer end-entity certificate. GSK_PEER_ECC_MIN_KEY_SIZE can be specified only for an SSL environment.

GSK_PEER_RSA_MIN_KEY_SIZE
Returns the X.509 certificate RSA minimum allowed key size for the peer end-entity certificate. GSK_PEER_RSA_MIN_KEY_SIZE can be specified only for an SSL environment.

GSK_SESSION_TICKET_CLIENT_MAXCACHED
Returns the maximum number of session tickets that are allowed to be cached by the client for each unique TLS V1.3 session.

GSK_SESSION_TICKET_CLIENT_MAXSIZE
Returns the maximum size in bytes of a session ticket that is allowed to be stored in the client session ticket cache. GSK_SESSION_TICKET_CLIENT_MAXSIZE can be specified only for an SSL environment.

GSK_SESSION_TICKET_SERVER_COUNT
Returns the number of session tickets that will be sent by the server after the initial TLS V1.3 handshake has completed. GSK_SESSION_TICKET_SERVER_COUNT can be specified only for an SSL environment.

GSK_SESSION_TICKET_SERVER_KEY_REFRESH
Returns the key refresh interval in seconds of the encryption key used by the server to encrypt session tickets. GSK_SESSION_TICKET_SERVER_KEY_REFRESH can be specified only for an SSL environment.

GSK_SESSION_TICKET_SERVER_TIMEOUT
Returns the maximum time that a server accepts a TLS V1.3 session resumption request from the client measured in seconds from the initial handshake. GSK_SESSION_TICKET_SERVER_TIMEOUT can be specified only for an SSL environment.

GSK_V2_SESSION_TIMEOUT
Returns the SSL Version 2 session timeout. GSK_V2_SESSION_TIMEOUT can be specified only for an SSL environment.

GSK_V2_SIDCACHE_SIZE
Returns the size of the SSL Version 2 session identifier cache. GSK_V2_SIDCACHE_SIZE can be specified only for an SSL environment.

GSK_V3_SESSION_TIMEOUT
Returns the session timeout value in seconds for the SSL V3 to TLS V1.2 session identifiers and TLS V1.3 session tickets in the cache. GSK_V3_SESSION_TIMEOUT can be specified only for an SSL environment.

GSK_V3_SIDCACHE_SIZE
Returns the size in number of entries in the SSL V3 to TLS V1.2 session identifier and TLS V1.3 session ticket cache. GSK_V3_SIDCACHE_SIZE can be specified only for an SSL environment.

Related topics

- "gsk_attribute_set_numeric_value()" on page 124
- "gsk_environment_open()" on page 136
- "gsk_secure_socket_open()" on page 173

gsk_attribute_set_buffer()

Sets the value of an attribute buffer.

Format

```
#include <gskssl.h>

gsk_status gsk_attribute_set_buffer (
                                       gsk_handle        ssl_handle,
                                       GSK_BUF_ID        buffer_id,
                                       const char *      buffer_value,
                                       int               buffer_length)
```

Parameters

ssl_handle
Specifies an SSL environment handle returned by **gsk_environment_open()** or an SSL connection handle returned by **gsk_secure_socket_open()**.

buffer_id
Specifies the buffer identifier.

buffer_value
Specifies the buffer value.

buffer_length
Specifies the buffer length. Specify 0 for this parameter if the buffer value is a null-delimited character string.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it is one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ATTRIBUTE_INVALID_ID]
The buffer identifier is not valid or cannot be used with the specified handle.

[GSK_ATTRIBUTE_INVALID_LENGTH]
The buffer length is not valid.

[GSK_INSUFFICIENT_STORAGE]
Insufficient storage is available.

[GSK_INVALID_HANDLE]
The handle is not valid.

[GSK_INVALID_STATE]
The environment or connection is not in the open state.

Usage

The **gsk_attribute_set_buffer()** routine sets a buffer value in an SSL environment or an SSL connection. The environment or connection must be in the open state and not in the initialized state (that is, **gsk_environment_init()** or **gsk_secure_socket_init()** has not been called).

The values set using this service are treated as independent values. They are not validated with other values set using **gsk_attribute_set_buffer()**, **gsk_attribute_set_enum()**, or **gsk_attribute_set_tls_extensions()** APIs until used together to perform an SSL/TLS handshake by calling **gsk_secure_socket_init()**.

If the input *buffer_value* is NULL, the target attribute is set to NULL.

These buffer identifiers are supported:

GSK_CLIENT_ECURVE_LIST

Specifies the list of elliptic curves or supported groups that are supported by the client as a string consisting of 1 or more 4-character decimal values in order of preference for use. GSK_CLIENT_ECURVE_LIST may be specified for an SSL environment or an SSL connection.

For TLS V1.0, TLS V1.1, and TLS V1.2 protocols, this list is used by the client to guide the server to the elliptic curves that are preferred when using ECC-based cipher suites. For the TLS V1.3 protocol, this list is used by the client to guide the server to the elliptic curves that are preferred and guide group selection to encrypt and decrypt TLS V1.3 handshake messages. This attribute is ignored when Suite B has been enabled for the client. The supported elliptic curve values are defined by the Suite B profile being used.

Only NIST recommended curves along with x25519 (0029) and x448 (0030) are able to be specified for this attribute. If x25519 or x448 is specified along with TLS V1.0, TLS V1.1, or TLS V1.2 and the partner server is using an ECDSA certificate, at least one other curve must be specified to match the elliptic curve in the partner's certificate. System SSL does not support x25519 or x448 certificates. If the application is only enabled for TLS V1.3, the secp192r1 (0019) and secp224r1 (0021) elliptic curves or supported groups are ignored. See Table 29 on page 804 for a list of valid 4-character elliptic curve and supported groups specifications.

To use Brainpool standard curve certificates for an SSL/TLS connection, the buffer must be reinitialized to NULL (empty) using either **gsk_attribute_set_buffer()** or the GSK_CLIENT_ECURVE_LIST environment variable. Brainpool certificates cannot be used in FIPS mode or if the selected protocol is TLS V1.3.

When TLS V1.3 is enabled, a NULL (empty) list is not allowed.

When executing in FIPS mode, a NULL (empty) list should not be utilized. A NULL (empty) list may result in an elliptic curve being selected or utilized by the partner that is not supported in FIPS mode.

When executing in FIPS mode, non-FIPS elliptic curves will be removed to meet the requirements of the FIPS level being utilized. For information about FIPS mode level support, see Chapter 4, "System SSL and FIPS 140-2," on page 19.

GSK_CLIENT_TLS_KEY_SHARES

Specifies the list of the key share groups that are supported by the client during a TLS V1.3 handshake. During a TLS V1.3 handshake, the client sends the key share groups that are in common and in the same order as the supported groups list (GSK_CLIENT_ECURVE_LIST). The server selects a group from the client's preferred order and the ones that it supports. The client and server use the selected group to encrypt and decrypt TLS V1.3 handshake messages. See Table 29 on page 804 for a list of valid 4-character key share specifications.

GSK_CLIENT_TLS_KEY_SHARES may be specified for an SSL environment or SSL connection.

GSK_HTTP_CDP_PROXY_SERVER_NAME

Specifies the DNS name or IP address of the HTTP proxy server for HTTP CDP CRL retrieval. GSK_HTTP_CDP_PROXY_SERVER_NAME can be specified only for an SSL environment.

GSK_KEYRING_FILE

Specifies the name of the key database file, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. A key database or PKCS #12 file is used if a database password is defined using either an environment variable or the **gsk_attribute_set_buffer()** routine. When a stash file is defined, a key database file is used. Otherwise, a SAF key ring or z/OS PKCS #11 token is used. GSK_KEYRING_FILE may be specified only for an SSL environment.

The SAF key ring name is specified as "userid/keyring". The current user ID is used if the user ID is omitted. The user must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by the user. The user must have UPDATE access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by another user.

A z/OS PKCS #11 token name is specified as *TOKEN*/token-name. *TOKEN* indicates a PKCS #11 token is being specified.

GSK_KEYRING_LABEL
Specifies the label of the key that is used to authenticate the application. The default key is used if a key label is not specified. GSK_KEYRING_LABEL may be specified for an SSL environment or an SSL connection. If either the GSK_CLIENT_CERT_CALLBACK function or the GSK_SNI_CALLBACK function is registered, the key label can be set or reset by the callback function after a call to **gsk_secure_socket_init()**.

If GSK_KEYRING_LABEL is specified along with GSK_SERVER_KEYRING_LABEL_LIST, GSK_KEYRING_LABEL is used when an SSL V2 secure connection is being established. Otherwise, GSK_KEYRING_LABEL is ignored.

GSK_KEYRING_PW
Specifies the password for the key database or PKCS #12 file. GSK_KEYRING_PW may be specified only for an SSL environment.

GSK_KEYRING_STASH_FILE
Specifies the name of the key database password stash file. The stash file name always has an extension of ".sth" and the supplied name is changed if it does not have the correct extension. The GSK_KEYRING_PW value is used instead of the GSK_KEYRING_STASH value if it is also specified. GSK_KEYRING_STASH_FILE may be specified only for an SSL environment.

GSK_LDAP_SERVER
Specifies one or more blank-separated LDAP server host names. Each host name can contain an optional port number that is separated from the host name by a colon. GSK_LDAP_SERVER may be specified only for an SSL environment. The LDAP server is used to obtain CA certificates when validating a certificate and the local database does not contain the required certificate. The local database must contain the required certificates if no LDAP server is specified. Even when an LDAP server is used, root CA certificates must be found in the local database since the LDAP server is not a trusted data source. The LDAP server is also used to obtain certificate revocation lists.

GSK_LDAP_USER
Specifies the distinguished name to use when connecting to the LDAP server. GSK_LDAP_USER may be specified only for an SSL environment.

GSK_LDAP_USER_PW
Specifies the password to use when connecting to the LDAP server. GSK_LDAP_USER_PW may be specified only for an SSL environment.

GSK_OCSP_PROXY_SERVER_NAME
Specifies the DNS name or IP address of the OCSP proxy server. GSK_OCSP_PROXY_SERVER_NAME can be specified only for an SSL environment.

GSK_OCSP_REQUEST_SIGALG
Specifies the hash and signature algorithm pair to be used to sign OCSP requests as a string consisting of a 4-character value. See Table 31 on page 805 for a list of valid 4-character signature algorithm pair specifications. Defaults to RSA with SHA256 ('0401').

Only requests sent to the OCSP responder identified via the GSK_OCSP_URL setting are signed. GSK_OCSP_REQUEST_SIGALG can be specified only for an SSL environment.

GSK_OCSP_REQUEST_SIGKEYLABEL
Specifies the label of the key to be used to sign OCSP requests. OCSP requests are only signed when a key label is specified.

Only requests sent to the OCSP responder identified via the GSK_OCSP_URL setting are signed (not ones selected from a certificate AIA extension). GSK_OCSP_REQUEST_SIGKEYLABEL can be specified only for an SSL environment.

GSK_OCSP_RESPONSE_SIGALG_PAIRS
Specifies a preference ordered list of hash and signature algorithm pair specifications that are sent on the OCSP request and may be used by the OCSP responder to select an appropriate algorithm for signing the OCSP response. The string consists of one or more 4-character values in order of preference for use. If specified, the OCSP response must be signed with one of these hash and signature algorithm pairs and if it is not, the OCSP response is rejected. It should be noted that not all OCSP responders support the preference ordered list and the OCSP response may be signed by

a signature algorithm that was not specified. These signature algorithm pair specifications only have relevance when OCSP is enabled in the application. See Table 31 on page 805 for a list of valid 4-character signature algorithm pair specifications.

GSK_OCSP_RESPONSE_SIGALG_PAIRS can be specified only for an SSL environment.

GSK_OCSP_URL

Specifies the URL of an OCSP responder. The OCSP responder is used to obtain certificate revocation status during certificate validation. A certificate does not need an AIA extension if a responder URL is configured using this option.

The value must conform to the definition of an HTTP URL:

```
http_URL = "http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]
```

where host can be an IPv4 or IPv6 address, or a domain name.

If GSK_OCSP_URL is specified, GSK_OCSP_ENABLE is set ON, and GSK_OCSP_URL_PRIORITY is set ON, the order that the responders are used is GSK_OCSP_URL defined responder first and then the responders identified in the AIA extension.

If GSK_OCSP_URL is specified, GSK_OCSP_ENABLE is set ON, and GSK_OCSP_URL_PRIORITY is set OFF, the order that the responders are used is the responders identified in the AIA extension first and then the GSK_OCSP_URL defined responder.

GSK_OCSP_URL can be specified only for an SSL environment.

GSK_PEER_ID

Specify the GSK_PEER_ID to uniquely identify the cached session to be used to resume an existing session or duplicate an existing session. Use **gsk_attribute_get_buffer()** to initially retrieve the GSK_PEER_ID to be used by **gsk_attribute_set_buffer()**.

GSK_PEER_ID may be specified only for an SSL connection and is only applicable for client SSL V3, TLS V1.0, or higher connections when SSL environment enumerator GSK_ENABLE_CLIENT_SET_PEERID is ON and V3 session caching is enabled (GSK_V3_SIDCACHE_SIZE is non-zero, GSK_V3_SESSION_TIMEOUT is non-zero, and GSK_SESSION_TICKET_CLIENT_ENABLE is set to ON).

For more information about using the GSK_PEER_ID, see Specifying a cached session in the **gsk_secure_socket_init()** usage section.

GSK_REFERENCE_ID_CN

Specifies a value or list of values to compare against the server's certificate subject DN common name value or values. Provided values must be fully qualified domain names containing at least three labels.

GSK_REFERENCE_ID_CN can be constructed as a comma or space separated list of common name values. If the value includes a comma, space, or backslash, it must be prepended with a backslash. The maximum number of characters for the reference list is 16384. If any of the reference CN values or the server's subject DN common names end in a period, the period will be removed before comparison.

For more information about using the GSK_REFERENCE_ID_CN, see "Server certificate domain-based validation" on page 70.

GSK_REFERENCE_ID_CN may be specified for an SSL environment or SSL connection.

GSK_REFERENCE_ID_DNS

Specifies a value or list of values to compare the server's certificate subject alternative name DNS value or values. Provided values must be fully qualified domain names containing at least three labels.

GSK_REFERENCE_ID_DNS can be constructed as a comma or space separated list of DNS values. If the value includes a comma, space, or backslash, it must be prepended with a backslash. The maximum number of characters for the reference list is 16384. If any of the reference DNS values or the server certificate's DNS entry for the subject alternative name end in a period, the period will be removed before comparison.

For more information about using the GSK_REFERENCE_ID_DNS, see "Server certificate domain-based validation" on page 70.

GSK_REFERENCE_ID_DNS may be specified for an SSL environment or SSL connection.

GSK_SERVER_ALLOWED_KEX_ECURVES

Specifies the list of elliptic curve specifications that are allowed by the server for the TLS V1.0, TLS V1.1, and TLS V1.2 server key exchange when using ECDHE-based cipher suites as a string consisting of one or more 4-character decimal values.

This list is used by the server to limit which elliptic curves can be used for the handshake key exchange when an ephemeral Elliptic Curve Diffie-Hellman (ECDHE) cipher is utilized. This attribute is ignored when Suite B has been enabled for the server. The allowed elliptic curve values are defined by the Suite B profile being used. If the server is enabled for sysplex session ID caching (GSK_SYSPLEX_SIDCACHE is set to ON), like servers in the sysplex must all be configured the same when running on V2R4 or higher with the PTFs for APAR OA61783 applied and active to minimize full handshakes

See Table 29 on page 804 for a list of valid 4-character elliptic curve and group specifications.

The specified list will be tailored to meet the requirements of the FIPS level being utilized. For information about FIPS mode level support, see Chapter 4, "System SSL and FIPS 140-2," on page 19.

GSK_SERVER_ALLOWED_KEX_ECURVE may be specified for an SSL environment or an SSL connection.

GSK_SERVER_KEYRING_LABEL_LIST

Specifies a list of labels that can be selected from in order to authenticate a server application. The list can consist of 1 to 8 key labels each delimited using a comma or a blank space. If either a comma or a blank space appears in the label name, the backslash (\) character must be used as an escape character. Each key label cannot be greater than 127 characters minus any escaping characters. GSK_SERVER_KEYRING_LABEL_LIST may be specified for an SSL environment or an SSL connection. See "Using server multiple key label support" on page 59 for additional information about using GSK_SERVER_KEYRING_LABEL_LIST.

GSK_SERVER_TLS_KEY_SHARES

Specifies the list of the key share groups that are supported by the server during a TLS V1.3 handshake. During a TLS V1.3 handshake, the server uses the client's preferred key share group order and selects a group that is in common with this list. The client and server use the selected group to encrypt and decrypt TLS V1.3 handshake messages. See Table 27 on page 799 for a list of valid 4-character key share specifications.

GSK_SERVER_TLS_KEY_SHARES may be specified for an SSL environment or SSL connection.

GSK_SID_VALUE

Specify the GSK_SID_VALUE to uniquely identify the cached session or session ticket to be used to resume an existing session. For SSL V3 through TLS V1.2 connections, the GSK_SID_VALUE may be specified only for an SSL connection and is only applicable for a server connection when V3 session caching is enabled (session cache and timeout are non-zero). For TLS V1.3 connections, the server must have issued a session ticket with the specified ID value (GSK_SESSION_TICKET_SERVER_ENABLE is ON and GSK_SESSION_TICKET_SERVER_COUNT is non-zero, or the **gsk_secure_socket_misc()** routine has been called with attribute ID GSK_SEND_SESSION_TICKET). Use **gsk_attribute_get_buffer()** to initially retrieve the GSK_SID_VALUE to be used by **gsk_attribute_set_buffer()**.

For more information about using the GSK_SID_VALUE, see Specifying a cached session in the **gsk_secure_socket_init()** usage section.

GSK_TLS_CERT_SIG_ALG_PAIRS

Specifies the list of hash and signature algorithm pair specifications that are supported by the client or server as a string consisting of one or more 4-character values in order of preference for use in digital signatures of X.509 certificates. The certificate signature algorithm pair specifications are sent by either the client or server to the session partner to indicate which signature/hash algorithm combinations are supported for digital signatures in X.509 certificates. The GSK_TLS_CERT_SIG_ALG_PAIRS setting overrides the GSK_TLS_SIG_ALG_PAIRS setting when checking the digital signatures of the remote peer's X.509 certificates. The certificate signature algorithm pair specification only has relevance for TLS V1.2 client or TLS V1.3 client and server sessions. See Table 30 on page 804 for a list of valid 4-character certificate signature algorithm pair specifications.

GSK_CERT_TLS_SIG_ALG_PAIRS may be specified for an SSL environment or SSL connection.

GSK_TLS_SIG_ALG_PAIRS
Specifies the list of hash and signature algorithm pair specifications that are supported by the client or server as a string consisting of 1 or more 4-character values in order of preference for use in digital signatures of X.509 certificates and TLS handshake messages. The signature algorithm pair specifications are sent by either the client or server to the session partner to indicate which signature/hash algorithm combinations are supported for digital signatures in X.509 certificates and TLS handshake messages. If the GSK_TLS_CERT_SIG_ALG_PAIRS setting is specified, the GSK_TLS_SIG_ALG_PAIRS setting is only used to indicate the signature/hash algorithm combinations supported for digital signatures in TLS handshake messages. The signature algorithm pair specification only has relevance for sessions using TLS V1.2 or higher protocols. See Table 30 on page 804 for a list of valid 4-character signature algorithm pair specifications.

GSK_TLS_SIG_ALG_PAIRS may be specified for an SSL environment or an SSL connection.

GSK_USER_DATA
Specifies the user data to be passed to SSL exit routines. The user data is copied to storage owned by the SSL run time and the address of this storage is passed to the SSL exit routines. The application may alter this copy of the user data but may not free it. GSK_USER_DATA may be specified only for an SSL connection.

GSK_V2_CIPHER_SPECS
Specifies the SSL V2 cipher specifications as a string consisting of 1 or more 1-character values. GSK_V2_CIPHER_SPECS may be specified for an SSL environment or an SSL connection. See Table 25 on page 795 for a list of valid SSL v2 cipher specifications.

Note: If Suite B support is enabled in the SSL environment, the SSL V2 cipher specifications are ignored.

GSK_V3_CIPHER_SPECS
Specifies the SSL V3 and TLS cipher specifications as a string consisting of 1 or more 2-character values. GSK_V3_CIPHER_SPECS may be specified for an SSL environment or an SSL connection. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 protocols. See Table 26 on page 795 for a list of valid 2-character cipher specifications.

Note: If Suite B support is enabled in the SSL environment, the SSL V3 2-character cipher specifications are ignored.

GSK_V3_CIPHER_SPECS_EXPANDED
Specifies the SSL V3 and TLS cipher specifications as a string consisting of 1 or more 4-character values. GSK_V3_CIPHER_SPECS_EXPANDED may be specified for an SSL environment or an SSL connection. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 protocols. See Table 27 on page 799 for a list of valid 4-character cipher specifications.

Applications wanting to use cipher suites that use Elliptic Curve Cryptography must set an appropriate cipher specification in GSK_V3_CIPHER_SPECS_EXPANDED.

Note: If Suite B support is enabled in the SSL environment, the SSL V3 4-character cipher specifications are ignored.

Related topics

- "gsk_attribute_get_buffer()" on page 76
- "gsk_environment_open()" on page 136
- "gsk_environment_init()" on page 134

- “gsk_secure_socket_open()” on page 173
- “gsk_secure_socket_init()” on page 157

gsk_attribute_set_callback()

Sets an SSL callback.

Format

```
#include <gskssl.h>

gsk_status gsk_attribute_set_callback (
                                        gsk_handle          ssl_handle,
                                        GSK_CALLBACK_ID     callback_id,
                                        void *              callback)
```

Parameters

ssl_handle
Specifies an SSL environment handle returned by **gsk_environment_open()** or an SSL connection handle returned by **gsk_secure_socket_open().**

callback_id
Specifies the callback identifier.

callback
Specifies the address of the callback parameter.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it is one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ATTRIBUTE_INVALID_ID]
The callback identifier is not valid or cannot be used with the specified handle.

[GSK_ATTRIBUTE_INVALID_PARAMETER]
The attribute parameter value is not valid.

[GSK_INVALID_HANDLE]
The handle is not valid.

[GSK_INVALID_STATE]
The environment or connection is not in the open state.

Usage

The **gsk_attribute_set_callback()** routine establishes a callback to an application routine by the SSL run time. A callback allows the application to replace the default routine used by the SSL run time. The SSL environment or SSL connection must be in the open state and not in the initialized state (that is, **gsk_environment_init()** or **gsk_secure_socket_init()** has not been called). The callback routine must use standard C linkage and not C++ linkage.

These callback identifiers are supported:

GSK_CERT_DIAGNOSTIC_CALLBACK
Indicates that the application is providing a routine to be used to collect certificate validation information for the peer's certificate and certificate chain. The callback parameter is the address of this routine.

This certificate validation information will be provided through the gsk_diag_summary structure. This structure contains the following data: the SSL and CMS return codes resulting from the validation process, the index of the failing certificate (which will be 0, if the validation was successful), the list of certificate sources available during validation, and a brief description of the error if one occurred.

The summary structure also has an array of gsk_cert_diag_detail structures, which contain information about each individual certificate of the certificate chain used to validate the subject. This data includes the certificate's Subject DN, the Issuer DN, the serial number, and the location (certificate's source) that the certificate was found. **gsk_name_to_dn()** is used to format the DN values.

Additionally, the diagnostic callback routine contains a diagnostic string, which is a consolidated subset form of the diagnostic information. This diagnostic string can be used as a means of viewing the most pertinent diagnostic information about the certificate validation result.

Once control is returned from the diagnostic callback routine, the diagnostic information will be cleared and freed. The callback routine is responsible for processing the information prior to returning.

By default, the diagnostic callback routine is called only when certificate validation fails. To have the diagnostic callback routine called for successful certificate validations or both successful and failure certificate validations, GSK_CERT_DIAG_INFO must be set. For information, see description of GSK_CERT_DIAG_INFO in "gsk_attribute_set_enum()" on page 112 and Appendix A, "Environment variables," on page 763.

For more information about the certificate diagnostic callback, see "Certificate diagnostic callback routine" on page 41.

The following is the prototype for the gsk_cert_diagnostic_callback function. It shows the parameters that were passed to the application:

```
void cert_diagnostic_callback (
                                gsk_handle                      soc_handle,
                                gsk_diag_summary *              diag_summary,
                                char *                          diag_string,
                                char *                          user_data);
```

GSK_CLIENT_CERT_CALLBACK

Indicates that the application is providing a routine to be used during a full handshake to prompt a client user to select a certificate from a list during the client authentication process. The *callback* parameter is the address of this routine. The exit routine can obtain the user data address by calling the **gsk_attribute_get_buffer()** routine. The **gsk_attribute_set_buffer()** routine should be called to set the selected key label before returning from the callback routine. The function return value should be 0 if a key label has been set or GSK_ERR_NO_CERTIFICATE if no client certificate is to be used. GSK_CLIENT_CERT_CALLBACK can be specified only for an SSL environment.

This is the prototype for the callback routine provided by the application. It shows the parameters passed to the application callback and the value returned by the callback.

```
int client_cert_callback (
                          gsk_handle     soc_handle)
```

GSK_IO_CALLBACK

Indicates that the application is providing the routines to perform read, write, and control functions. The *callback* parameter is the address of a *gsk_iocallback* structure. Each entry in the structure overrides the corresponding SSL runtime routine. A NULL entry will cause the current callback routine to be used or the SSL runtime routine will be used if there is no callback routine. GSK_IO_CALLBACK can be specified for an SSL environment or an SSL connection.

The routine specified by the *io_read* entry is used to read data from the network. The *fd* parameter is the socket descriptor, the *buffer* parameter is the address of the data buffer, the *count* parameter is the buffer size, and the *user_data* parameter is the user data address. The function return value should be 0 if the connection has been closed by the remote partner, -1 if an error is detected, or the number of bytes read from the network. The error code is returned in the *errno* runtime variable. The default routine uses the **recv()** library routine to read data from the network.

```
int io_read (
              int        fd,
              void *     buffer,
              int        count,
              char *     user_data)
```

The routine specified by the *io_write* entry is used to write data to the network. The *fd* parameter is the socket descriptor, the *buffer* parameter is the address of the data buffer, the *count* parameter is the data length, and the *user_data* parameter is the user data address. The function return value should be -1 if an error is detected or the number of bytes written to the network. The error code is returned in the *errno* runtime variable. The default routine uses the **send()** library routine to write data to the network.

```
int io_write (
               int        fd,
               void *     buffer,
               int        count,
               char *     user_data)
```

The routine specified by the *io_getpeerid* entry is used to get the 32-bit network identifier for the remote partner. The *fd* parameter is the socket descriptor and the *user_data* parameter is the user data address. However, the *io_getpeerid* entry is deprecated and should not be used since it does not support IPv6 networks which use a 16-byte network identifier. Instead, the *io_getpeername* entry should be used for both IPv4 and IPv6 networks. The *io_getpeerid* entry will not be used if the *io_getpeername* entry is not NULL.

```
unsigned long io_getpeerid (
                             int        fd,
                             char *     user_data)
```

The routine specified by the *io_setsocketoptions* entry is used to set socket options. The *fd* parameter is the socket descriptor, the **cmd** parameter is the function to be performed (GSK_SET_SOCKET_STATE_FOR_HANDSHAKE, GSK_SET_SOCKET_STATE_FOR_READ_WRITE), and the *user_data* parameter is the user data address. The return value should be -1 if an error is detected and 0 otherwise. The error code is returned in the *errno* runtime variable. The **io_setsocketoptions()** routine is called by the **gsk_secure_socket_init()** routine before initiating the SSL handshake (GSK_SET_SOCKET_STATE_FOR_HANDSHAKE) and again upon completion of the SSL handshake (GSK_SET_SOCKET_STATE_FOR_READ_WRITE). The default **io_setsocketoptions()** routine puts the socket into blocking mode for GSK_SET_SOCKET_STATE_FOR_HANDSHAKE and restores the original mode for GSK_SET_SOCKET_STATE_FOR_READ_WRITE.

```
int io_setsocketoptions (
                          int        fd,
                          int        cmd,
                          char *     user_data)
```

When the application is using non-blocking sockets and wants the SSL handshake processing to continue to use non-blocking sockets, an **io_setsocketoptions()** callback routine should be provided that accepts the *fd* and *cmd* values and returns with a return value of 0. No changes should be done to the state of the socket.

The routine specified by the *io_getpeername* entry is used to get the network identifier for the remote partner. The *fd* parameter is the socket descriptor, the *buffer* parameter is the address of the return buffer, the length parameter is the size of the return buffer, and the *user_data* parameter is the user data address. Upon return, the *length* parameter should contain the actual length of the network identifier. The function return value should be -1 if an error is detected and 0 otherwise. The error code is returned in the *errno* runtime variable. The default routine uses the **getpeername()** library routine and returns the IP address of the remote partner (4 bytes for IPv4 and 16 bytes for IPv6) followed by the 2-byte port number.

```
int io_getpeername (
                  int        fd,
                  void *     buffer,
                  int *      length,
                  char *     user_data)
```

GSK_SESSION_RESET_CALLBACK

Indicates that the application is providing the routines to be called when a session renegotiation has been initiated or completed to establish a new session key or have the session cipher reset. The callback parameter is the address of a gsk_reset_callback structure.

GSK_SESSION_RESET_CALLBACK can be specified for an SSL environment or an SSL connection. The callback is only invoked when using SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 protocols.

The routine specified by the *Reset_Init* entry is called when a session renegotiation has been initiated, and the SSL client has commenced the renegotiation process. The *con_handle* parameter is the handle for the SSL connection.

```
void (Reset_Init)  (
              gsk_handle        con_handle)
```

The *Reset_Complete* routine is called when a session renegotiation has been completed. If session renegotiation does not successfully complete, for example because of renegotiation not being allowed, then the *Reset_Complete* routine is not invoked even though the *Reset_Init* routine was called at the commencement of renegotiation. The *con_handle* parameter is the handle for the SSL connection.

```
void (Reset_Complete)  (
               gsk_handle        con_handle)
```

GSK_SID_CACHE_CALLBACK

Indicates that the application is providing the routines to maintain the session identifier cache. The callback parameter is the address of a gsk_sidcache_callback structure. GSK_SID_CACHE_CALLBACK can be specified only for an SSL environment and will be used only for SSL servers (the internal cache is always used for SSL clients).

The routine specified by the *Get* entry is called to retrieve an entry from the session identifier cache. The *session_id* parameter is the session identifier, the *session_id_length* parameter is the length of the session identifier, and the *ssl_version* parameter is the SSL protocol version number (GSK_SSLVERSION_V2 or GSK_SSLVERSION_V3). The function return value is the address of the session data buffer or NULL if an error is detected. The *FreeDataBuffer* routine will be called to release the session data buffer when it is no longer needed by the SSL run time.

```
gsk_data_buffer * Get (
                        const unsigned char *     session_id,
                        unsigned int              session_id_length,
                        gsk_sslversion            ssl_version)
```

The routine specified by the *Put* entry is called to store an entry in the session identifier cache. The *ssl_session_data* parameter is the session data, the *session_id* parameter is the session identifier, the *session_id_length* parameter is the length of the session identifier, and the *ssl_version* parameter is the SSL protocol version number (GSK_SSLVERSION_V2 or GSK_SSLVERSION_V3). The function return value is ignored and can be a NULL address. The callback routine must make its own copy of the session data since the SSL structure will be released when the connection is closed.

```
gsk_data_buffer * Put (
                        gsk_data_buffer *         ssl_session_data,
                        const unsigned char *     session_id,
                        unsigned int              session_id_length,
                        gsk_sslversion            ssl_version)
```

The routine specified by the *Delete* entry is called to remove an entry from the session identifier cache. The *session_id* parameter is the session identifier, the *session_id_length* parameter is the length of the session identifier, and the *ssl_version* parameter is the SSL protocol version number (GSK_SSLVERSION_V2 or GSK_SSLVERSION_V3).

```
void Delete (
              const unsigned char *     session_id,
              unsigned int              session_id_length,
              gsk_sslversion            ssl_version)
```

The routine specified by the *FreeDataBuffer* entry is called to release the data buffer returned by the *Get* routine.

```
void FreeDataBuffer (
                      gsk_data_buffer *     ssl_session_data)
```

GSK_SNI_CALLBACK

Indicates that the application is providing the routine to allow a server to interrogate a list of server names supplied by the client and select an appropriate key label for use as the server certificate based on the information received from the client. The selected certificate from the key database, PKCS #12 file, key ring or token will be sent to the client as the server certificate during the handshake process. The callback parameter is the address of this routine. The exit routine can obtain the server name list provided by the client by calling the **gsk_attribute_get_buffer()** routine. The **gsk_attribute_set_buffer()** routine should be called to set the selected key label before returning from the callback routine.

The callback routine cannot enforce the required use of the server name indication extension. The failure to select a key label causes a fatal UNRECOGNIZED_NAME alert. To enforce such actions with the callback routine the user must set the GSK_TLS_EXTID_SNI_SERVER_LABELS extension by calling the **gsk_attribute_set_tls_extension()** routine. The required and unrecognized_name_fatal fields of the extension must be set appropriately to achieve the requested outcome, although the serverKeyLabel list may be empty.

The function return value should be 0 if a key label has been set or GSK_ERR_UNRECOGNIZED_NAME if no server certificate is selected. Enforcement of the required and unrecognized_name_fatal settings occur on return from the callback routine. GSK_SNI_CALLBACK can be specified only for an SSL environment.

This is the prototype for the callback routine provided by the application. It shows the parameters passed to the application callback and the value returned by the callback.

```
 int sni_callback (
                      gsk_handle     soc_handle)
```

Related topics

- "gsk_environment_init()" on page 134
- "gsk_secure_socket_init()" on page 157

gsk_attribute_set_enum()

Sets an enumerated value.

Format

```
#include <gskssl.h>

gsk_status gsk_attribute_set_enum (
                                   gsk_handle          ssl_handle,
                                   GSK_ENUM_ID         enum_id,
                                   GSK_ENUM_VALUE      enum_value)
```

Parameters

ssl_handle
Specifies an SSL environment handle that is returned by **gsk_environment_open()** or an SSL connection handle that is returned by **gsk_secure_socket_open()**.

enum_id
Specifies the enumeration identifier.

enum_value
Specifies the enumeration value.

Results

The function return value is 0 (**GSK_OK**) if no error is detected. Otherwise, it is one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ATTRIBUTE_INVALID_ENUMERATION]
The value of an attribute is not valid.

[GSK_ATTRIBUTE_INVALID_ID]
The enumeration identifier is not valid or cannot be used with the specified handle.

[GSK_ERR_PROTOCOL_NOT_SUPPORTED_WITH_FIPS]
Protocol is not supported in FIPS mode.

[GSK_INVALID_HANDLE]
The handle is not valid.

[GSK_INVALID_STATE]
The environment or connection is not in the open state.

Usage

The **gsk_attribute_set_enum()** routine sets an enumerated value for an SSL environment or an SSL connection. The environment or connection must be in the open state and not in the initialized state (that is, **gsk_environment_init()** or **gsk_secure_socket_init()** has not been called).

The values set that uses this service are treated as independent values. They are not validated with other values set that uses **gsk_attribute_set_buffer()**, **gsk_attribute_set_enum()**, or **gsk_attribute_set_tls_extensions()** APIs until used together to perform an SSL/TLS handshake by calling **gsk_secure_socket_init()**.

These enumeration identifiers are supported:

GSK_3DES_KEYCHECK
Specifies that when Triple DES session keys are generated, each key part must be unique.

GSK_3DES_KEYCHECK_ON
Key parts are compared for uniqueness.

GSK_3DES_KEYCHECK_OFF
When operating in non-FIPS mode, the key parts are not compared for uniqueness. Key uniqueness is always enforced in FIPS mode.

GSK_3DES_KEYCHECK can be specified only for an SSL environment.

GSK_AIA_CDP_PRIORITY
Specifies the priority order that the AIA and CDP extensions are checked for revocation information.

Specify GSK_AIA_CDP_PRIORITY_ON to indicate that the AIA extension is processed before the CDP extension during certificate revocation checking. This means that any OCSP responders specified in the AIA extension or the OCSP responder specified in GSK_OCSP_URL are contacted before attempting to contact the HTTP servers specified in the URI values in the CDP extension. This is the default setting.

Specify GSK_AIA_CDP_PRIORITY_OFF to indicate that the CDP extension is queried prior to the AIA extension. This means that HTTP servers in the URI values in the CDP extension are contacted before attempting to contact the OCSP responders in the AIA extension or the OCSP responder specified in GSK_OCSP_URL.

GSK_AIA_CDP_PRIORITY can be specified only for an SSL environment.

GSK_CERT_DIAG_INFO
Specifies that when validation is being performed against the peer's certificate and certificate chain, certificate validation information will be provided for failures, successes, or both through the GSK_CERT_DIAGNOSTIC_CALLBACK.

Specify GSK_CERT_DIAG_INFO_FAILURE_ONLY when certificate diagnostic information is only provided for validation failures. This is the default setting.

Specify GSK_CERT_DIAG_INFO_SUCCESS_ONLY when certificate diagnostic information is only provided for successful validations.

Specify GSK_CERT_DIAG_INFO_SUCCESS_OR_FAILURE when certificate diagnostic information is to be provided for both success and failure validations.

GSK_CERT_DIAG_INFO can be specified for an SSL environment or an SSL connection, but is only applicable when a certificate diagnostic callback routine has been set through the **gsk_attribute_set_callback()** routine.

GSK_CERT_VALIDATE_KEYRING_ROOT
Specifies the setting of how certificates in a SAF key ring are validated. Specify GSK_CERT_VALIDATE_KEYRING_ROOT_ON if SAF key ring certificates must be validated to the root CA certificate. Specify GSK_CERT_VALIDATE_KEYRING_ROOT_OFF if SAF key ring certificates are only validated to the trust anchor certificate. If a sole intermediate certificate is found in a SAF key ring and the next issuer is not found in the same SAF key ring, the intermediate certificate acts as a trust anchor and the certificate chain is considered complete. By default, SAF key ring certificates are only validated to the trust anchor certificate. This setting does not affect the validation of SSL key database file, PKCS #12 file, and PKCS #11 token certificates as these certificates are always validated to the root CA certificate.

GSK_CERT_VALIDATE_KEYRING_ROOT can be specified only for an SSL environment.

GSK_CERT_VALIDATION_MODE
Specifies the method of certificate validation. RFC 2459, RFC 3280, and RFC 5280 describe differing methods of certificate validation. Specify GSK_CERT_VALIDATION_MODE_2459 if certificate validation according to the RFC 2459 method is required, GSK_CERT_VALIDATION_MODE_3280 if certificate validation according to the RFC 3280 method is required, or GSK_CERT_VALIDATION_MODE_5280 if certificate validation according to the RFC 5280 method is required.

Specify GSK_CERT_VALIDATION_MODE_ANY if certificate validation can use any supported X.509 certificate validation method.

GSK_CERT_VALIDATION_MODE can be specified only for an SSL environment.

GSK_CLIENT_AUTH_ALERT

Specify GSK_CLIENT_AUTH_NOCERT_ALERT_OFF if the SSL server application is to allow client connections where client authentication is requested and the client fails to supply an X.509 certificate. Specify GSK_CLIENT_AUTH_NOCERT_ALERT_ON if the SSL server application is to terminate client connections where client authentication is requested and the client fails to supply an X.509 certificate.

GSK_CLIENT_AUTH_ALERT can be specified only for an SSL environment and is only applicable for server sessions with client authentication active.

GSK_CLIENT_AUTH_TYPE

Specifies GSK_CLIENT_AUTH_FULL_TYPE to validate client certificates. If a certificate is not valid, the connection is not started and an error code is returned by the **gsk_secure_socket_init()** routine. If an LDAP server is specified, the LDAP server is queried for CA certificates and certificate revocation lists. If the LDAP server is not available, only local validation is performed. If no client certificate is received and either GSK_CLIENT_AUTH_ALERT is not specified or is set to GSK_CLIENT_AUTH_NOCERT_ALERT_OFF, the connection is successful. The application can check for this case by calling the **gsk_attribute_get_cert_info()** routine and checking for a NULL return address.

When a client's certificate is being requested, the client can be required to provide a certificate by setting GSK_CLIENT_AUTH_ALERT to GSK_CLIENT_NOCERT_ALERT_ON. If no certificate is received, the requested handshake fails. For more information about the GSK_CLIENT_AUTH_ALERT setting, see "gsk_attribute_set_enum()" on page 112.

Specify GSK_CLIENT_AUTH_PASSTHRU_TYPE to bypass client certificate validation. The application can retrieve the certificate by calling the **gsk_attribute_get_cert_info()** routine.

GSK_CLIENT_AUTH_TYPE can be specified only for an SSL environment and is only applicable for server sessions with client authentication active.

GSK_CLIENT_EPHEMERAL_DH_GROUP_SIZE

Specifies the minimum accepted server Diffie-Hellman group size that is allowed for an ephemeral Diffie-Hellman key exchange message. GSK_CLIENT_EPHEMERAL_DH_GROUP_SIZE can be specified only at the environment level.

Specify GSK_CLIENT_EPHEMERAL_DH_GROUP_SIZE_LEGACY for the client application to enforce a minimum group size of 1024 for each server new handshake in non-FIPS mode and group size 2048 when operating in FIPS mode.

Specify GSK_CLIENT_EPHEMERAL_DH_GROUP_SIZE_2048 for the client server application to enforce a minimum group size of 2048 for each new server handshake.

This setting is only meaningful if specified for a client connection within an SSL environment.

GSK_CLIENT_EXTENDED_MASTER_SECRET

Specifies if the TLS client sends the extended master secret extension to the server. This option is only applicable for TLS V1.0, TLS V1.1, and TLS V1.2 handshakes.

Specify GSK_CLIENT_EXTENDED_MASTER_SECRET_OFF if the TLS client does not send the extended master secret extension to the server.

Specify GSK_CLIENT_EXTENDED_MASTER_SECRET_ON if the TLS client sends the extended master secret extension to the server, but does not require the server to support the extension. This is the default.

Specify GSK_CLIENT_EXTENDED_MASTER_SECRET_REQUIRED if the TLS client sends the extended master secret extension to the server and requires the server to support the extension. If a server does not send the extended master secret extension, the handshake fails. Before setting this option to REQUIRED, ensure that the server being communicated with supports the extended master secret extension. If the remote server partner is a z/OS System SSL application, it must be running z/OS V2R3 or later and have PTFs for APAR OA60105 (z/OS V2R3 and V2R4) applied and active before setting this option to REQUIRED.

GSK_CLIENT_EXTENDED_MASTER_SECRET can be specified for an SSL environment or an SSL connection.

GSK_CRL_CACHE_EXTENDED
Specifies that LDAP extended CRL cache support is enabled.

Specify GSK_CRL_CACHE_EXTENDED_ON to indicate that LDAP extended CRL cache support is enabled. LDAP extended CRL cache support enables the following support:

- LDAP CRLs are only cached when there is an expiration time present and it is greater than the current time.
- A limit on the maximum number of CRLs that can be stored in the LDAP cache, which can be overridden by specifying GSK_CRL_CACHE_SIZE. The default is 32.
- By default, GSK_CRL_CACHE_TEMP_CRL is disabled.

Specify GSK_CRL_CACHE_EXTENDED_OFF to indicate that LDAP basic CRL cache support is enabled. When LDAP basic CRL cache support is enabled, retrieved LDAP CRLs are only cached if GSK_CRL_CACHE_TIMEOUT is greater than 0 and GSK_CRL_CACHE_SIZE is set to a non-zero number. LDAP basic CRL cache support is the default.

Note: When set to GSK_CRL_CACHE_EXTENDED_ON, the GSK_CRL_CACHE_TIMEOUT value is ignored.

GSK_CRL_CACHE_EXTENDED can be specified only for an SSL environment.

GSK_CRL_CACHE_TEMP_CRL
Specifies if a temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not reside on the LDAP server.

Specify GSK_CRL_CACHE_TEMP_CRL_ON if a temporary CRL cache entry is to be added to the LDAP CRL cache.

Specify GSK_CRL_CACHE_TEMP_CRL_OFF if a temporary CRL cache entry is not to be added to the LDAP CRL cache.

If a temporary CRL is cached, it prevents continual attempts to contact the LDAP server and allows connections to be successful.

GSK_CRL_CACHE_TEMP_CRL can be specified only for an SSL environment.

GSK_CRL_SECURITY_LEVEL
Specifies the level of security to be used when contacting LDAP servers to check CRLs for revoked certificates during certificate validation.

Three levels of security are available:

GSK_CRL_SECURITY_LEVEL_LOW
Certificate validation does not fail if the LDAP server cannot be contacted.

GSK_CRL_SECURITY_LEVEL_MEDIUM
Certificate validation requires the LDAP server to be contactable, but does not require a CRL to be retrieved. This is the default.

GSK_CRL_SECURITY_LEVEL_HIGH
Certificate validation requires revocation information to be provided by the LDAP server.

GSK_CRL_SECURITY_LEVEL can be specified only for an SSL environment.

GSK_ENABLE_CLIENT_SET_PEERID
Specify GSK_ENABLE_CLIENT_SET_PEERID_ON to enable the use of a cached GSK_PEER_ID for SSL V3, TLS V1.0, or higher client connections. GSK_ENABLE_CLIENT_SET_PEERID can be specified only for an SSL environment and is only applicable for client connections. GSK_ENABLE_CLIENT_SET_PEERID_OFF is the default setting.

GSK_ENABLE_CLIENT_SET_PEERID_ON limits the number of full client handshakes that can be cached over the lifetime of the SSL environment to a maximum number of 4.29 billion. If this maximum number is reached, all new SSL/TLS connections that are not using a cached GSK_PEER_ID results in full handshakes and will not add entries into the session cache. Also, if the maximum number is reached, reusing a cached GSK_PEER_ID is allowed if the GSK_PEER_ID can still be located in the cache.

GSK_EXTENDED_RENEGOTIATION_INDICATOR

Specify GSK_EXTENDED_RENEGOTIATION_INDICATOR_OPTIONAL to not require the renegotiation indicator during initial SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 handshake. This is the default.

Specify GSK_EXTENDED_RENEGOTIATION_INDICATOR_CLIENT to allow the client initial handshake to proceed only if the server indicates support for RFC 5746 Renegotiation.

Specify GSK_EXTENDED_RENEGOTIATION_INDICATOR_SERVER to allow the server initial handshake to proceed only if the client indicates support for RFC 5746 Renegotiation.

Specify GSK_EXTENDED_RENEGOTIATION_INDICATOR_BOTH to allow the server and client initial handshakes to proceed only if the partner indicates support for RFC 5746 Renegotiation.

GSK_EXTENDED_RENEGOTIATION_INDICATOR can be specified only for an SSL environment.

GSK_HTTP_CDP_ENABLE

Specifies whether the HTTP URIs within the CDP extension are to be used for certificate revocation checking.

Specify GSK_HTTP_CDP_ENABLE_OFF to indicate that certificate revocation checking with the HTTP URI values in the CDP is not enabled. This is the default.

Specify GSK_HTTP_CDP_ENABLE_ON to indicate that certificate revocation checking with the HTTP URI values in the CDP extension is enabled.

GSK_HTTP_CDP_ENABLE can be specified only for an SSL environment.

GSK_MIDDLEBOX_COMPAT_MODE

Specifies if the TLS V1.3 handshake process ought to use or tolerate handshake messages in a manner compliant with earlier TLS protocols to alleviate possible issues with middleboxes or proxies.

Specify GSK_MIDDLEBOX_COMPAT_MODE_ON if the TLS V1.3 handshake process ought to use or tolerate handshake messages in a manner compliant with earlier TLS protocols to alleviate possible issues with middleboxes or proxies.

Specify GSK_MIDDLEBOX_COMPAT_MODE_OFF if the TLS V1.3 handshake process ought to use the pure TLS V1.3 handshake message format.

GSK_MIDDLEBOX_COMPAT_MODE can be specified only for an SSL environment.

GSK_OCSP_ENABLE

Specifies whether the AIA extensions are to be used for certificate revocation checking.

Specify GSK_OCSP_ENABLE_ON to activate certificate revocation checking using the HTTP URI values in the certificate's AIA extension.

Specify GSK_OCSP_ENABLE_OFF to disable use of the AIA extension. This is the default.

If GSK_OCSP_URL is specified, GSK_OCSP_ENABLE is set to ON, and GSK_OCSP_URL_PRIORITY is set to ON, then the order the responders are used is GSK_OCSP_URL defined responder first and then the responders identified in the AIA extension. If GSK_OCSP_URL is specified, GSK_OCSP_ENABLE is set to ON and GSK_OCSP_URL_PRIORITY is set to OFF, then the order that responders are used is the responders identified in the AIA extension first and then the GSK_OCSP_URL defined responder.

GSK_OCSP_ENABLE can be specified only for an SSL environment.

GSK_OCSP_NONCE_CHECK_ENABLE

Specifies if OCSP response nonce checking is on or off.

Specify GSK_OCSP_NONCE_CHECK_ENABLE_ON to have the nonce in the OCSP response verified to ensure it matches the nonce sent in the OCSP request.

Note: Setting GSK_OCSP_NONCE_CHECK_ENABLE_ON also implies that GSK_OCSP_NONCE_GENERATION_ENABLE_ON is also set to ON.

Specify GSK_OCSP_NONCE_CHECK_ENABLE_OFF to disable checking of the nonce in the OCSP response. This is the default.

GSK_OCSP_NONCE_CHECK_ENABLE can be specified only for an SSL environment.

GSK_OCSP_NONCE_GENERATION_ENABLE
Specifies whether the OCSP request includes a generated nonce.

Specify GSK_OCSP_NONCE_GENERATION_ENABLE_ON to enable nonce generation.

Specify GSK_OCSP_NONCE_GENERATION_ENABLE_OFF to disable OCSP nonce generation. This is the default.

GSK_OCSP_NONCE_GENERATION_ENABLE can be specified only for an SSL environment.

GSK_OCSP_RETRIEVE_VIA_GET
Specifies whether the HTTP request to the OCSP responder is sent using the HTTP Get Method or the HTTP Post method.

Specify GSK_OCSP_RETRIEVE_VIA_GET_ON to indicate that the HTTP GET method should be used when sending an OCSP request whose total request size after Base64 encoding is less than 255 bytes. This option allows HTTP caching on the OCSP responder when the responder has been enabled for caching.

Specify GSK_OCSP_RETRIEVE_VIA_GET_OFF to indicate the HTTP request should always be sent via an HTTP Post method. This is the default.

GSK_OCSP_RETRIEVE_VIA_GET can be specified only for an SSL environment.

GSK_OCSP_URL_PRIORITY
Specifies the order of precedence for contacting the OCSP responder locations if both GSK_OCSP_URL and GSK_OCSP_ENABLE are active.

Specify GSK_OCSP_URL_PRIORITY_ON to indicate that the GSK_OCSP_URL defined responder is used first and then the responders identified in the AIA extension. This is the default.

Specify GSK_OCSP_URL_PRIORITY_OFF to indicate that the responder identified in the AIA extension is used first and then the GSK_OCSP_URL defined responder.

GSK_OCSP_URL_PRIORITY can be specified only for an SSL environment.

GSK_PEER_CERT_MIN_VERSION
Specifies that certificate validation should ensure that the partner's end-entity certificate is a minimum X.509 version. This setting is ignored if the selected protocol is TLS V1.3 as it requires a X.509 version 3 certificate.

Specify GSK_PEER_CERT_VERSION_3 so that the partner's end-entity certificate must be an X.509 version 3.

Specify GSK_PEER_CERT_VERSION_ANY so that the partner's end-entity certificate can be any supported System SSL X.509 version.

GSK_PEER_CERT_MIN_VERSION can only be specified at the SSL environment level.

GSK_PROTOCOL_SSLV2
Specifies GSK_PROTOCOL_SSLV2_ON to enable the SSL Version 2 protocol or GSK_PROTOCOL_SSLV2_OFF to disable the SSL Version 2 protocol. The SSL V2 protocol should be disabled whenever possible since the SSL V3 and TLS protocols provide significant security enhancements.

GSK_PROTOCOL_SSLV2 can be specified for an SSL environment or an SSL connection.

When operating in FIPS mode, the SSL Version 2 protocol is not used. Enabling this protocol has no effect.

When TLS extensions are defined for the client and any of the TLS protocols are enabled for the connection, the SSL Version 2 protocol is not used. Enabling this protocol has no effect.

GSK_PROTOCOL_SSLV3
Specifies GSK_PROTOCOL_SSLV3_ON to enable the SSL Version 3 protocol or GSK_PROTOCOL_SSLV3_OFF to disable the SSL Version 3 protocol.

GSK_PROTOCOL_SSLV3 can be specified for an SSL environment or an SSL connection.

When operating in FIPS mode, the SSL Version 3 protocol is not used. Enabling this protocol has no effect.

GSK_PROTOCOL_TLSV1
Specifies GSK_PROTOCOL_TLSV1_ON to enable the TLS Version 1.0 protocol or GSK_PROTOCOL_TLSV1_OFF to disable the TLS Version 1.0 protocol.

GSK_PROTOCOL_TLSV1 can be specified for an SSL environment or an SSL connection.

GSK_PROTOCOL_TLSV1_1
Specifies GSK_PROTOCOL_TLSV1_1_ON to enable the TLS Version 1.1 protocol or GSK_PROTOCOL_TLSV1_1_OFF to disable the TLS Version 1.1 protocol.

GSK_PROTOCOL_TLSV1_1 can be specified for an SSL environment or an SSL connection.

GSK_PROTOCOL_TLSV1_2
Specify GSK_PROTOCOL_TLSV1_2_ON to enable the TLS Version 1.2 protocol or GSK_PROTOCOL_TLSV1_2_OFF to disable the TLS Version 1.2 protocol.

GSK_PROTOCOL_TLSV1_2 can be specified for an SSL environment or an SSL connection.

GSK_PROTOCOL_TLSV1_3
Specify GSK_PROTOCOL_TLSV1_3_ON to enable the TLS Version 1.3 protocol or GSK_PROTOCOL_TLSV1_3_OFF to disable the TLS Version 1.3 protocol.

Note: The TLS V1.3 protocol is not currently supported in FIPS mode. If an attempt is made to enable this protocol while running in FIPS mode, an error is returned from this routine.

GSK_PROTOCOL_TLSV1_3 can be specified for an SSL environment or an SSL connection.

GSK_RENEGOTIATION
Specify GSK_RENEGOTIATION_NONE to disable SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 handshake renegotiation as a server and allow RFC 5746 renegotiation. This is the default.

Specify GSK_RENEGOTIATION_DISABLED to disable SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 handshake renegotiation as a server and also disable RFC 5746 renegotiation.

Specify GSK_RENEGOTIATION_ALL to allow SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 handshake renegotiation as a server while also allowing RFC 5746 renegotiation.

Specify GSK_RENEGOTIATION_ABBREVIATED to allow SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 abbreviated handshake renegotiation as a server for resuming the current session only, while disabling SSL V3 and TLS full handshake renegotiation as a server. With this enumeration value set, the System SSL session ID cache is not checked when resuming the current session. RFC 5746 renegotiation is allowed.

GSK_RENEGOTIATION can be specified only for an SSL environment.

GSK_RENEGOTIATION_PEER_CERT_CHECK
Specify GSK_RENEGOTIATION_PEER_CERT_CHECK_OFF to not perform an identity check against the peer's certificate during renegotiation. This allows the peer certificate to change during renegotiation. This is the default.

Specify GSK_RENEGOTIATION_PEER_CERT_CHECK_ON to perform a comparison against the peer's certificate to ensure that certificate does not change during renegotiation.

GSK_RENEGOTIATION_PEER_CERT_CHECK can be specified only for an SSL environment.

GSK_REQ_CACHED_SESSION
Specify GSK_REQ_CACHED_SESSION_ON to require the cached session that is identified by GSK_PEER_ID to be used for an upcoming SSL V3, TLS V1.0, or higher connections. If either a cached or full handshake is allowed, specify GSK_REQ_CACHED_SESSION_OFF.

GSK_REQ_CACHED_SESSION_OFF is the default setting.

GSK_REQ_CACHED_SESSION can be specified only for an SSL environment and is only applicable for client connections.

GSK_REVOCATION_SECURITY_LEVEL

Specifies the level of security to be used when contacting an OCSP responder or an HTTP server specified in a URI value of the CDP extension.

Three levels of security are available:

- GSK_REVOCATION_SECURITY_LEVEL_LOW: Certificate validation does not fail if the OCSP responder or HTTP server specified in the URI value of the CDP extension cannot be contacted.
- GSK_REVOCATION_SECURITY_LEVEL_MEDIUM: Certificate validation requires the OCSP responder or the HTTP server in a URI value in the CDP extension to be contactable. For an OCSP responder, it must be able to provide a valid certificate revocation status. If the certificate status is revoked or unknown, certificate validation fails. For an HTTP server in a CDP extension, it must be contactable and able to provide a CRL. This is the default setting.
- GSK_REVOCATION_SECURITY_LEVEL_HIGH: Certificate validation requires revocation information to be provided by the OCSP responder or HTTP server. If OCSP revocation checking via the AIA extension is enabled, there must be HTTP URI values present in the certificate that are able to be contactable and able to provide a valid certificate revocation status. If HTTP CRL checking is enabled, there must be HTTP URI values in the CDP extension that are able to be contactable and able to provide a CRL.

GSK_REVOCATION_SECURITY_LEVEL can be specified only for an SSL environment.

GSK_SERVER_EPHEMERAL_DH_GROUP_SIZE

Specifies the minimum server Diffie-Hellman group size that is allowed for an ephemeral Diffie-Hellman key exchange message between client and server. GSK_SERVER_EPHEMERAL_DH_GROUP_SIZE can only be specified at the environment level.

Specify GSK_SERVER_EPHEMERAL_DH_GROUP_SIZE_LEGACY for the server application to use minimum group size of 1024 for each new handshake in non-FIPS mode and group size 2048 when operating in FIPS mode.

Specify GSK_SERVER_EPHEMERAL_DH_GROUP_SIZE_2048 for the server application to use minimum group size of 2048 for each new handshake.

Specify GSK_SERVER_EPHEMERAL_DH_GROUP_SIZE_MATCH for the server application to match the ephemeral Diffie-Hellman group to the server certificate's key strength. If the key size is less than or equal to 1024, a group size of 1024 is used. If the key size is greater than 1024, a group size of 2048 is used.

Note: With the Legacy and 2048 settings, a defined set of Diffie-Hellman parameters is used to generate a unique key for each connection. With the Match setting, the Diffie-Hellman parameters are generated and then used to generate a unique key. The additional processing to generate the Diffie-Hellman parameters is done in software and is CPU-intensive.

This setting is only meaningful if specified for a server connection within an SSL environment.

GSK_SERVER_EXTENDED_MASTER_SECRET

Specifies if the TLS server supports negotiating the extended master secret extension from clients. This option is only applicable for TLS V1.0, TLS V1.1, and TLS V1.2 handshakes.

Specify GSK_SERVER_EXTENDED_MASTER_SECRET_OFF if the TLS server does not support negotiating the extended master secret extension from clients.

Specify GSK_SERVER_EXTENDED_MASTER_SECRET_ON if the TLS server supports negotiating the extended master secret extension from clients. This is the default.

Specify GSK_SERVER_EXTENDED_MASTER_SECRET_REQUIRED if the TLS server requires negotiating the extended master secret extension from clients. If a client does not send the extended master secret extension, the handshake fails. Before setting this option to REQUIRED, ensure that all clients communicating with this server support the extended master secret extension. If the remote client partner is a z/OS System SSL application, it must be running z/OS V2R3 or later and have PTFs for APAR OA60105 (z/OS V2R3 and V2R4) applied and active before setting this option to REQUIRED. If the server is enabled for sysplex session ID caching (GSK_SYSPLEX_SIDCACHE is set to ON), all systems must be running z/OS V2R3 or later and any z/OS V2R3 or z/OS V2R4 systems must have the PTFs for APAR OA60105 applied and active before setting this option to REQUIRED for maximum compatibility.

GSK_SERVER_EXTENDED_MASTER_SECRET can be specified for an SSL environment or an SSL connection.

GSK_SERVER_FALLBACK_SCSV

Specifies whether the server accepts the TLS fallback Signaling Cipher Suite Value (SCSV) when the client's cipher list includes it during an SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 handshake. The SCSV indicates to the server that the client is attempting to fallback to an earlier TLS or SSL protocol version after a previous handshake attempt failed.

Specify GSK_SERVER_FALLBACK_SCSV_ON to indicate that the server supports the TLS fallback Signaling Cipher Suite Value (SCSV) when included in the client's supported cipher list during an SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 handshake. If the SCSV is present in the client's supported list and the TLS or SSL protocol level that is specified by the client during the handshake is less than the highest TLS or SSL protocol level that is supported by the server, the SSL or TLS handshake attempt fails.

Specify GSK_SERVER_FALLBACK_SCSV_OFF to indicate that the server ignores the SCSV when included in the client's supported cipher list during an SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 handshake. This is the default setting.

GSK_SERVER_FALLBACK_SCSV can be specified only for an SSL environment.

GSK_SERVER_OCSP_STAPLING

Specifies if the server supports the retrieval of the OCSP responses for the server's end entity certificate or the server's certificate chain if the client specifies support for the OCSP responses in the TLS handshake. The client indicates support for the retrieval of the OCSP responses by including the Certificate Status Request or the Multiple Certificate Status Request TLS extensions in a TLS handshake message. The OCSP responses are retrieved by the server and are sent to the client as part of the TLS handshake. The client can then parse the OCSP responses to determine the revocation status of the server's end entity certificate or the server's certificate chain. The inclusion of the OCSP responses in a TLS handshake message is commonly referred to as OCSP stapling.

Specify GSK_SERVER_OCSP_STAPLING_ENDENTITY to enable the server to contact the configured OCSP responders to retrieve the OCSP response for the server's end entity certificate.

Specify GSK_SERVER_OCSP_STAPLING_ANY to enable the server to contact the configured OCSP responders to retrieve the OCSP responses for the server's end entity certificate or the server's certificate chain. If the negotiated handshake protocol is TLS V1.2 and earlier, the OCSP responses that are retrieved by the server and sent to the client depend on the Certificate Status Request and the Multiple Certificate Status Request extensions being present in the TLS handshake message from the client. If both extensions are specified by the client in a TLS V1.2 and earlier handshake, the Multiple Certificate Status Request extension takes precedence. If the negotiated handshake protocol is TLS V1.3, the Multiple Certificate Status Request extension is not supported and the Certificate Status Request extension allows for the retrieval of the OCSP response for only the server’s end entity certificate.

Specify GSK_SERVER_OCSP_STAPLING_OFF if the server is not enabled to contact the configured OCSP responders to retrieve the OCSP responders for the server's end entity certificate or the server's certificate chain. GSK_SERVER_OCSP_STAPLING_OFF is the default setting.

The GSK_OCSP_URL or the GSK_OCSP_ENABLE settings must be specified before initializing the TLS environment. These settings are required to contact the desired OCSP responder to retrieve the OCSP responses for the server's end entity certificate or the server's certificate chain.

Notes:

- When OCSP stapling is enabled, extra processing time is required by the server to contact the OCSP responder to retrieve the OCSP response.
- For information about the OCSP-related options that are ignored or allowed when OCSP Stapling is enabled, see "Enabling OCSP server stapling" on page 57.

GSK_SERVER_OCSP_STAPLING can be specified only for an SSL environment.

GSK_SESSION_TICKET_CLIENT_ENABLE

Specifies if the client supports caching session tickets received from a server after a TLS V1.3 handshake has completed and supports TLS V1.3 resumption attempts to the server.

Specify GSK_SESSION_TICKET_CLIENT_ENABLE_ON if the client supports caching session tickets received from a server after a TLS V1.3 handshake has completed and supports TLS V1.3 resumption attempts to the server. The GSK_V3_SESSION_TIMEOUT and GSK_V3_SIDCACHE_SIZE settings also must be set to values greater than 0 to allow client session ticket caching. This is the default setting.

Specify GSK_SESSION_TICKET_CLIENT_ENABLE_OFF if the client does not support caching session tickets received from a server after a TLS V1.3 handshake has completed and does not support TLS V1.3 resumption attempts to the server.

GSK_SESSION_TICKET_CLIENT_ENABLE can be specified only for an SSL environment.

GSK_SESSION_TICKET_SERVER_ALGORITHM

Specifies the algorithm to be used by the server to encrypt and decrypt the session tickets used for TLS V1.3 session resumption. This algorithm will be used to generate a key for the environment.

Specify GSK_SESSION_TICKET_SERVER_ALGORITHM_AESCBC128 to use 128-bit AES-CBC encryption for session tickets. This is the default setting.

Specify GSK_SESSION_TICKET_SERVER_ALGORITHM_AESCBC256 to use 256-bit AES-CBC encryption for session tickets.

GSK_SESSION_TICKET_SERVER_ALGORITHM can be specified only for an SSL environment.

GSK_SESSION_TICKET_SERVER_ENABLE

Specifies if the server supports sending session tickets after a TLS V1.3 handshake has completed and if it will accept TLS V1.3 resumption attempts from the client.

Specify GSK_SESSION_TICKET_SERVER_ENABLE_ON if the server supports sending session tickets after a TLS V1.3 handshake has completed and if it will accept TLS V1.3 resumption attempts from the client. This is the default setting.

Specify GSK_SESSION_TICKET_SERVER_ENABLE_OFF if the server does not support sending session tickets after a TLS V1.3 handshake has completed and will not accept TLS V1.3 resumption attempts from the client.

GSK_SESSION_TICKET_SERVER_ENABLE can be specified only for an SSL environment.

GSK_SESSION_TYPE

Specifies GSK_CLIENT_SESSION to perform the SSL handshake as a client, GSK_SERVER_SESSION to perform the SSL handshake as a server, or GSK_SERVER_SESSION_WITH_CL_AUTH to perform the SSL handshake as a server requiring client authentication.

GSK_SESSION_TYPE can be specified for an SSL environment or an SSL connection.

GSK_SUITE_B_PROFILE

Specifies the Suite B profile that an SSL server or client applies to TLS sessions. RFCs 5430 and 6460 define the cipher suites that are valid for use when using the compliant Suite B profile for TLS. Specify:

- GSK_SUITE_B_PROFILE_128 if only the 128-bit Suite B security profile is required.
- GSK_SUITE_B_PROFILE_128MIN if a 128-bit minimum Suite B security profile is required.
- GSK_SUITE_B_PROFILE_192 if only the 192-bit Suite B security profile is required.
- GSK_SUITE_B_PROFILE_192MIN if a 192-bit minimum Suite B security profile is required.
- GSK_SUITE_B_PROFILE_ALL if both the 128-bit and 192-bit Suite B security profiles are required.

- GSK_SUITE_B_PROFILE_OFF if the Suite B security profile is not to be applied to any TLS sessions.

GSK_SUITE_B_PROFILE can be specified only for an SSL environment. Because this setting affects the cipher suites that are allowed, this also has an implicit effect on the Elliptic Curves and Certificates that can be used. Suite B Cryptography requires that key establishment and authentication algorithms that are used in TLS sessions be based on Elliptic Curve Cryptography, and that the encryption algorithm is AES. For more information about the cipher suites, elliptic curves, and certificates that are allowed by Suite B, see "Suite B cryptography support" on page 49.

GSK_SYSPLEX_SESSION_TICKET_CACHE

Specifies if sysplex session ticket caching for TLS V1.3 sessions is enabled for this SSL environment within this server application.

Specify GSK_SYSPLEX_SESSION_TICKET_CACHE_ON if sysplex session ticket caching for TLS V1.3 server sessions is enabled.

Specify GSK_SYSPLEX_SESSION_TICKET_CACHE_OFF if sysplex session ticket caching for TLS V1.3 server sessions is not enabled. This is the default setting.

GSK_SYSPLEX_SESSION_TICKET_CACHE can be specified only for an SSL environment.

GSK_SYSPLEX_SIDCACHE

Specifies if sysplex session caching for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 sessions is enabled for this SSL environment within this server application.

Specify GSK_SYSPLEX_SIDCACHE_ON if sysplex session caching for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 server sessions is enabled.

Specify GSK_SYSPLEX_SIDCACHE_OFF if sysplex session caching for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 server sessions is not enabled. This is the default setting.

GSK_SYSPLEX_SIDCACHE can be specified only for an SSL environment.

GSK_T61_AS_LATIN1

Specify GSK_T61_AS_LATIN1_ON to use the ISO8859-1 character set when processing a TELETEX string. Specify GSK_T61_AS_LATIN1_OFF to use the T.61 character set. The default is to use the ISO8859-1 character set.

Note: Selecting the incorrect character set can cause strings to be converted incorrectly. GSK_T61_AS_LATIN1 can be specified only for an SSL environment. This setting is global and affects all string conversions for all SSL environments.

GSK_TLS_CBC_PROTECTION_METHOD

Specifies an optional SSL V3.0 or TLS V1.0 CBC IV protection method when writing application data.

Specify GSK_TLS_CBC_PROTECTION_METHOD_NONE to indicate that no CBC protection is enabled. This is the default.

Specify GSK_TLS_CBC_PROTECTION_METHOD_ZEROBYTEFRAGMENT to indicate that zero byte record fragmenting is enabled. When specified, a zero byte record fragment is sent before the application data records are sent.

Specify GSK_TLS_CBC_PROTECTION_METHOD_ONEBYTEFRAGMENT to indicate that one byte record fragmenting is enabled. When specified, the first record is sent in two record fragments with the first record fragment containing only one byte of application data. The rest of the application data from the first record is sent in the second record fragment. All the following records are written whole. For example, a write of 256 bytes of data that is broken into 64 byte record fragments would be written as:

```
1 byte, 63 bytes, 64 bytes, 64 bytes, 64 bytes
```

GSK_TLS_CBC_PROTECTION_METHOD can be specified only for an SSL environment.

GSK_V3_CIPHERS

Specify GSK_V3_CIPHERS_CHAR2 if the cipher specification is specified using 1 or more 2-character values in GSK_V3_CIPHER_SPECS. Specify GSK_V3_CIPHERS_CHAR4 if the cipher specification is specified using 1 or more 4-character values in GSK_V3_CIPHER_SPECS_EXPANDED. GSK_V3_CIPHERS can be specified for an SSL environment or an SSL connection.

GSK_WILDCARD_VALIDATION_ENABLE

Specifies whether an asterisk as the wildcard character will be acceptable to replace zero or more characters within the server certificate's DNS entry for the subject alternative name or subject DN common name value. For more information on wildcarding, see "Server certificate domain-based validation" on page 70.

Specify GSK_WILDCARD_VALIDATION_ENABLE_ON to accept the wildcard character.

Specify GSK_WILDCARD_VALIDATION_ENABLE_OFF to not accept the wildcard character.

GSK_WILDCARD_VALIDATION_ENABLE may be specified for an SSL environment or an SSL connection.

Related topics

- "gsk_attribute_get_enum()" on page 87
- "gsk_environment_open()" on page 136
- "gsk_environment_init()" on page 134
- "gsk_secure_socket_open()" on page 173
- "gsk_secure_socket_init()" on page 157

gsk_attribute_set_numeric_value()

Sets a numeric value.

Format

```
#include <gskssl.h>

gsk_status gsk_attribute_set_numeric_value (
                                             gsk_handle      ssl_handle,
                                             GSK_NUM_ID      num_id,
                                             int             num_value)
```

Parameters

ssl_handle
Specifies an SSL environment handle returned by **gsk_environment_open()** or an SSL connection handle returned by **gsk_secure_socket_open()**.

num_id
Specifies the numeric identifier.

num_value
Specifies the numeric value.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ATTRIBUTE_INVALID_ID]
The numeric identifier is not valid or cannot be used with the specified handle.

[GSK_ATTRIBUTE_INVALID_NUMERIC_VALUE]
The numeric value is not within the valid range.

[GSK_INVALID_HANDLE]
The handle is not valid.

[GSK_INVALID_STATE]
The environment or connection is not in the open state.

Usage

The **gsk_attribute_set_numeric_value()** routine sets a numeric value for an SSL environment or an SSL connection. The environment or connection must be in the open state and not in the initialized state (that is, **gsk_environment_init()** or **gsk_secure_socket_init()** has not been called).

These numeric identifiers are supported:

GSK_CRL_CACHE_ENTRY_MAXSIZE
Sets the maximum size in bytes of a CRL that is allowed to be stored in the LDAP CRL cache. Any CRLs larger than this size are not cached. The valid cache entry sizes are 0 through 2147483647. The default is 0, which means there is no limit on the size of the CRL stored in the LDAP CRL cache.

GSK_CRL_CACHE_ENTRY_MAXSIZE can be specified only for an SSL environment.

GSK_CRL_CACHE_SIZE
Sets the maximum number of CRLs that are allowed to be stored in the LDAP CRL cache. The valid cache sizes are -1 through 32000. If LDAP extended CRL cache support is enabled, the default is 32 and CRLs are only cached if they contain an expiration time that is later than the current time. If LDAP basic CRL cache support is enabled, the default is -1 (which is unlimited) and caching only occurs if

GSK_CRL_CACHE_TIMEOUT is set to a value greater than 0. A value of 0 for GSK_CRL_CACHE_SIZE means that LDAP CRL caching is not enabled.

GSK_CRL_CACHE_SIZE can be specified only for an SSL environment.

GSK_CRL_CACHE_TEMP_CRL_TIMEOUT
Sets the time in hours that a temporary CRL cache entry resides in the LDAP CRL cache. A temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not reside on the LDAP server. The range is 1-720 hours and defaults to 24 hours.

Note: This support is only available when LDAP extended CRL cache support is activated and caching of temporary CRLs is enabled.

GSK_CRL_CACHE_TEMP_CRL_TIMEOUT can be specified only for an SSL environment.

GSK_CRL_CACHE_TIMEOUT
Sets the LDAP basic CRL cache timeout. This is the number of hours that a cached CRL remains valid in the LDAP basic CRL cache. The range is 0-720 and defaults to 24. A value of 0 disables LDAP CRL caching.

GSK_CRL_CACHE_TIMEOUT can be specified only for an SSL environment.

GSK_FD
Sets the socket descriptor for network operations. GSK_FD can be specified only for an SSL connection. The socket must not be closed until the **gsk_secure_socket_close()** routine has been called to terminate the secure connection.

GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE
Sets the maximum size in bytes of a CRL that is allowed to be stored in the HTTP CDP CRL cache. Any CRLs larger than this size are not cached. The valid sizes are 0 through 2147483647. The default is 0, which means there is no limit on the size of the CRL stored in the HTTP CDP CRL cache.

GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE can be specified only for an SSL environment.

GSK_HTTP_CDP_CACHE_SIZE
Sets the maximum number of CRLs that are allowed to be stored in the HTTP CDP CRL cache. The valid sizes are 0 through 32000. The default is 32 and a value of 0 means that HTTP CDP CRL caching is disabled.

GSK_HTTP_CDP_CACHE_SIZE can be specified only for an SSL environment.

GSK_HTTP_CDP_MAX_RESPONSE_SIZE
Sets the maximum size in bytes accepted as a response from an HTTP server when retrieving a CRL. The valid sizes are 0 through 2147483647. A value of 0 will disable checking the size and allow a CRL of any size. Setting the maximum response size too small could implicitly disable HTTP CRL support. The default is 204800 (200K).

GSK_HTTP_CDP_MAX_RESPONSE_SIZE can be specified only for an SSL environment.

GSK_HTTP_CDP_PROXY_SERVER_PORT
Sets the HTTP proxy server port for HTTP CDP CRL retrieval. The port must be between 1 and 65535. Port 80 is used if no HTTP proxy server port is set.

GSK_HTTP_CDP_PROXY_SERVER_PORT can be specified only for an SSL environment.

GSK_HTTP_CDP_RESPONSE_TIMEOUT
Sets the time in seconds to wait for a complete response from the HTTP server. The valid time limits are 0 through 43200 seconds (12 hours). The default is 15 seconds and a value of 0 means there is no time limit for HTTP CRL retrievals.

GSK_HTTP_CDP_RESPONSE_TIMEOUT can be specified only for an SSL environment.

GSK_LDAP_RESPONSE_TIMEOUT
Sets the time in seconds to wait for a response from the LDAP server. The valid time limits are 0 through 43200 seconds (12 hours). The default is 15 seconds and a value of 0 means that there is no time limit for LDAP CRL retrievals.

GSK_LDAP_RESPONSE_TIMEOUT can be specified only for an SSL environment.

GSK_LDAP_SERVER_PORT

Sets the LDAP server port. The port must be between 1 and 65535. Port 389 will be used if no LDAP server port is set.

GSK_LDAP_SERVER_PORT can be specified only for an SSL environment.

GSK_MAX_SOURCE_REV_EXT_LOC_VALUES

Sets the maximum number of locations values that will be contacted per HTTP CDP or AIA extension when attempting validation of a certificate. The locations for revocation information are specified by the accessLocation in the AIA certificate extension for OCSP and the distributionPoint in the CDP extension for HTTP CRLs. When an HTTP URI is present in an AIA or CDP extension, validation will attempt to contact the remote HTTP server to obtain revocation information. Both of these extensions can contain multiple location values and therefore have the potential to impact performance when there are a very large number of locations present. The valid values are 0 through 256. The default value is 10 and a value of 0 indicates there is no limit on the number of locations contacted.

GSK_MAX_SOURCE_REV_EXT_LOC_VALUES can be specified only for an SSL environment.

GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES

Sets the maximum number of location values that will be contacted when performing validation of a certificate. The locations for revocation information are specified by the accessLocation in the AIA certificate extension for OCSP and the distributionPoint in the CDP extension for HTTP CRLs. When an HTTP URI is present in an AIA or CDP extension, validation will attempt to contact the remote HTTP server to obtain revocation information. Both of these extensions can contain multiple location values and therefore has the potential to negatively impact performance when there are a very large number of locations present. The valid values are 0 through 1024. The default value for this option is 100 and a value of 0 indicates there is no limit on the number of locations contacted.

GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES can be specified only for an SSL environment.

GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE

Sets the maximum number of OCSP responses or cached certificate statuses that are allowed to be kept in the OCSP response cache for an issuing CA certificate. The valid sizes are 0 through 32000 and must be less than or equal to the size specified for GSK_OCSP_CLIENT_CACHE_SIZE. By default, the size is set to 0 which means there is no limit on the number of cached certificate statuses allowed for a specific issuing CA certificate other than the limit imposed by GSK_OCSP_CLIENT_CACHE_SIZE.

Note: GSK_OCSP_CLIENT_CACHE_SIZE specifies the total number of cached certificate statuses allowed in the entire OCSP cache.

If this count is exceeded, any expired certificate statuses are first removed. If there are no expired certificate statuses that have the same issuing CA certificate, the certificate status that is closest to the expiration time is removed first. This cache size is rounded up to the nearest multiple of 16 with a minimum size of 16.

GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE can be specified only for an SSL environment.

GSK_OCSP_CLIENT_CACHE_SIZE

Sets the maximum number of OCSP responses or cached certificate statuses to be kept in the OCSP response cache. The valid cache sizes are 0 through 32000 and defaults to 256. The OCSP response cache will be disabled if 0 is specified. The OCSP response cache will be allocated using the requested size rounded up to the nearest multiple of 16 with a minimum size of 16.

GSK_OCSP_CLIENT_CACHE_SIZE can be specified only for an SSL environment.

GSK_OCSP_MAX_RESPONSE_SIZE

Sets the maximum size in bytes allowed in a response from an OCSP responder. The valid response sizes are 0 through 2147483647 and defaults to 20480. A value of 0 will disable checking the size and allows an OCSP response of any size. Setting the maximum response size too small could implicitly disable OCSP support.

GSK_OCSP_MAX_RESPONSE_SIZE can be specified only for an SSL environment.

GSK_OCSP_NONCE_SIZE
Sets the size in bytes for the value of the nonce to be sent in OCSP requests.

The valid nonce sizes are 8 through 256 and defaults to 8.

GSK_OCSP_NONCE_SIZE can be specified only for an SSL environment.

GSK_OCSP_PROXY_SERVER_PORT
Sets the OCSP responder port for the proxy. The port must be between 1 and 65535. Port 80 is used if no OCSP proxy server port is set.

GSK_OCSP_PROXY_SERVER_PORT can be specified only for an SSL environment.

GSK_OCSP_RESPONSE_TIMEOUT
Sets the time in seconds to wait for a complete response from the OCSP responder. The valid time limits are 0 through 43200 seconds (12 hours) and defaults to 15 seconds. A value of 0 indicates there is no time limit for the retrieval of the OCSP response.

GSK_OCSP_RESPONSE_TIMEOUT can be specified only for an SSL environment.

GSK_PEER_DH_MIN_KEY_SIZE
Sets the minimum allowed X.509 certificate Diffie-Hellman key size for the peer end-entity certificate. GSK_PEER_DH_MIN_KEY_SIZE can be specified only for an SSL environment.

GSK_PEER_DSA_MIN_KEY_SIZE
Sets the minimum allowed X.509 certificate DSA key size for the peer end-entity certificate. GSK_PEER_DSA_MIN_KEY_SIZE can be specified only for an SSL environment.

GSK_PEER_ECC_MIN_KEY_SIZE
Sets the minimum allowed X.509 certificate ECC key size for the peer end-entity certificate. GSK_PEER_ECC_MIN_KEY_SIZE can be specified only for an SSL environment.

GSK_PEER_RSA_MIN_KEY_SIZE
Sets the minimum allowed X.509 certificate RSA key size for the peer end-entity certificate. GSK_PEER_RSA_MIN_KEY_SIZE can be specified only for an SSL environment.

GSK_SESSION_TICKET_CLIENT_MAXCACHED
Sets the maximum number of session tickets that are allowed to be cached by the client for each unique TLS V1.3 session. The valid maximum number of cached session tickets is 1 through 128 tickets and defaults to 8. The GSK_SESSION_TICKET_CLIENT_ENABLE setting must be enabled and the GSK_V3_SESSION_TIMEOUT and GSK_V3_SIDCACHE_SIZE settings must be set to values greater than 0 to allow for client session ticket caching.

GSK_SESSION_TICKET_CLIENT_MAXCACHED can be specified only for an SSL environment.

GSK_SESSION_TICKET_CLIENT_MAXSIZE
Specifies the maximum size in bytes of a session ticket that can be stored in the client session ticket cache. The valid sizes are 0 through 2147483647. The default size is 8192 (8K). A value of 0 will disable checking the session ticket size and allows a session ticket of any size. Setting the maximum session ticket size too small could implicitly disable session ticket caching on the client side. The GSK_V3_SESSION_TIMEOUT and GSK_V3_SIDCACHE_SIZE settings also must be set to values greater than 0 to allow client session ticket caching.

GSK_SESSION_TICKET_CLIENT_MAXSIZE can be specified only for an SSL environment.

GSK_SESSION_TICKET_SERVER_COUNT
Specifies the number of session tickets that will be sent by the server after the initial TLS V1.3 handshake has completed. Valid values are 0 through 16. The default value is 2. If the count is greater than 0, each subsequent resumed TLS V1.3 handshake will also send a single session ticket to replace the one used for resumption. Additionally, GSK_SESSION_TICKET_SERVER_ENABLE must be set to ON for the session tickets to be sent. The GSK_SEND_SESSION_TICKET option in **gsk_secure_socket_misc()** is another mechanism that can be used to send session tickets after the initial TLS V1.3 handshake negotiation is complete. However, it cannot be used in conjunction with GSK_SESSION_TICKET_SERVER_COUNT, which must be set to 0 for the call to **gsk_secure_socket_misc()** to be successful.

GSK_SESSION_TICKET_SERVER_COUNT can be specified only for an SSL environment.

GSK_SESSION_TICKET_SERVER_KEY_REFRESH

Specifies the key refresh interval in seconds of the encryption key used by the server to encrypt session tickets. Valid values are 0 through 604800. The default value is 300 seconds. If 0 is specified, the encryption key will never refresh. In order to encrypt and decrypt session tickets, GSK_SESSION_TICKET_SERVER_ENABLE must be set to ON and the server must be configured to send session tickets, either via GSK_SESSION_TICKET_SERVER_COUNT or via the GSK_SEND_SESSION_TICKET option in **gsk_secure_socket_misc()**.

When the encryption key is refreshed and a new primary encryption key is generated, the former encryption key is retained as a secondary key that can be used only for decryption until the subsequent refresh occurs. When the ticket is decrypted, the server only accepts the ticket if the GSK_SESSION_TICKET_SERVER_TIMEOUT has not yet passed.

GSK_SESSION_TICKET_SERVER_KEY_REFRESH can be specified only for an SSL environment.

GSK_SESSION_TICKET_SERVER_TIMEOUT

Specifies the maximum time that a server accepts a TLS V1.3 session resumption request from the client measured in seconds from the initial handshake. Valid values are 1 through 604800 seconds (seven days).

If sysplex session ticket caching is not enabled (GSK_SYSPLEX_SESSION_TICKET_CACHE is set to OFF) and GSK_SESSION_TICKET_SERVER_ENABLE is set to ON, the session ticket encryption key must be available when the client attempts TLS V1.3 resumption as the ticket needs to be decrypted. In this configuration, the GSK_SESSION_TICKET_SERVER_KEY_REFRESH value impacts the lifetime of a session ticket. The default session ticket timeout value is 300 seconds.

If sysplex session ticket caching is enabled (GSK_SYSPLEX_SESSION_TICKET_CACHE is set to ON), the default session ticket timeout value is 600 seconds.

GSK_SESSION_TICKET_SERVER_TIMEOUT can be specified only for an SSL environment.

GSK_V2_SESSION_TIMEOUT

Sets the SSL Version 2 session timeout. This is the number of seconds until an SSL V2 session identifier expires. The range is 0-100 and defaults to 100. System SSL remembers SSL V2 session identifiers for this amount of time. This reduces the amount of data exchanged during the SSL handshake when a complete initial handshake is performed. Session identifiers are not remembered if a value of 0 is specified.

GSK_V2_SESSION_TIMEOUT can be specified only for an SSL environment.

GSK_V2_SIDCACHE_SIZE

Sets the size of the SSL Version 2 session identifier cache. The oldest entry is removed when the cache is full to add a new entry. The range is 0-32000 and defaults to 256. Session identifiers are not remembered if a value of 0 is specified. The session identifier cache is allocated using the requested size rounded up to a power of 2 with a minimum size of 16.

GSK_V2_SIDCACHE_SIZE can be specified only for an SSL environment.

GSK_V3_SESSION_TIMEOUT

Specifies the session timeout value in seconds for the SSL V3 to TLS V1.2 session identifiers and TLS V1.3 session tickets in the cache. This is the number of seconds until an SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 session identifier or TLS V1.3 session ticket expires. The range is 0-86400 and defaults to 86400. System SSL keeps the SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 session identifiers or TLS V1.3 session tickets for this amount of time in the cache. This reduces the amount of data exchanged during the SSL/TLS handshake when a complete initial handshake has already been performed. Session identifiers and session tickets are not remembered if a value of 0 is specified.

GSK_V3_SESSION_TIMEOUT can be specified only for an SSL environment.

GSK_V3_SIDCACHE_SIZE

Specifies the size in number of entries in the SSL V3 to TLS V1.2 session identifier and TLS V1.3 session ticket cache. The oldest entry will be removed when the cache is full in order to add a new entry. The range is 0-64000 and defaults to 512. Session identifiers and session tickets are not remembered if a value of 0 is specified. For the SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 protocols, the cache stores session identifiers for use on the server and client sides. For the TLS V1.3 protocol on the client side, the cache is used to store session tickets when GSK_SESSION_TICKET_CLIENT_ENABLE is set to ON. The session identifier and session ticket cache is allocated by using the requested size rounded up to a power of 2 with a minimum size of 16.

GSK_V3_SIDCACHE_SIZE can be specified only for an SSL environment.

Related topics

- "gsk_attribute_get_numeric_value()" on page 97
- "gsk_environment_open()" on page 136
- "gsk_environment_init()" on page 134
- "gsk_secure_socket_init()" on page 157
- "gsk_secure_socket_open()" on page 173

gsk_attribute_set_tls_extension()

Defines a TLS extension to the SSL environment or connection.

Format

```
#include <gskssl.h>

gsk_attribute_set_tls_extension (
                                 gsk_handle             ssl_handle,
                                 gsk_tls_extension *    tls_extension)
```

Parameters

ssl_handle
Specifies an SSL environment handle returned by **gsk_environment_open()** or an SSL connection handle returned by **gsk_secure_socket_open()**.

tls_extension
Specifies the TLS extension structure containing extension data.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ATTRIBUTE_INVALID_TLS_EXTENSION]
The TLS extension type identifier is not valid or cannot be used with the specified handle.

[GSK_ATTRIBUTE_INVALID_TLS_EXT_DATA]
TLS extension data has been incorrectly defined.

[GSK_INVALID_HANDLE]
The handle is not valid.

[GSK_INVALID_STATE]
The handle is closed.

Usage

The **gsk_attribute_set_tls_extension()** routine defines a TLS extension for an SSL environment or an SSL connection. The environment or connection must be in the open state and not in the initialized state (that is, **gsk_environment_init()** or **gsk_secure_socket_init()** is not called). TLS extensions that are defined for an SSL environment applies to all connections made as part of that environment unless explicitly deactivated or replaced using a call to **gsk_attribute_set_tls_extension()** for the connection. TLS extensions are applied to TLS V1.0 or higher connections only.

The application must prime the TLS extension structure with the appropriate TLS extension data before calling the routine, including the TLS extension type identifier and the specific data that is required for the TLS extension type. The TLS extension may be designated as required or optional in the *gsk_tls_extension* structure. A required setting enforces support requirements of the specific extension type on the communicating partner. If the partner indicates that it does not support the extension, the connection is rejected. An optional setting allows the connection to continue without support for that particular extension type if the communicating partner indicates that it does not support the TLS extension type.

Note:

1. Setting an extension as required for a server means that all clients connecting to the server must have the extension enabled. Failure for a client to do so results in the server rejecting the connection request from the client. It is recommended that for maximum interoperability, that the required field is not enabled on the server side.

2. The gsk_tls_extension structure contains a 32-byte field, *rsvd*, which is reserved for future use. This field must contain binary zeros; any non-zero data results in **gsk_attribute_set_tls_extension()** returning a GSK_ATTRIBUTE_INVALID_TLS_EXT_DATA error.
3. Definition of TLS extensions for the client when any of the TLS protocols are enabled prevents the SSL V2 protocol from being used.

The values set by using this service are treated as independent values. They are not validated with other values set using **gsk_attribute_set_buffer()**, **gsk_attribute_set_enum()**, or **gsk_attribute_set_tls_extensions()** APIs until used together to perform an SSL/TLS handshake by calling **gsk_secure_socket_init()**.

These TLS extension type identifiers are supported:

GSK_TLS_EXTID_CLIENT_MFL
Specifies the Maximum Fragment Length type TLS extension requirements for the TLS client. Specify the size of the maximum fragment length to be used using settings GSK_TLS_MFL_512 (2^9 bytes), GSK_TLS_MFL_1024 (2^{10}), GSK_TLS_MFL_2048 (2^{11}) or GSK_TLS_MFL_4096 (2^{12}). The GSK_TLS_MFL_OFF setting deactivates a previously registered GSK_TLS_EXTID_CLIENT_MFL type TLS extension setting.

GSK_TLS_EXTID_SERVER_MFL
Specifies the Maximum Fragment Length type TLS extension requirements for the TLS server. Specify to the TLS server whether to support the Maximum Fragment Length TLS extension using the GSK_TLS_MFL_ON setting. The GSK_TLS_MFL_OFF setting deactivates a previously registered GSK_TLS_EXTID_SERVER_MFL type TLS extension setting.

GSK_TLS_EXTID_SNI_CLIENT_SNAMES
Specifies the server name (or list of server names) that the client sends to the server in a Server Name Indication type TLS extension to indicate with which server the client wants to communicate. The list of server names is defined using a pointer to an array of pointers to strings containing the server names.

Set the *setSni* setting of the *gsk_sni_client_names* extension data to TRUE to register the extension data with the SSL environment or connection. A *setSni* setting of FALSE deactivates a previously registered GSK_TLS_EXTID_SNI_CLIENT_SNAMES type TLS extension setting.

If the TLS server does not recognize any server names in the clients server name list, the server sends an unrecognized_name alert to the client, which, by default, is a warning. Set the *unrecognized_name_fatal* flag in the *gsk_sni_client_names* extension data to TRUE to treat the unrecognized_name alert as fatal and close the connection.

GSK_TLS_EXTID_SNI_CLIENT_SNAMES can be defined on both the server and client sides. Its settings, however, are effective when running as a client; it is ignored for servers.

Note:

1. It is recommended that the *gsk_sni_client_snames* structure to be included in the *gsk_tls_extension* data be initialized with binary zeros before setting the required server label data. This will ensure future application compatibility when additional bits within the *gsk_sni_client_snames* structure are used.
2. System SSL only supports server names that contain US-ASCII characters.

GSK_TLS_EXTID_SNI_SERVER_LABELS
Specifies the pairings of server name to certificate key label to be used when the TLS server receives a Server Name Indication type TLS extension from the TLS client. The server name/key label pairs are used with the server name details received from the client to determine which certificate from the key database, PKCS #12 file, key ring or token is sent to the client as the servers certificate.

Set the *setSni* setting of the *gsk_sni_server_labels* extension data to TRUE to register the extension data with the SSL environment or connection. A *setSni* setting of FALSE deactivates a previously registered GSK_TLS_EXTID_SNI_SERVER_LABELS type TLS extension setting.

If the TLS server does not recognize any server names in the clients server name list, the server sends an unrecognized_name alert to the client, which, by default, is a warning. Set the *unrecognized_name_fatal* flag in the *gsk_sni_server_labels* extension data to TRUE to treat the unrecognized_name alert as fatal and close the connection.

GSK_TLS_EXTID_SNI_SERVER_LABELS can be defined on both the server and client sides. Its settings, however, are effective when running as a server; it is ignored for clients.

Note:

1. It is recommended that the *gsk_sni_server_labels* structure to be included in the *gsk_tls_extension* data be initialized with binary zeros before setting the required server label data. This ensures future application compatibility when additional bits within the *gsk_sni_server_labels* structure are used.
2. System SSL only supports server names that contain US-ASCII characters.

GSK_TLS_EXTID_TRUNCATED_HMAC

Specifies whether the TLS server or client supports the Truncated HMAC type TLS extension. Set *truncateHmac* to TRUE to enable the extension. A *truncateHmac* setting of FALSE deactivates a previously registered GSK_TLS_EXTID_TRUNCATED_HMAC type TLS extension setting.

Note: The Truncated HMAC extension is no longer supported in the TLS V1.3 protocol. If the TLS client is only enabled for the TLS V1.3 protocol, the Truncated HMAC extension is not sent to the TLS server. Because the Truncated HMAC extension is no longer supported in the TLS V1.3 protocol, a TLS server that attempts a TLS V1.3 protocol negotiation does not include the Truncated HMAC extension.

The required setting on the Truncated HMAC extension is ignored by the TLS server or client if a TLS V1.3 connection is negotiated.

gsk_environment_close()

Closes an SSL environment.

Format

```
#include <gskssl.h>

gsk_status gsk_environment_close (
                                   gsk_handle *     env_handle)
```

Parameters

env_handle
Specifies the SSL environment handle returned by the **gsk_environment_open()** routine. The environment handle will be set to NULL upon completion.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_INVALID_HANDLE]
The environment handle is not valid.

[GSK_INVALID_STATE]
The environment is already closed.

Usage

The **gsk_environment_close()** routine closes an environment created by the **gsk_environment_open()** routine. The storage that is allocated for the environment is not released until all connections created using the environment are closed. The SSL environment cannot be used to create new connections upon completion of the close.

Related topics

- "gsk_environment_open()" on page 136
- "gsk_environment_init()" on page 134
- "gsk_secure_socket_init()" on page 157
- "gsk_secure_socket_close()" on page 156

gsk_environment_init()

Initializes an SSL environment.

Format

```
#include <gskssl.h>

gsk_status gsk_environment_init (
                                  gsk_handle      env_handle)
```

Parameters

env_handle
Specifies the SSL environment handle returned by the **gsk_environment_open()** routine.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_CERTIFICATE_NOT_AVAILABLE]
The key database, PKCS #12 file, key ring or token does not contain any certificates.

[GSK_ERR_BAD_KEYFILE_PASSWORD]
The key database or PKCS #12 file password is not correct.

[GSK_ERR_BAD_SIG_ALG_PAIR]
Signature algorithms pairs list is not valid.

[GSK_ERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[GSK_ERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[GSK_ERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[GSK_ERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[GSK_ERR_LDAP]
Unable to initialize the LDAP client.

[GSK_ERR_LDAP_NOT_AVAILABLE]
The LDAP server is not available.

[GSK_ERR_OCSP_NOT_ENABLED]
OCSP stapling requires OCSP support to be enabled.

[GSK_ERR_OCSP_REQUEST_SIGALG_NOT_VALID]
OCSP request signature algorithm pair is not valid.

[GSK_ERR_PERMISSION_DENIED]
Not authorized to access key database, PKCS #12 file, SAF key ring or z/OS PKCS #11 token.

[GSK_INSUFFICIENT_STORAGE]
Insufficient storage is available.

[GSK_INTERNAL_ERROR]
An internal processing error has occurred.

[GSK_INVALID_HANDLE]
The environment handle is not valid.

[GSK_INVALID_STATE]
The environment is not in the open state.

[GSK_KEYFILE_INVALID_FORMAT]
The database is not a key database.

[GSK_KEYFILE_IO_ERR]
An input/output error occurred while reading the key database, PKCS #12 file, key ring or token.

[GSK_KEYFILE_PASSWORD_EXPIRED]
The key database password is expired.

[GSK_KEYRING_OPEN_ERROR]
Unable to open the key database, PKCS #12 file, key ring or token.

[GSK_NO_KEYFILE_PASSWORD]
The key database password is not available.

Usage

The **gsk_environment_init()** routine initializes an SSL environment created by the **gsk_environment_open()** routine. After the SSL environment has been initialized, it can be used to create one or more SSL connections by calling the **gsk_secure_socket_open()** routine. The **gsk_environment_close()** routine should be called to close the environment when it is no longer needed. The **gsk_environment_close()** routine should also be called if an error is returned by the **gsk_environment_init()** routine.

If TLS V1.3 is enabled in the SSL environment handle, the following updated defaults are used for the signature algorithm pairs and supported elliptic curve list:

- The supported elliptic curve list is set to "00210023002400250019 0029".
- The signature algorithm pair list is set to "0601060305010503040104030402030103030302020102030202080608050804".

Related topics

- “gsk_environment_open()” on page 136
- “gsk_environment_close()” on page 133
- “gsk_secure_socket_open()” on page 173

gsk_environment_open()

Creates an SSL environment.

Format

```
#include <gskssl.h>

gsk_status gsk_environment_open (
                                  gsk_handle *      env_handle)
```

Parameters

env_handle
Returns the handle for the environment. The application should call the **gsk_environment_close()** routine to release the environment when it is no longer needed.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ATTRIBUTE_INVALID_ENUMERATION]
The value of an environment variable is not valid.

[GSK_ATTRIBUTE_INVALID_LENGTH]
The length of an environment variable value is not valid.

[GSK_ATTRIBUTE_INVALID_NUMERIC_VALUE]
The value of an environment variable is not valid.

[GSK_ERR_PROTOCOL_NOT_SUPPORTED_WITH_FIPS]
Protocol is not supported in FIPS mode.

[GSK_INSUFFICIENT_STORAGE]
Insufficient storage is available.

Usage

The **gsk_environment_open()** routine creates an SSL environment. The environment will be initialized with default values and then any SSL environment variables will be processed. These values can be changed by the application using the appropriate **gsk_attribute_set_*()** routines. The **gsk_environment_init()** routine should then be called to initialize the SSL environment. This environment can then be used to establish one or more SSL connections.

When not executing in FIPS mode, the following default values are set:

- TLS V1.0 is enabled (SSL V2, SSL V3, TLS V1.1, TLS V1.2, and TLS V1.3 are disabled by default).
- The connection type is set to CLIENT.
- The SSL V2 connection timeout is set to 100 seconds.
- The SSL V3 connection timeout is set to 86400 seconds.
- The SSL V2 cache size is set to 256.
- The SSL V3 cache size is set to 512.
- The sysplex session cache is disabled.
- The default key will be used.
- No revoked certificate checking performed.
- The default callback routines will be used.

- The SSL V2 cipher specification is set to "34" if United States only encryption is enabled (System SSL Security Level 3 FMID installed or CPACF Feature 3863 installed) and "4" otherwise.
- 2-character cipher definitions in GSK_V3_CIPHER_SPECS will be used for SSL V3 cipher values.
- The SSL V3 cipher specification is set to "3538392F3233" if United States only encryption is enabled (System SSL Security Level 3 FMID installed or CPACF Feature 3863 installed) and "" (empty string - no default) otherwise.
- The Signaling Cipher Suite Value (SCSV) is disabled.
- The supported client elliptic curve list is set to "00210023002400250019".
- The allowed server elliptic curve list is set to "00230024002500210019".
- The signature algorithm pair list is set to "0601060305010503040104030402030103030302020102030202".
- No TLS extensions are initialized.
- Suite B is disabled.
- OCSP revocation and OCSP server stapling support is disabled (OCSP URL is not defined and AIA extensions are not enabled).
- HTTP CDP CRL support is disabled.
- LDAP CRL support is disabled.
- Strict 3DES key enforcement is not enabled.
- Minimum peer x.509 end-entity certificate key size is set to RSA (1024), DSA(1024), DH(1024), and ECC(192).
- The client and server key share lists have no default value.
- Middlebox compatibility mode for TLS V1.3 connections is disabled.
- Session ticket support is enabled for both the client and the server.
- The maximum session ticket size accepted by the client is 8192 (8K) bytes.
- The algorithm used to encrypt server session tickets is set to AES-CBC 128.
- The number of session tickets that are sent by the server after a TLS V1.3 handshake has completed is set to 2.
- The server session ticket key refresh is set to 300 seconds.
- The sysplex session cache for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 server sessions is disabled.
- The sysplex session ticket cache for TLS V1.3 server sessions is disabled.
- The maximum number of session tickets that are allowed to be cached by the client for a TLS V1.3 session is set to 8.
- The server session ticket timeout is set to 300 seconds when sysplex session ticket cache is disabled. If sysplex session ticket cache is enabled, the server session ticket timeout is set to 600 seconds.
- The signature algorithm cert list is empty.
- The client (when enabled for TLS V1.0, TLS V1.1, or TLS V1.2) sends the extended master secret extension to the server, but does not require the server to negotiate the extension.
- The server (when enabled for TLS V1.0, TLS V1.1, or TLS V1.2) supports negotiating the extended master secret if the client sends the extension, but does not require the client to send the extension.

When executing in FIPS mode, the following default values are set:

- TLS V1.0 is enabled (SSL V2, SSL V3, TLS V1.1, TLS V1.2, and TLS V1.3 are disabled by default).
- The connection type is set to CLIENT.
- The connection timeout is set to 86400 seconds.
- The cache size is set to 512.
- The sysplex session cache is disabled.
- The default key will be used.

- No revoked certificate checking performed.
- The default callback routines will be used.
- 2-character cipher definitions in GSK_V3_CIPHER_SPECS will be used for SSL V3 cipher values.
- The SSL V3 cipher specification is set to "3538392F3233".
- The Signaling Cipher Suite Value (SCSV) is disabled.
- The supported client elliptic curve list is set to "00210023002400250019".
- The allowed server elliptic curve list is set to "00230024002500210019".
- The signature algorithm pair list is set to "0601060305010503040104030402030103030302020102030202".
- Suite B is disabled.
- OCSP revocation and OCSP server stapling support is disabled (OCSP URL is not defined and AIA extensions are not enabled).
- HTTP CDP CRL support is disabled.
- LDAP CRL support is disabled.
- Strict 3DES key enforcement is enabled.
- Minimum peer x.509 end-entity certificate key size is set to RSA (1024), DSA(1024), DH(2048), and ECC(192).
- The signature algorithm cert list is empty.
- The client (when enabled for TLS V1.0, TLS V1.1, or TLS V1.2) sends the extended master secret extension to the server, but does not require the server to negotiate the extension.
- The server (when enabled for TLS V1.0, TLS V1.1, or TLS V1.2) supports negotiating the extended master secret if the client sends the extension, but does not require the client to send the extension.

Note: The TLS V1.3 protocol is not currently supported in FIPS mode. If an attempt is made to enable this protocol while running in FIPS mode, an error is returned during environment initialization.

See Table 25 on page 795 for a list of supported SSL V2 cipher specifications.

See Table 26 on page 795 for a list of supported 2-character SSL V3 cipher specifications.

See Table 27 on page 799 for a list of supported 4-character SSL V3 cipher specifications.

See Table 29 on page 804 for a list of supported 4-character elliptic curve specifications.

Applications wanting to use cipher suites that use elliptic curve certificates must set an appropriate cipher specification in GSK_V3_CIPHER_SPECS_EXPANDED. If an application requires an SSL V3, TLS V1.0, or higher session to use the 4-character cipher suites specified in GSK_V3_CIPHER_SPECS_EXPANDED then it must explicitly call **gsk_attribute_set_enum()** and set the enumeration identifier GSK_V3_CIPHERS to have a value of GSK_V3_CIPHERS_CHAR4.

If the application is to be enabled for TLS V1.3, see "TLS V1.3 protocol support" on page 61 for the configuration requirements.

If an application has indicated it is using the 4-character cipher specifications by setting GSK_V3_CIPHERS to GSK_V3_CIPHERS_CHAR4, but does not set a cipher specification in GSK_V3_CIPHER_SPECS_EXPANDED, the default cipher specification will be set as follows:

- Executing in non-FIPS mode with United States only encryption enabled (System SSL Security Level 3 FMID installed or CPACF Feature 3863 installed):

  ```
  "003500380039002F00320033"
  ```

- Executing in non-FIPS mode with United States only encryption disabled (System SSL Security Level 3 FMID is not installed and CPACF Feature 3863 is not installed):

  ```
  ("" empty string – no default)
  ```

- Executing in FIPS mode:

```
"003500380039002F00320033"
```

If an application has indicated it will be running in Suite B compatibility mode by setting GSK_SUITE_B_PROFILE to a value other than GSK_SUITE_B_PROFILE_OFF, the cipher specifications in GSK_V2_CIPHER_SPECS, GSK_V3_CIPHER_SPECS, and GSK_V3_CIPHER_SPECS_EXPANDED are ignored. The Suite B ciphers that are in use in the initialized SSL environment can be obtained by calling the **gsk_attribute_get_buffer()** routine by passing in the GSK_SUITE_B_CIPHER_SPECS buffer identifier. The cipher specifications will be set based on the values for GSK_SUITE_B_PROFILE as follows:

- Executing with GSK_SUITE_B_PROFILE_128 "C02BC023".
- Executing with GSK_SUITE_B_PROFILE_128MIN "C02BC02C".
- Executing with GSK_SUITE_B_PROFILE_192 "C02CC024".
- Executing with GSK_SUITE_B_PROFILE_192MIN "C02C".
- Executing with GSK_SUITE_B_PROFILE_ALL "C02CC024C02BC023".

If executing in FIPS mode, the following cipher specifications are supported:

- When using 2-character cipher suites:

  0A 0D 10 13 16 2F 30 31 32 33 35 36 37 38 39 3C 3D 3E 3F 40 67 68 69
  6A 6B 9C 9D 9E 9F A0 A1 A2 A3 A4 A5

- When using 4-character cipher suites:

  000A 000D 0010 0013 0016 002F 0030 0031 0032 0033 0035 0036 0037 0038
  0039 003C 003D 003E 003F 0040 0067 0068 0069 006A 006B 009C 009D 009E
  009F 00A0 00A1 00A2 00A3 00A4 00A5 C003 C004 C005 C008 C009 C00A
  C00D C00E C00F C012 C013 C014 C023 C024 C025 C026 C027 C028 C029
  C02A C02B C02C C02D C02E C02F C030 C031 C032

If using TLS V1.1 or TLS V1.2 protocols, export ciphers are not supported. The 40-bit ciphers (cipher specifications "03" and "06" or "0003" and "0006") will be ignored if specified.

If using TLS V1.2, the 56-bit DES cipher suites "09", "0C", "0F", "12" and "15" (or "0009", "000C", "000F", "0012" and "0015") will be ignored if specified.

These environment variables are processed (See Appendix A, “Environment variables,” on page 763 for information about environment variables):

GSK_3DES_KEYCHECK
Specifies that each part of a Triple DES key is checked to be unique when in non-FIPS mode.

GSK_AIA_CDP_PRIORITY
Specifies the priority order that the AIA and the CDP extensions are checked for certificate revocation information.

GSK_CERT_DIAG_INFO
Specifies that when validation is being performed against the peer’s certificate and certificate chain, certificate validation information will be provided for failures, successes, or both through the GSK_CERT_DIAGNOSTIC_CALLBACK.

GSK_CERT_VALIDATION_KEYRING_ROOT
Specifies how certificates in a SAF key ring are validated.

GSK_CERT_VALIDATION_MODE
Specifies which internet standard is used for certificate validation.

GSK_CLIENT_AUTH_NOCERT_ALERT
Specifies whether the SSL server application accepts a connection from a client where client authentication is requested and the client fails to supply an X.509 certificate.

GSK_CLIENT_ECURVE_LIST
Specifies the list of elliptic curves or groups that are supported by the client.

GSK_CLIENT_EPHEMERAL_DH_GROUP_SIZE
Specifies the minimum Diffie-Hellman group size required by the client to be used by the server for an ephemeral Diffie-Hellman key exchange.

GSK_CLIENT_EXTENDED_MASTER_SECRET
Specifies if the TLS client sends the extended master secret extension to the server for TLS V1.0, TLS V1.1, or TLS V1.2 handshakes.

GSK_CLIENT_TLS_KEY_SHARES
Specifies the list of key share groups that are supported by the client during a TLS V1.3 handshake.

GSK_CRL_CACHE_ENTRY_MAXSIZE
Specifies the maximum size in bytes of a CRL to be kept in the LDAP CRL cache.

GSK_CRL_CACHE_EXTENDED
Specifies that LDAP extended CRL cache support is enabled.

GSK_CRL_CACHE_SIZE
Specifies the maximum number of CRLs that are allowed to be stored in the LDAP CRL cache.

GSK_CRL_CACHE_TIMEOUT
Specifies the number of hours that a cached LDAP CRL remains valid.

GSK_CRL_CACHE_TEMP_CRL
Specifies if a temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not reside on the LDAP server.

GSK_CRL_CACHE_TEMP_CRL_TIMEOUT
Specifies the time in hours that a temporary CRL cache entry resides in the LDAP CRL cache.

GSK_CRL_SECURITY_LEVEL
Specifies the level of security used when contacting LDAP servers to check CRLs for revoked certificates.

GSK_EXTENDED_RENEGOTIATION_INDICATOR
Specifies the level of enforcement of renegotiation indication.

GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE
Specifies the maximum size in bytes of a CRL that can be stored in the HTTP CDP CRL cache.

GSK_HTTP_CDP_CACHE_SIZE
Specifies the maximum number of CRLs that are allowed to be stored in the HTTP CDP CRL cache.

GSK_HTTP_CDP_ENABLE
Specifies if certificate revocation checking with the HTTP URI values in the CDP extension is enabled.

GSK_HTTP_CDP_MAX_RESPONSE_SIZE
Specifies the maximum size in bytes accepted as a response from an HTTP server when retrieving a CRL.

GSK_HTTP_CDP_PROXY_SERVER_NAME
Specifies the DNS name or IP address of the HTTP proxy server.

GSK_HTTP_CDP_PROXY_SERVER_PORT
Specifies the HTTP proxy server port.

GSK_HTTP_CDP_RESPONSE_TIMEOUT
Specifies the time in seconds to wait for a response from the HTTP server.

GSK_KEY_LABEL
Specifies the label of the key that used to authenticate the application.

GSK_KEYRING_FILE
Specifies the name of the key database file, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token.

GSK_KEYRING_PW
Specifies the password for the key database or PKCS #12 file.

GSK_KEYRING_STASH
Specifies the name of the key database password stash file.

GSK_LDAP_PASSWORD
Specifies the password to use when connecting to the LDAP server.

GSK_LDAP_PORT
Specifies the LDAP server port.

GSK_LDAP_RESPONSE_TIMEOUT
Specifies the time in seconds to wait for a response from the LDAP server.

GSK_LDAP_SERVER
Specifies one or more blank-separated LDAP server host names.

GSK_LDAP_USER
Specifies the distinguished name to use when connecting to the LDAP server.

GSK_MAX_SOURCE_REV_EXT_LOC_VALUES
Specifies the maximum number of location values that will be contacted per data source when attempting validation of a certificate.

GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES
Specifies the maximum number of location values that will be contacted when performing validation of a certificate.

GSK_MIDDLEBOX_COMPAT_MODE
Specifies that the TLS V1.3 handshake process ought to use or tolerate handshake messages in a manner compliant with earlier TLS protocols to alleviate possible issues with middleboxes or proxies.

GSK_OCSP_CLIENT_CACHE_ENTRY_MAXSIZE
Specifies the maximum number of OCSP responses or cached certificate statuses that are allowed to be kept in the OCSP response cache for an issuing CA certificate.

GSK_OCSP_CLIENT_CACHE_SIZE
Specifies the maximum number of OCSP responses or cached certificate statuses to be kept in the OCSP response cache.

GSK_OCSP_ENABLE
Specifies whether the AIA extensions are to be used for revocation checking.

GSK_OCSP_MAX_RESPONSE_SIZE
Specifies the maximum size in bytes allowed in a response from an OCSP responder.

GSK_OCSP_NONCE_CHECK_ENABLE
Specifies if OCSP response nonce checking is on or off.

GSK_OCSP_NONCE_GENERATION_ENABLE
Specifies whether an OCSP request will contain a generated nonce.

GSK_OCSP_NONCE_SIZE
Specifies the size in bytes for the value of the nonce to be sent in OCSP requests.

GSK_OCSP_PROXY_SERVER_NAME
Specifies the DNS name or IP address of the OCSP proxy server.

GSK_OCSP_PROXY_SERVER_PORT
Specifies the OCSP responder port for the proxy.

GSK_OCSP_REQUEST_SIGALG
Specifies the hash and signature algorithm pair to be used to sign OCSP requests.

GSK_OCSP_REQUEST_SIGKEYLABEL
Specifies the label of the key to be used to sign OCSP requests.

GSK_OCSP_RESPONSE_SIGALG_PAIRS
Specifies a preference ordered list of hash and signature algorithm pair specifications that are sent on the OCSP request and may be used by the OCSP responder to select an appropriate algorithm for signing the OCSP response.

GSK_OCSP_RESPONSE_TIMEOUT
Specifies the time in seconds to wait for a complete response from the OCSP responder.

GSK_OCSP_RETRIEVE_VIA_GET
Specifies whether the HTTP request to the OCSP responder is sent using the HTTP Get Method or the HTTP Post method.

GSK_OCSP_URL
Specifies the URI of an OCSP responder.

GSK_OCSP_URL_PRIORITY
Specifies the order of precedence for contacting OCSP responder locations if both GSK_OCSP_URL and GSK_OCSP_ENABLE are active.

GSK_PEER_CERT_MIN_VERSION
Specifies the minimum X.509 version number allowed for a peer's X.509 end-entity certificate.

GSK_PEER_DH_MIN_KEY_SIZE
Specifies the minimum allowed X.509 certificate Diffie-Hellman key size for a peer's X.509 end-entity certificate.

GSK_PEER_DSA_MIN_KEY_SIZE
Specifies the minimum allowed X.509 certificate DSA key size for a peer's X.509 end-entity certificate.

GSK_PEER_ECC_MIN_KEY_SIZE
Specifies the minimum allowed X.509 certificate ECC key size for a peer's X.509 end-entity certificate.

GSK_PEER_RSA_MIN_KEY_SIZE
Specifies the minimum allowed X.509 certificate RSA key size for a peer's X.509 end-entity certificate.

GSK_PROTOCOL_SSLV2
Specifies whether the SSL V2 protocol is supported.

GSK_PROTOCOL_SSLV3
Specifies whether the SSL V3 protocol is supported.

GSK_PROTOCOL_TLSV1
Specifies whether the TLS V1.0 protocol is supported.

GSK_PROTOCOL_TLSV1_1
Specifies whether the TLS V1.1 protocol is supported.

GSK_PROTOCOL_TLSV1_2
Specifies whether the TLS V1.2 protocol is supported.

GSK_PROTOCOL_TLSV1_3
Specifies whether the TLS V1.3 protocol is supported.

GSK_REFERENCE_ID_CN
Specifies a list of common name values to compare against the server certificate’s subject DN common name.

GSK_REFERENCE_ID_DNS
Specifies a list of DNS values to compare against the server certificate’s DNS entry for the subject alternative name.

GSK_REVOCATION_SECURITY_LEVEL
Specifies the level of security to be used when contacting an OCSP responder or an HTTP server specified in a URI value of the CDP extension.

GSK_RENEGOTIATION
Specifies the type of session renegotiation allowed for an SSL environment.

GSK_RENEGOTIATION_PEER_CERT_CHECK
Specifies if the peer certificate is allowed to change during renegotiation.

GSK_SERVER_ALLOWED_KEX_ECURVES
Specifies the list of elliptic curves that are allowed by the server for the TLS V1.0, TLS V1.1, and TLS V1.2 server key exchange.

GSK_SERVER_EPHEMERAL_DH_GROUP_SIZE
Specifies the minimum Diffie-Hellman group size to be used by the server for an ephemeral Diffie-Hellman key exchange.

GSK_SERVER_EXTENDED_MASTER_SECRET
Specifies if the TLS server supports negotiating the extended master secret extension from clients for TLS V1.0, TLS V1.1, or TLS V1.2 handshakes.

GSK_SERVER_FALLBACK_SCSV
Specifies if the server accepts the TLS fallback Signaling Cipher Suite Value (SCSV) when the client's cipher list includes it during an SSL or TLS handshake.

GSK_SERVER_KEY_LABEL_LIST
Specifies one or more labels that can be used to authenticate the server application.

GSK_SERVER_OCSP_STAPLING
Specifies if the server supports the retrieval of the OCSP responses for the server's end entity certificate or the server's certificate chain if the client specifies support for the OCSP responses in the TLS handshake.

GSK_SERVER_TLS_KEY_SHARES
Specifies the list of key share groups that are supported by the server during a TLS V1.3 handshake.

GSK_SESSION_TICKET_CLIENT_ENABLE
Specifies if the client supports caching session tickets received from a server after a TLS V1.3 handshake has completed and if it supports TLS V1.3 resumption attempts to the server.

GSK_SESSION_TICKET_CLIENT_MAXCACHED
Specifies the maximum of session tickets that are allowed to be cached by the client for a TLS V1.3 session.

GSK_SESSION_TICKET_CLIENT_MAXSIZE
Specifies the maximum size in bytes of a session ticket that is allowed to be stored in the client session ticket cache.

GSK_SESSION_TICKET_SERVER_ALGORITHM
Specifies the algorithm to be used by the server to encrypt and decrypt the session tickets used for TLS V1.3 session resumption.

GSK_SESSION_TICKET_SERVER_COUNT
Specifies the number of session tickets that will be sent by the server after the initial TLS V1.3 handshake has completed.

GSK_SESSION_TICKET_SERVER_ENABLE
Specifies if the server supports sending session tickets after a TLS V1.3 handshake has completed and if it will accept resumption attempts from the client.

GSK_SESSION_TICKET_SERVER_KEY_REFRESH
Returns the key refresh interval of the encryption key used by the server to encrypt session tickets, in seconds.

GSK_SESSION_TICKET_SERVER_TIMEOUT
Specifies the maximum time that a server accepts a TLS V1.3 session resumption request from the client measured in seconds from the initial handshake.

GSK_SUITE_B_PROFILE
Specifies the Suite B profile to be applied to TLS V1.2 sessions.

GSK_SYSPLEX_SESSION_TICKET_CACHE
Specifies whether sysplex session ticket caching for TLS V1.3 server sessions is supported.

GSK_SYSPLEX_SIDCACHE
Specifies whether sysplex session caching for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 server sessions is supported.

GSK_TLS_CBC_PROTECTION_METHOD
Specifies an optional SSL V3.0 or TLS V1.0 CBC IV protection method when writing application data.

GSK_TLS_CERT_SIG_ALG_PAIRS
Specifies the list of TLS V1.2 and TLS V1.3 hash and signature algorithm pair specifications that are supported by a TLS V1.2 or TLS V1.3 client or a TLS V1.3 server for use in digital signatures of X.509 certificates.

GSK_TLS_SIG_ALG_PAIRS
Specifies the list of TLS V1.2 and TLS V1.3 hash and signature algorithm pair specifications that are supported by the client or server for use in digital signatures of X.509 certificates and TLS handshake messages.

GSK_V2_CIPHER_SPECS
Specifies the SSL V2 cipher specifications in order of preference.

GSK_V2_SESSION_TIMEOUT
Specifies the session timeout value in seconds for the SSL V2 protocol.

GSK_V2_SIDCACHE_SIZE
Specifies the number of session identifiers that can be contained in the SSL V2 cache.

GSK_V3_CIPHER_SPECS
Specifies the SSL V3 cipher specifications in order of preference (2-character values).

GSK_V3_CIPHER_SPECS_EXPANDED
Specifies the SSL V3 cipher specifications in order of preference (4-character values).

GSK_V3_SESSION_TIMEOUT
Specifies the session timeout value in seconds for the SSL V3 to TLS V1.2 session identifiers and TLS V1.3 session tickets in the cache.

GSK_V3_SIDCACHE_SIZE
Specifies the size in number of entries in the SSL V3 to TLS V1.2 session identifier and TLS V1.3 session ticket cache.

GSK_WILDCARD_VALIDATION_ENABLE
Specifies whether the asterisk will be accepted as the wildcard character within the first label of the server certificate's subject alternative name DNS or subject DN common name to replace zero or more values.

Related topics

- "gsk_environment_init()" on page 134
- "gsk_environment_close()" on page 133

gsk_free_cert_data()

Releases the storage allocated for a certificate data array.

Format

```
#include <gskssl.h>

void gsk_free_cert_data (
                          gsk_cert_data_elem *     cert_data,
                          int                      elem_count)
```

Parameters

cert_data
Specifies the certificate data array to be released.

elem_count
Specifies the number of elements in the certificate data array.

Usage

The **gsk_free_cert_data()** routine releases the storage allocated for an array of certificate data elements.

Related topics

- "gsk_attribute_get_cert_info()" on page 81
- "gsk_get_cert_by_label()" on page 148

gsk_get_all_cipher_suites()

Returns the available SSL cipher suites.

Format

```
#include <gskssl.h>

gsk_status gsk_get_all_cipher_suites (
                                        gsk_all_cipher_suites *     cipher_suites)
```

Parameters

cipher_suites
Returns the runtime version, release, security level, and cipher suites.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it is one of the return codes listed in the **gskssl.h** include file. This is a possible error:

[GSK_ERR_STRUCTURE_TOO_SMALL]
Size that is specified for supplied structure is too small.

Usage

The **gsk_get_all_cipher_suites()** routine returns the System SSL runtime version, release, security level, and available cipher suites.

See Table 15 on page 146 for the version value and name identifiers that represent the level of the System SSL runtime. The version name is the name for that version. The current System SSL runtime is Version 4 Release 5.

Table 15. Version value and name

Version value	Version name
1	OS/390 Version 1
2	OS/390 Version 2
3	z/OS Version 1
4	z/OS Version 2

The security level equates to the GSK_SECURITY_LEVEL and determines whether strong cryptographic ciphers are supported. When set to GSK_SECURITY_LEVEL_EXPORT (1001), ciphers 56-bits and weaker are supported. When set to GSK_SECURITY_LEVEL_US (1000), all ciphers that are provided by System SSL are supported. See Appendix C, “Cipher suite definitions,” on page 795 for more information about supported ciphers.

The cipher suites are static null-terminated character strings that must not be modified or freed by the application. The available cipher suites for protocols SSL V3.0, TLS V1.0, and higher are returned in both 2-character and 4-character formats.

The cipher lists include all supported ciphers for the returned security level. As new ciphers are added, the lists are modified to contain the newly added ciphers. The adding of ciphers may cause cipher selection to be modified as new ciphers are added, and different ciphers to be selected if the lists are being used as the cipher list strings.

If executing in FIPS mode, the cipher suites are those that meet FIPS 140-2 criteria. For more information about the FIPS cipher suites, see “gsk_environment_open()” on page 136.

The application must initialize the size field in the gsk_all_cipher_suites structure to the size of the gsk_all_ciphers_suites structure before using this function.

gsk_get_cert_by_label()

Gets certificate information for a record label.

Format

```
#include <gskssl.h>

gsk_status gsk_get_cert_by_label (
                                  gsk_handle                 ssl_handle,
                                  const char *               record_label,
                                  gsk_cert_data_elem **      cert_data,
                                  int *                      elem_count)
```

Parameters

ssl_handle
Specifies an SSL environment handle returned by **gsk_environment_open()** or an SSL connection handle returned by **gsk_secure_socket_open()**.

record_label
Specifies the record label for the certificate.

cert_data
Returns the certificate data array. The **gsk_free_cert_data()** routine should be called to release the array when the certificate information is no longer needed.

elem_count
Returns the number of elements in the array of gsk_cert_data_elem structures.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ERR_ASN]
Unable to decode certificate.

[GSK_ERR_MULTIPLE_LABEL]
Multiple certificates exist for label.

[GSK_INSUFFICIENT_STORAGE]
Insufficient storage is available.

[GSK_INVALID_HANDLE]
The handle is not valid.

[GSK_KEY_LABEL_NOT_FOUND]
The key record is not found.

Usage

The **gsk_get_cert_by_label()** routine returns certificate information for a record label. The supplied handle can be for an SSL environment or an SSL connection.

Each element of the certificate data array has an element identifier. The element identifiers used for a particular certificate depends upon the contents of the certificate. These element identifiers are currently provided:

CERT_BODY_BASE64
Certificate body in Base64-encoded format

CERT_BODY_DER
Certificate body in binary ASN.1 DER-encoded format

CERT_COMMON_NAME
Subject common name (CN)

CERT_COUNTRY
Subject country (C)

CERT_DN_DER
Subject distinguished name in binary ASN.1 DER-encoded format

CERT_DN_PRINTABLE
Subject distinguished name as a printable character string

These DN attribute names are recognized by the System SSL run time.

- C - Country
- CN - Common name
- DC - Domain component
- DNQUALIFIER - Distinguished name qualifier
- EMAIL - email address
- GENERATIONQUALIFIER - Generation qualifier
- GIVENNAME - Given name
- INITIALS - Initials
- L - Locality
- MAIL - RFC 822 style address
- NAME - Name
- O - Organization name
- OU - Organizational unit name
- PC - Postal code
- SERIALNUMBER - Serial number
- SN - Surname
- ST - State or province
- STREET - Street
- T - Title

CERT_DNQUALIFIER
Subject distinguished name qualifier (DNQUALIFIER)

CERT_DOMAIN_COMPONENT
Subject domain component (DC)

CERT_EMAIL
Subject email address (EMAIL)

CERT_GENERATIONQUALIFIER
Subject generation qualifier (GENERATIONQUALIFIER)

CERT_GIVENNAME
Subject given name (GIVENNAME)

CERT_INITIALS
Subject initials (INITIALS)

CERT_ISSUER_COMMON_NAME
Issuer common name (CN)

CERT_ISSUER_COUNTRY
Issuer country (C)

CERT_ISSUER_DN_DER
Issuer distinguished name in binary ASN.1 DER-encoded format

CERT_ISSUER_DN_PRINTABLE
Issuer distinguished name as a printable character string

These DN attribute names are recognized by the System SSL run time.

- C - Country
- CN - Common name
- DC - Domain component
- DNQUALIFIER - Distinguished name qualifier
- EMAIL - email address
- GENERATIONQUALIFIER - Generation qualifier
- GIVENNAME - Given name
- INITIALS - Initials
- L - Locality
- MAIL - RFC 822 style address
- NAME - Name
- O - Organization name
- OU - Organizational unit name
- PC - Postal code
- SERIALNUMBER - Serial number
- SN - Surname
- ST - State or province
- STREET - Street
- T - Title

CERT_ISSUER_DNQUALIFIER
Issuer distinguished name qualifier (DNQUALIFIER)

CERT_ISSUER_DOMAIN_COMPONENT
Issuer domain component (DC)

CERT_ISSUER_EMAIL
Issuer email address (EMAIL)

CERT_ISSUER_GENERATIONQUALIFIER
Issuer generation qualifier (GENERATIONQUALIFIER)

CERT_ISSUER_GIVENNAME
Issuer given name (GIVENNAME)

CERT_ISSUER_INITIALS
Issuer initials (INITIALS)

CERT_ISSUER_LOCALITY
Issuer locality (L)

CERT_ISSUER_MAIL
Issuer RFC 822 style address (MAIL)

CERT_ISSUER_NAME
Issuer name (NAME)

CERT_ISSUER_ORG
Issuer organization (O)

CERT_ISSUER_ORG_UNIT
Issuer organizational unit (OU)

CERT_ISSUER_POSTAL_CODE
Issuer postal code (PC)

CERT_ISSUER_SERIALNUMBER
Issuer serial number (SERIALNUMBER)

CERT_ISSUER_STATE_OR_PROVINCE
Issuer state or province (ST)

CERT_ISSUER_STREET
Issuer street (STREET)

CERT_ISSUER_SURNAME
Issuer surname (SN)

CERT_ISSUER_TITLE
Issuer title (T)

CERT_LOCALITY
Subject locality (L)

CERT_MAIL
Subject RFC 822 style address (MAIL)

CERT_NAME
Subject name (NAME)

CERT_ORG
Subject organization (O)

CERT_ORG_UNIT
Subject organizational unit (OU)

CERT_POSTAL_CODE
Subject postal code (PC)

CERT_SERIAL_NUMBER
Certificate serial number

CERT_SERIALNUMBER
Subject serial number (SERIALNUMBER)

CERT_STATE_OR_PROVINCE
Subject state or province (ST)

CERT_STREET
Subject street (STREET)

CERT_SURNAME
Subject surname (SN)

CERT_TITLE
Subject title (T)

The CERT_BODY_DER, CERT_BODY_BASE64, CERT_DN_DER, and CERT_ISSUER_DN_DER elements are not null-terminated and the 'cert_data_l' field must be used to get the element length. All of the other elements are null-terminated character strings and the 'cert_data_l' field is the length of the string excluding the string delimiter.

Related topics

- "gsk_environment_init()" on page 134
- "gsk_secure_socket_init()" on page 157

gsk_get_cipher_suites()

Returns the available SSL cipher suites.

Format

```
#include <gskssl.h>

void gsk_get_cipher_suites (
                             gsk_cipher_suites *        cipher_suites)
```

Parameters

cipher_suites
Returns the runtime version, release, security level, and cipher suites.

Usage

The **gsk_get_cipher_suites()** routine returns the System SSL runtime version, release, security level, and available default cipher suites.

See Table 16 on page 152 for the version value and name identifiers that represent the level of the System SSL runtime. The version name is the name for that version. The current System SSL runtime is Version 4 Release 5.

Table 16. Version value and name

Version value	**Version name**
1	OS/390 Version 1
2	OS/390 Version 2
3	z/OS Version 1
4	z/OS Version 2

The security level equates to the GSK_SECURITY_LEVEL and determines whether strong cryptographic ciphers are supported. When set to GSK_SECURITY_LEVEL_EXPORT (1001), ciphers 56-bits and weaker are available. When set to GSK_SECURITY_LEVEL_US (1000), any strength cipher is available. See Appendix C, “Cipher suite definitions,” on page 795 for more information about supported ciphers.

The cipher suites are static null-terminated character strings which must not be modified or freed by the application. The cipher lists include the available default ciphers for the returned security level.

If executing in FIPS mode, the cipher suites are those that meet FIPS 140-2 criteria. For more information about the FIPS cipher suites, see “gsk_environment_open()” on page 136.

gsk_get_ssl_vector()

Obtain the address of the Secure Socket Layer function vector.

Format

```
#include <gskssl.h>

void gsk_get_ssl_vector (
                             gsk_uint32 *          function_mask,
                             gsk_ssl_vector **     function_vector)
```

Parameters

function_mask
Returns a bit mask indicating the Secure Socket Layer level.

function_vector
Returns the address of the Secure Socket Layer function vector.

Usage

The Secure Socket Layer (SSL) functions can be called using either static binding or runtime binding. Static binding is performed when the application is compiled while runtime binding is performed when the application is run.

In order to use static binding, the SSL sidedeck file is specified as input to the binder. This causes all SSL functions to be resolved at bind time and causes the SSL DLL to be implicitly loaded when the application is run.

In order to use runtime binding, the SSL DLL must be explicitly loaded by the application and the SSL functions must be called using indirect addresses. The **gsk_get_ssl_vector()** routine allows an application to obtain the address of the SSL function vector containing an entry for each SSL API routine. This eliminates the need for the application to build the function vector through repeated calls to the **dllqueryfn()** routine.

The function mask indicates the capabilities of the SSL DLL. These values are defined:

GSKSSL_API_LVL1
SSL functions provided as part of z/OS Version 1 Release 6 are available.

GSKSSL_API_LVL2
SSL functions provided as part of z/OS Version 1 Release 11 are available.

GSKSSL_API_LVL3
SSL functions provided as part of z/OS Version 1 Release 13 are available.

gsk_get_update()

Checks for a key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token update.

Format

```
#include <gskssl.h>

gsk_status gsk_get_update (
                            gsk_handle     env_handle,
                            long *         update_flag)
```

Parameters

env_handle
Specifies the SSL environment handle returned by the **gsk_environment_open()** routine.

update_flag
Returns 1 if the key database, PKCS #12 file, SAF key ring or z/OS PKCS #11 token has been updated or 0 if it has not been updated.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ERR_PERMISSION_DENIED]
Not authorized to access key database, PKCS #12 file, SAF key ring or z/OS PKCS #11 token.

[GSK_INVALID_HANDLE]
The environment handle is not valid.

[GSK_INVALID_STATE]
The environment is not in the initialized state.

[GSK_KEYRING_OPEN_ERROR]
Unable to open the key database, PKCS #12 file, key ring or token.

Usage

The **gsk_get_update()** routine tests if the key database, PKCS #12 file, SAF key ring or z/OS PKCS #11 token associated with the SSL environment has been updated since the last time that **gsk_get_update()** was called or since the environment was initialized if **gsk_get_update()** has not been called yet. If an update has occurred, the application can close the current environment and then create a new environment to pick up the updates.

Related topics

- "gsk_environment_open()" on page 136

gsk_list_free()

Releases storage allocated for a list.

Format

```
#include <gskssl.h>

void gsk_list_free (
                     gsk_list *     list)
```

Parameters

list
Specifies the list to be released.

Usage

The **gsk_list_free()** routine releases storage allocated for a list. This includes the gsk_list structure itself and all gsk_list structures anchored by the structure passed on the function call.

Related topics

- "gsk_attribute_get_data()" on page 85

gsk_secure_socket_close()

Closes a secure socket connection.

Format

```
#include <gskssl.h>

gsk_status gsk_secure_socket_close (
                                    gsk_handle *      soc_handle)
```

Parameters

soc_handle
Specifies the connection handle returned by the **gsk_secure_socket_open()** routine. The connection handle will be set to NULL upon completion.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_CONNECTION_ACTIVE]
The connection has an active read or write request.

[GSK_INVALID_HANDLE]
The connection handle is not valid.

[GSK_WOULD_BLOCK_WRITE]
An attempt to write pending data failed with EWOULDBLOCK.

Usage

The **gsk_secure_socket_close()** routine closes a secure socket connection created by the **gsk_secure_socket_open()** routine. The socket itself is not closed (the application is responsible for closing the socket). The connection can no longer be used for secure communications after calling the **gsk_secure_socket_close()** routine.

The **gsk_secure_socket_close()** routine can return GSK_WOULD_BLOCK_WRITE if the socket is in non-blocking mode and there is pending write data. The connection is not closed in this case and the application should call **gsk_secure_socket_close()** again when the socket is ready to accept a write request.

Be sure **gsk_secure_socket_shutdown()** call is issued before a **gsk_secure_socket_close()** call.

Related topics

- "gsk_secure_socket_open()" on page 173
- "gsk_secure_socket_init()" on page 157
- "gsk_secure_socket_read()" on page 174
- "gsk_secure_socket_write()" on page 179

gsk_secure_socket_init()

Initializes a secure socket connection.

Format

```
#include <gskssl.h>

gsk_status gsk_secure_socket_init(
                                  gsk_handle     soc_handle)
```

Parameters

soc_handle
Specifies the connection handle that is returned by the **gsk_secure_socket_open()** routine.

Results

The function return value is 0 (**GSK_OK**) if no error is detected. Otherwise, it is one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_CERTIFICATE_NOT_AVAILABLE]
No certificates available.

[GSK_ERR_3DES_KEYS_NOT_UNIQUE]
Triple DES key parts are not unique.

[GSK_ERR_ALERT_NOT_YET_SUPPORTED]
Alert received from remote partner is allowed by protocol, but not expected by System SSL.

[GSK_ERR_BAD_CERT]
Certificate is not valid.

[GSK_ERR_BAD_CERTIFICATE_STATUS_RESPONSE]
Certificate status response is not valid.

[GSK_ERR_BAD_DATE]
Certificate is not valid yet or is expired.

[GSK_ERR_BAD_EC_PARAMS]
EC parameters are not supplied.

[GSK_ERR_BAD_HTTP_RESPONSE]
HTTP response is not valid.

[GSK_ERR_BAD_KEY_LABEL_LIST]
Key label list is not valid.

[GSK_ERR_BAD_KEY_SHARE_LIST]
Key share list is not valid.

[GSK_ERR_BAD_KEYFILE_LABEL]
The specified key is not found in the key database or the key is not trusted.

[GSK_ERR_BAD_MAC]
Message verification failed.

[GSK_ERR_BAD_MESSAGE]
Incorrectly formatted message that is received from peer application.

[GSK_ERR_BAD_MSG_LEN]
Incorrectly formatted TLS extension data that is contained within message that is received from peer application.

[GSK_ERR_BAD_OCSP_RESPONSE]
OCSP response is not valid.

[GSK_ERR_BAD_PEER]
Peer application has violated the SSL protocol.

[GSK_ERR_BAD_PEER_ID]
Specified peer identifier is not valid.

[GSK_ERR_BAD_PSK_EXCHANGE_MODES]
PSK exchange modes extension from the remote partner does not contain a supported value.

[GSK_ERR_BAD_PSK_IDENTITY_VALUE]
Remote partner indicates a bad PSK identity value.

[GSK_ERR_BAD_SID_VALUE]
Specified session identifier is not valid.

[GSK_ERR_BAD_SIG_ALG_PAIR]
Signature algorithm pairs list is not valid.

[GSK_ERR_BAD_V2_CIPHER]
SSL V2 cipher is not valid.

[GSK_ERR_BAD_V3_CIPHER]
SSL V3 cipher is not valid.

[GSK_ERR_BAD_V3_EXPANDED_CIPHER]
SSL V3 expanded cipher is not valid.

[GSK_ERR_CERT_VALIDATION]
Certificate validation error.

[GSK_ERR_CERTIFICATE_REVOKED]
Peer certificate is revoked.

[GSK_ERR_CLIENT_AND_SERVER_SID_DO_NOT_MATCH]
Client SID does not match the server SID.

[GSK_ERR_CRYPTO]
Cryptographic error is detected.

[GSK_ERR_EC_PARAMETERS_NOT_SUPPLIED]
EC parameters are not supplied.

[GSK_ERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve is not supported in FIPS mode.

[GSK_ERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[GSK_ERR_FALLBACK_ATTEMPT_DETECTED]
Client has detected that the server has attempted an unexpected protocol fallback.

[GSK_ERR_HOSTNAME_NOT_VALID]
HTTP server hostname is not valid.

[GSK_ERR_HTTP_IO_ERROR]
HTTP server communication error.

[GSK_ERR_HTTP_RESPONSE_TIMEOUT]
HTTP response is not received within configured time limit.

[GSK_ERR_ICSF_CLEAR_KEY_SUPPORT_NOT_AVAILABLE]
Clear key support not available due to ICSF key policy.

[GSK_ERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[GSK_ERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[GSK_ERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[GSK_ERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[GSK_ERR_INCOMPATIBLE_KEY]
Certificate key is not compatible with cipher suite.

[GSK_ERR_INCORRECT_EXT_KEY_USAGE]
Extended key usage certificate extension does not permit the requested purpose.

[GSK_ERR_INCORRECT_KEY_ATTRIBUTE]
TKDS Private Key attributes do not support digital signature or RSA operation.

[GSK_ERR_INVALID_COMPRESSION_FIELD]
Legacy compression field must be a single byte set to 0.

[GSK_ERR_INVALID_FRAGMENT_LENGTH]
An unsupported fragment length was received.

[GSK_ERR_INVALID_SIGALG_USED]
Signature algorithm used by the remote partner for a secure connection is not correct.

[GSK_ERR_IO]
I/O error communicating with peer application.

[GSK_ERR_KEY_IS_SMALLER_THAN_MINIMUM]
Peer's end-entity certificate key size is smaller than the minimum size allowed.

[GSK_ERR_KEY_SHARES_LARGER_THAN_ECCURVE]
Key share list cannot contain more groups than the elliptic curve list.

[GSK_ERR_LDAP]
An LDAP error is detected.

[GSK_ERR_LDAP_NOT_AVAILABLE]
The LDAP server is not available.

[GSK_ERR_LDAP_RESPONSE_TIMEOUT]
LDAP response is not received within configured time limit.

[GSK_ERR_MAX_RESPONSE_SIZE_EXCEEDED]
Maximum response size exceeded.

[GSK_ERR_MISMATCH_ECCURVE_KEY_SHARES]
Mismatch between elliptic curve and key share lists.

[GSK_ERR_MISMATCH_TLS_EXT_MASTER_SECRET]
Extended master secret extension mismatch detected on cached TLS handshake attempt.

[GSK_ERR_MISSING_KEY_ALGORITHM]
Certificate key algorithm is not in signature algorithm pairs list.

[GSK_ERR_MISSING_REQUEST_SIGALGS]
Missing required certificate request signature algorithms.

[GSK_ERR_MISSING_SIGNATURE_ALGORITHM]
Signature algorithm is not in signature algorithm pairs list.

[GSK_ERR_MISSING_TLS_EXTENSION]
Required TLS extension is missing from remote partner.

[GSK_ERR_MULTIPLE_DEFAULT]
Multiple keys are marked as the default.

[GSK_ERR_MULTIPLE_LABEL]
Multiple certificates exist for label.

[GSK_ERR_NO_CACHE_SESSION]
No cached session entry exists.

[GSK_ERR_NO_CERTIFICATE]
No certificate received from partner.

[GSK_ERR_NO_CIPHERS]
No cipher specifications.

[GSK_ERR_NO_COMMON_KEY_SHARES]
No key share groups in common with partner.

[GSK_ERR_NO_KEY_LABEL_LIST_MATCH]
There are no non-expired or valid key labels found that support the protocol, cipher, and session attributes.

[GSK_ERR_NO_PRIVATE_KEY]
Certificate does not contain a private key or the private key is unusable.

[GSK_ERR_NON_SUITE_B_CERTIFICATE]
Certificate does not meet Suite B requirements.

[GSK_ERR_OCSP_EXPIRED]
OCSP response is expired.

[GSK_ERR_OCSP_NONCE_CHECK_FAILED]
Nonce in OCSP response does not match value in OCSP request.

[GSK_ERR_OCSP_REQ_ERROR]
Error creating OCSP request.

[GSK_ERR_OCSP_RESPONDER_ERROR]
OCSP request failed with internal responder error.

[GSK_ERR_OCSP_RESPONSE_DUPLICATE_FOUND]
OCSP response contains duplicate certificate statuses.

[GSK_ERR_OCSP_RESPONSE_NOT_FOUND]
OCSP response does not contain requested certificate status.

[GSK_ERR_OCSP_RESPONSE_SIGALG_NOT_VALID]
OCSP response signature algorithm is not in the signature algorithm pairs list.

[GSK_ERR_OCSP_RESPONSE_TIMEOUT]
OCSP response was not received within configured time limit.

[GSK_ERR_OCSP_SIG_REQUIRED]
OCSP responder requires a signed request.

[GSK_ERR_PEER_ID_ATTRIBUTES_CONFLICT]
Client session cache attributes do not agree.

[GSK_ERR_REMOTE_BAD_CERTIFICATE]
Remote partner indicates sent certificate is not valid.

[GSK_ERR_REMOTE_BAD_KEY_SHARE_LIST]
Key share list received from remote partner is not correct.

[GSK_ERR_REMOTE_BAD_MESSAGE]
Remote partner indicates sent TLS message is not valid.

[GSK_ERR_REMOTE_HANDSHAKE_FAILURE]
Remote partner indicates a handshake failure.

[GSK_ERR_REMOTE_MISSING_TLS_EXTENSION]
Remote partner indicates a required TLS extension is missing.

[GSK_ERR_REMOTE_UNSUPPORTED_CERTIFICATE]
Remote partner indicates unsupported certificate.

[GSK_ERR_REQ_CERT_BC_EXT_MISSING]
The required basic constraints certificate extension is missing.

[GSK_ERR_REQUIRED_4CHAR_CIPHERS]
4-character cipher specifications are required.

[GSK_ERR_REQUIRED_CIPHERS_NOT_SPECIFIED]
Required cipher specifications have not been specified.

[GSK_ERR_REQUIRED_TLS_EXT_MASTER_SECRET]
Missing required TLS extended master secret extension from the remote partner.

[GSK_ERR_RESUMPTION_NOT_VALID]
Resumption attempt is not valid.

[GSK_ERR_REVINFO_NOT_YET_VALID]
Revocation information is not yet valid.

[GSK_ERR_RNG]
Error encountered when generating random bytes.

[GSK_ERR_SECURE_LABEL_OPERATION_UNSUPPORTED]
A secure private key cannot be used with a fixed ECDH key exchange.

[GSK_ERR_SELF_SIGNED]
A self-signed certificate cannot be validated.

[GSK_ERR_SERVER_REF_ID_NO_MATCH]
Server certificate validation failed with provided reference ID list.

[GSK_ERR_SESSION_ID_NOT_VALID]
Session ID received from remote partner is not correct.

[GSK_ERR_SIGNATURE_NOT_SUPPLIED]
Signature not supplied.

[GSK_ERR_SOCKET_CLOSED]
Socket connection closed by peer application.

[GSK_ERR_TLS_EXTENSION_NOT_ALLOWED]
TLS handshake message from the remote partner included an extension that is not allowed or supported.

[GSK_ERR_UNKNOWN_CA]
A certification authority certificate is missing.

[GSK_ERR_UNRECOGNIZED_NAME]
The requested server name is not recognized.

[GSK_ERR_UNSUPPORTED]
SSL protocol or certificate type is not supported.

[GSK_ERR_UNSUPPORTED_CERTIFICATE_TYPE]
The certificate type is not supported by System SSL.

[GSK_ERR_UNSUPPORTED_REQUIRED_EXTENSION]
A required TLS extension has been rejected.

[GSK_ERR_UNSUPPORTED_EXTENSION]
An unrequested TLS Extension has been encountered.

[GSK_ERR_INAPPROPRIATE_PROTOCOL_FALLBACK]
An inappropriate protocol fallback is detected.

[GSK_INSUFFICIENT_STORAGE]
Insufficient storage is available.

[GSK_INVALID_HANDLE]
The connection handle is not valid.

[GSK_INVALID_STATE]
The connection is not in the open state or a previous initialization request has failed.

[GSK_RSA_TEMP_KEY_PAIR]
Unable to generate temporary RSA public/private key pair.

[GSK_WOULD_BLOCK_READ]
An attempt to read a handshake message failed with EWOULDBLOCK.

[GSK_WOULD_BLOCK_WRITE]
An attempt to write a handshake message failed with EWOULDBLOCK.

Usage

The **gsk_secure_socket_init()** routine initializes a secure socket connection created by the **gsk_secure_socket_open()** routine. After the connection has been initialized, it can be used for secure data transmission using the **gsk_secure_socket_read()** and **gsk_secure_socket_write()** routines. The

gsk_secure_socket_close() routine should be called to close the connection when it is no longer needed. The **gsk_secure_socket_close()** routine should also be called if an error is returned by the **gsk_secure_socket_init()** routine.

Before calling the **gsk_secure_socket_init()** routine, the application must create a connected socket and store the socket descriptor in the SSL connection by calling the **gsk_attribute_set_numeric_value()** routine. For a client, this means calling the **socket()** and **connect()** routines. For a server, this means calling the **socket()**, **listen()**, and **accept()** routines. However, SSL does not require the use of TCP/IP for the communications layer. The socket descriptor can be any integer value which is meaningful to the application. The application must provide its own socket routines if it is not using TCP/IP by calling the **gsk_attribute_set_callback()** routine.

The **gsk_secure_socket_init()** routine can return GSK_WOULD_BLOCK_READ or GSK_WOULD_BLOCK_WRITE if the socket is in non-blocking mode. The connection is not initialized in this case and the application must call **gsk_secure_socket_init()** again when the socket is ready to accept a read request (GSK_WOULD_BLOCK_READ) or a write request (GSK_WOULD_BLOCK_WRITE). The application must provide its own callback routine for **io_setsocketoptions()** to have the SSL handshake processed in non-blocking mode (the default **io_setsocketoptions()** routine places the socket into blocking mode during the handshake processing).

In FIPS mode, only DSA certificates with domain parameters that conform to FIPS 186-4: Digital Signature Standard (DSS) (nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf) are supported. In non-FIPS mode, if the key size is less than 1024 bits, then domain parameters that conform to FIPS 186-2 are supported. In non-FIPS mode, if the key size is greater than or equal to 1024 bits, the domain parameters must conform to FIPS 186-4, with the exception that parameters that have a prime modulus (p) of 2048 bits and a prime divisor (q) of 160 bits are also tolerated.

Be sure a **gsk_secure_socket_shutdown()** call is issued before a **gsk_secure_socket_close()** call.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, "System SSL and FIPS 140-2," on page 19.

Protocol Selection

An SSL handshake is performed as part of the processing of the **gsk_secure_socket_init()** routine. This establishes the server identity and optionally the client identity. It also negotiates the cryptographic parameters to be used for the connection. The client and server attempts to use the highest available protocol version as determined by the intersection of the enabled protocol versions for the client and the server and the compatible ciphers. Thus:

- TLS V1.3 is used if it is enabled on both the client and the server.
- If TLS V1.3 cannot be used and TLS V1.2 is enabled, negotiations drop back to TLS V1.2.
- If TLS V1.2 cannot be used and TLS V1.1 is enabled, negotiations drop back to TLS V1.1.
- If TLS V1.1 cannot be used and TLS V1.0 is enabled, negotiations drop back to TLS V1.0.
- If TLS V1.0 cannot be used and SSL V3 is enabled, negotiations drop back to SSL V3.
- If SSL V3 cannot be used, TLS V1.2 or TLS V1.3 were not enabled on the client or server, and SSL V2 is enabled, negotiations drop back to SSL V2.

Notes:

1. SSL V2 and SSL V3 are not as secure as TLS and should be disabled whenever possible to avoid attacks that force the client and server to drop back to SSL V2 or SSL V3 even though they are capable of using TLS V1.0, TLS V1.1, or TLS V1.2.
2. When TLS extensions are defined for a client and any of the TLS protocols are enabled for the connection, SSL V2 is not negotiated even if it is enabled.
3. If TLS V1.2 is enabled on the client, establishment of SSL sessions with SSL V2 servers is not supported.
4. If TLS V1.3 is enabled, SSL V2 and SSL V3 are not supported even when enabled.

5. If TLS V1.3 is enabled, TLS V1.0 is not enabled by default. TLS V1.0 must be explicitly enabled when TLS V1.3 is enabled.
6. It is recommended that there are no gaps in enabled protocols when TLS V1.3 is enabled on the client side. If TLS V1.3 and TLS V1.1 are enabled on the client side, it is recommended that TLS V1.2 is also enabled.

Cipher selection

The client sends a list of ciphers it supports during the SSL handshake. The server application uses this list, and the defined ciphers that are supported by the server, to determine the cipher to be used during the SSL handshake. If the client is operating in FIPS mode, then the list provided only contains FIPS ciphers. A server executing in FIPS mode will only use FIPS ciphers. The cipher selection is done by looking through the servers cipher list for a match in the clients list. The first matching cipher is used.

When building the server's list of cipher suites for comparison with the list sent by the client, the server might omit some ciphers from the list as follows:

- When executing in an export level cryptographic environment, any ciphers that are not permitted for use in an export level environment.
- When executing in FIPS mode, any cipher suites that are not valid for use in FIPS mode.
- Any cipher suites that specify a *key algorithm* that is not supported for use with the server certificate's key. For example, if the cipher requires an RSA key algorithm but the server certificate uses a DSA key algorithm.
- When using protocol SSL V3.0 or earlier, any cipher suites that specify Elliptic Curve Cryptography.
- When using protocol TLS V1.1 or earlier, any cipher suites that specify:
  - A *sign key algorithm* that is not supported for use with the server certificate's key. For example, if the cipher requires a Diffie-Hellman certificate signed with an RSA signature, but the server certificate is a Diffie-Hellman certificate that is signed with a DSA signature.
  - SHA-2 message authentication.
  - AES-GCM encryption.
- When using protocol TLS V1.0, TLS V1.1, or TLS V1.2, any ephemeral elliptic curve (ECDHE) cipher suite is ignored if the client and server do not have a supported elliptic curve in common. The client provides its list of supported elliptic curves as part of the TLS handshake. If the server's certificate is ECDSA and does not match one of the client’s elliptic curves or the server does not support at least one curve supported by the client, ECDHE cipher suites cannot be used. A z/OS System SSL client specifies the list of supported elliptic curves through the GSK_CLIENT_ECURVE_LIST setting. An empty list means all elliptic curves supported can be used. The FIPS setting may further restrict what curves can be used. A z/OS System SSL server specifies the list of supported elliptic curves through the GSK_SERVER_ALLOWED_KEX_CURVES setting. See Table 29 on page 804 for the supported elliptic curve (group) definitions for TLS V1.0, TLS V1.1, and TLS V1.2.
- When using protocol TLS V1.1 and higher, any cipher suites that specify 40-bit export encryption.
- When using protocol TLS V1.2, any cipher suites that specify 56-bit DES encryption.
- When using protocol TLS V1.2, a key algorithm that is not specified in the signature algorithms pairs that is supplied by the client.
- When using protocol TLS V1.3, any ciphers that are supported by earlier protocols versions.

Note: For protocols TLS V1.1 and TLS V1.2, export cipher suites cannot be used. 40-bit ciphers is ignored if TLS V1.1 or higher is negotiated as the security protocol. If TLS V1.1 or TLS V1.2 is the intended protocol and only 40-bit ciphers are available, the connection fails with GSK_ERR_NO_CIPHERS.

If the client is enabled for TLS V1.3, the following checks are performed:

- There must be at least one valid TLS V1.3 cipher suite specified.
- If the client is enabled for TLS V1.0, TLS V1.1, or TLS V1.2 protocols, there must be at least one non-TLS V1.3 cipher suite specified.

Server certificate

If the selected protocol is TLS V1.2 and earlier, the server certificate can use either RSA, DSA, Diffie-Hellman, or ECC as the public/private key algorithm. If the selected protocol is TLS V1.3, the server certificate can use either RSA or ECC as the public/private key algorithm. DSA, Diffie-Hellman, and Brainpool certificates are not supported with TLS V1.3.

In FIPS mode, the RSA or DSA key size must be at least 1024 bits, the Diffie-Hellman key size must be at least 2048 bits, and the ECC key size must be at least 192 bits and use a NIST-approved named curve.

If the selected protocol is TLS V1.3, the RSA key size must be at least 2048 bits and the ECC key size must be at least 256 bits. DSA, Diffie-Hellman, and Brainpool certificates are not supported with TLS V1.3.

If the selected protocol is TLS V1.2 or earlier, the following rules apply for certificates and key exchanges:

- An RSA certificate can be used with an RSA, ephemeral Diffie-Hellman, or ephemeral ECDH key exchange.
- A DSA certificate can be used with an ephemeral Diffie-Hellman key exchange.
- A Diffie-Hellman certificate can be used in a fixed Diffie-Hellman key exchange.
- An ECC certificate can be used with a fixed ECDH or ephemeral ECDH key exchange.

If the selected protocol is TLS V1.3, the key exchange mechanism that is used is not tied to the certificate. Therefore, the above rules do not apply to TLS V1.3.

If the selected protocol is TLS V1.2 or earlier and the server's certificate contains a key usage extension during the SSL/TLS handshake, it must allow key usage as follows:

- RSA certificates using export restricted ciphers (40-bit RC4 encryption and 40-bit RC2 encryption) with a public key size greater than 512 bits must allow digital signature. If operating in FIPS mode, export restricted ciphers cannot be selected.
- Diffie-Hellman certificates that are used in fixed Diffie-Hellman key exchange must allow key agreement.
- ECC certificates that are used in fixed ECDH key exchange must allow key agreement.
- ECC certificates that are used in ephemeral ECDH key exchange must allow digital signature.
- RSA certificates that are used in ephemeral ECDH key exchange must allow digital signature.
- DSA certificates using ephemeral Diffie-Hellman key exchange must allow digital signature.
- Other RSA certificates must allow key encipherment.

If the selected protocol is TLS V1.3 and the server's certificate contains a key usage extension during the TLS handshake, it must allow digital signature. If the selected protocol is TLS V1.3 and the server's certificate contains an extended key usage extension during the TLS handshake, it must allow for server authentication.

System SSL does not accept VeriSign Global Server ID certificates. When specified, System SSL uses these certificates as any other certificate when determining the encryption cipher to be used for the SSL/TLS session.

When using TLS V1.2 as the SSL/TLS session protocol, the client may pass to the server a list of signature algorithm pairs as part of the SSL/TLS handshake. When using TLS V1.3 as the TLS session protocol, the client must pass to the server a list of signature algorithms pairs as part of the TLS handshake. The key algorithm and signature algorithm of the server certificate must be present in this list of signature algorithm pairs. In addition, any peer certificates in the server certificate chain must also be signed using a signature algorithm present in the list.

The signature algorithm pair list under the TLS V1.2 protocol may allow some TLS ciphers to operate using certificates that were previously incompatible with the cipher specification. In previous versions of TLS, these ciphers (primarily ciphers that use a fixed Diffie-Hellman or fixed ECDH key exchange) required the server certificate to be signed with a specific signature key algorithm. Under TLS V1.2, the signature algorithm pairs list allows the cipher to be used if the signature algorithm is specified in the list.

During the TLS handshake process, the client can be configured to perform additional domain name validation against the server's certificate. This validation involves comparing the server's domain name value against the values provided in the server's certificate. The server's certificate can present its domain name either through its subject alternative name DNS or subject DN common name. This validation is enabled through the GSK_REFERENCE_ID_DNS for server certificate's DNS entry for the subject alternative name comparisons and GSK_REFERENCE_ID_CN for subject DN common name comparisons. The values are fully qualified DNS domain name, case insensitive, that can be passed in a single value or list that is comma or space separated. If a value includes either a comma or a space, the character must be escaped with a backlash. If any of the values end in a period, the period will be removed prior to comparison, including any values provided from the server's certificate or the client reference list or lists.

Additionally, GSK_WILDCARD_VALIDATION_ENABLE can be set to GSK_WILDCARD_VALIDATION_ENABLE_ON to indicate that an asterisk as the wildcard character can be used to replace zero or more characters within the first label of the server certificate's DNS entry for the subject alternative name or subject DN common name values. For example, within the values, any characters up to the first period can be intermingled with the wildcard character to represent any characters where the wildcard is indicated. If there is a failure to validate the server's certificate with the reference list, a handshake failure occurs.

For more information on domain name comparison, see "Server certificate domain-based validation" on page 70.

Client certificate

The SSL server always provides its certificate to the SSL client as part of the handshake. The client always performs server authentication using the certificate that is provided by the server. Server authentication requires that the provided certificate be validated. The validation process involves building the partner's certificate chain, using trusted certificates from the trusted certificate store and untrusted certificates that are provided through the handshake messages, as well as checking for certificate revocation information (LDAP CRLs, HTTP CRLs, or OCSP responses). If the selected protocol is TLS V1.3, the certificate validation mode is set to RFC 5280 unless explicitly specified. See "gsk_validate_certificate_mode()" on page 491 for a description how the certificate and revocation information is provided through data sources and the steps that are performed during certificate validation. Depending upon the server handshake type, the server may ask the client to provide its certificate. The key label that is stored in the connection is used to retrieve the certificate from the key database, PKCS #12 file, key ring, or token. The default key is used if no label is set. The key record must contain both an X.509 certificate and a private key. See "gsk_validate_certificate_mode()" on page 491 for a description of the steps that are performed during certificate validation.

To gather failure or successful diagnostic information related to the validation of the peer certificate, see "Certificate diagnostic callback routine" on page 41.

If the selected protocol is TLS V1.3 and the client's certificate contains a key usage extension during the TLS handshake, it must allow digital signature. If the selected protocol is TLS V1.3 and the client's certificate contains an extended key usage extension during the TLS handshake, it must allow for client authentication. If the selected protocol is TLS V1.3, the RSA key size must be at least 2048 bits and the ECC key size must be at least 256 bits. DSA, Diffie-Hellman, and Brainpool certificates are not supported with TLS V1.3.

If the selected protocol is TLS V1.2 or earlier, the client certificate can use either RSA, Digital Signature Standard algorithm (DSA), ECC, or Diffie-Hellman as the public/private key algorithm. If the selected protocol is TLS V1.3, the client certificate can use either RSA or ECC as the public/private key algorithm. DSA, Diffie-Hellman, and Brainpool certificates are not supported with TLS V1.3.

If the selected protocol is TLS V1.2 or earlier, the type of client certificate that can be used depends on the key exchange method being used for the session cipher that is selected by the server, as detailed in the following list:

- RSA key exchange - RSA or DSA.
- Fixed Diffie-Hellman key exchange - Diffie-Hellman.
- Ephemeral Diffie-Hellman key exchange - RSA or DSA.
- Fixed ECDH key exchange - RSA, DSA, or ECC.
- Ephemeral ECDH key exchange - RSA, DSA, or ECC.

If the selected protocol is TLS V1.3, the key exchange mechanism that is used is not tied to the certificate. Therefore, the above rules do not apply to TLS V1.3.

Client certificates that are used in a fixed Diffie-Hellman or fixed ECDH key exchange when the selected protocol is TLS V1.2 or earlier where the client certificate is used to send the client's public key to the server must support key agreement. This means the certificate key usage extension (if any) must allow key agreement.

In all other cases, the client certificate must support digital signatures. This means the certificate key usage extension (if any) must allow digital signature.

When using TLS V1.2 as the SSL/TLS session protocol, the server may pass to the client a list of signature algorithm pairs as part of the SSL/TLS handshake. When using TLS V1.3 as the TLS session protocol, the server must pass to the client a list of signature algorithms pairs as part of the TLS handshake. The key algorithm and signature algorithm of the client certificate must be present in this list of signature algorithm pairs. In addition, any peer certificates in the client certificate chain must also be signed using a signature algorithm present in the list.

Key size and Diffie-Hellman group size support

When executing in non-FIPS mode and the selected protocol is TLS V1.2 or earlier, the default minimum supported key sizes are:

- Diffie-Hellman: 1024
- DSA: 1024
- ECC: 192
- RSA: 1024

The default minimum key size can be overridden using attributes:

- GSK_PEER_DH_MIN_KEY_SIZE
- GSK_PEER_DSA_MIN_KEY_SIZE
- GSK_PEER_ECC_MIN_KEY_SIZE
- GSK_PEER_RSA_MIN_KEY_SIZE

Setting any of these attributes to a larger key size than is required for a FIPS level means that the peer must provide larger key sizes than required for the security strength. If the key size is set smaller than the required security strength, the weaker key may be accepted during early handshake processing, but the connection will fail later due to weak security strength of the key.

If the selected protocol is TLS V1.3, the default minimum supported key sizes are:

- RSA: 2048
- ECC: 256

Note: If the selected protocol is TLS V1.3 and the RSA or ECC minimum key sizes are set to more stringent values, the more stringent values are used to check the peer’s certificate key size. The DSA and Diffie-Hellman minimum key sizes are ignored as DSA and Diffie-Hellman certificates are not supported if the selected protocol is TLS V1.3.

For ephemeral Diffie-Hellman key exchanges when the selected protocol is TLS V1.2 or earlier, the following attributes may be used to set minimum exchanged DHE group size:

GSK_CLIENT_EPHEMERAL_DH_GROUP_SIZE

- LEGACY uses group size 1024 in non-FIPS mode and group size 2048 in FIPS mode.
- 2048 always enforces group size 2048 regardless of FIPS mode.

GSK_SERVER_EPHEMERAL_DH_GROUP_SIZE

- LEGACY uses group size 1024 in non-FIPS mode and group size 2048 in FIPS mode.
- 2048 always enforces group size 2048 regardless of FIPS mode.

- MATCH uses the DSA or RSA key size to define the required size to generate the Diffie-Hellman parameters. In non-FIPS mode if the key being matched is less than or equal to key size 1024, the group size 1024 is used. If the key being matched is greater than 1024, the group size 2048 is used. If executing in FIPS mode, the group size 2048 is used.

Specifying multiple labels

A server application can utilize environment variable GSK_SERVER_KEY_LABEL_LIST or specify attribute GSK_SERVER_KEYRING_LABEL_LIST for the environment or connection to specify a list of 1 to 8 key labels (certificate labels) to be chosen from in order to authenticate the server. The labels are specified in order of preference. The ability to specify more than one label allows for the server to negotiate with a wide range of clients.

During the negotiation of a secure connection, the first appropriate certificate is selected for use by the session based on the attributes for the session. The selection process uses the SSL attributes from both the client and server to make the decision. It is possible that no certificate is usable for a combination of attributes.

The selection order criteria for TLS V1.2, TLS V1.1, TLS V1.0, and SSL V3 is as follows:

- Protocol
- Cipher
- Certificate

TLS V1.3 ignores the cipher when selecting a certificate because TLS V1.3 has dissociated the cipher list and certificate algorithms. When TLS V1.3 has been negotiated, the server's key label list is stepped through and basic validation is performed on the validity and algorithms of each certificate. The first match found from the list is used for the handshake. The specified certificates should match the capabilities of the specified session attributes.

When the negotiated protocol is TLS V1.2, TLS V1.1, TLS V1.0, or SSL V3, the server's cipher comparison list (see Cipher selection for information about how the server cipher comparison list is built) is processed. The cipher list is stepped through looking for a matching cipher suite supported by the client. When a matching cipher is encountered, the server's key label list is searched to locate the first non-expired certificate that supports the matching cipher suite and client supplied ecurves and signature algorithm session attributes when applicable. Additional validation will be performed against the selected certificate as the negotiation process continues. The specified server certificates should match the capabilities of the specified ciphers and session attributes. Should the server attempt to use a cipher that the certificate cannot accommodate, the connection will fail.

Specifying a cached session

When acting as a client, if the client intends to resume a newly created SSL V3 through a TLS V1.2 session at a later time, immediately following the successful **gsk_secure_socket_init()** invocation to establish the connection, a call to **gsk_attribute_get_buffer()** needs to be performed in order to retrieve the peer ID (GSK_PEER_ID) for the newly generated connection. If a TLS V1.3 session has been established successfully in **gsk_secure_socket_init()**, the client must call **gsk_secure_socket_read()** to read in any session tickets that the server may have sent after the handshake has completed. Client applications should use non-blocking sockets when calling the **gsk_secure_socket_read()** routine when reading in session tickets from the server. The use of a blocking socket may cause the **gsk_secure_socket_read()** routine to hang if the server does not send any session tickets. After a successful **gsk_secure_socket_read()** invocation has read in one or more session tickets from the server, a call to **gsk_attribute_get_buffer()** can then be performed in order to retrieve the peer ID (GSK_PEER_ID) for the newly generated connection. GSK_PEER_ID requires the enablement of the peer ID support by specifying environment attribute GSK_ENABLE_CLIENT_SET_PEERID to ON during the establishment of the SSL environment.

The GSK_PEER_ID value is used to locate a session entry or session ticket in the client session cache so that it can be used to resume the session with the server if one exists and is not expired. Session tickets for TLS V1.3 sessions may expire based upon the more restrictive of the GSK_V3_SESSION_TIMEOUT setting or the session ticket lifetime defined by the server issuing the tickets. For SSL V3 through TLS V1.2 sessions, if a matching session entry cannot be located by the server, a full handshake is performed.

For TLS V1.3, the server must receive a valid session ticket from the client that contains a valid session identifier. If not, the server does a full handshake.

The peer ID value provides information to locate the session information within the client session cache. Later, if the client wants to resume this session with the same server, as long as the SSL environment has not been closed, the client can set the peer ID to be used through a call to **gsk_attribute_set_buffer()** before calling **gsk_secure_socket_init()**. If the peer ID's cache entry and session ticket (if using TLS V1.3) is located, the following criteria is verified in order for this cache entry or session ticket to be used:

- The requested cipher must be included within the session's cipher list.
- The requested protocol must be included within the session's supported protocol list.
- The certificate labels must match.
- For TLS V1.3 connections, the key share must be included in the cached session's selected key share.

If a cached session is required, set the GSK_REQ_CACHED_SESSION attribute to ON prior to calling **gsk_secure_socket_init()**. **gsk_secure_socket_init()** fails if the cached entry or session ticket cannot be found. For SSL V3 through TLS V1.2 handshakes, if the session ID value from a server hello does not match the cached session ID, **gsk_secure_socket_init()** fails. For TLS V1.3 connections, if the server does not indicate that the session ticket has been selected for resumption in the server hello, **gsk_secure_socket_init()** fails.

If a cached session is not required, ensure that the GSK_REQ_CACHED_SESSION attribute is OFF (the default) prior to calling **gsk_secure_socket_init()**. This results in a full handshake if the cached entry cannot be found.

When acting as a server, if the next SSL V3, TLS V1.0, or higher connection from a particular client needs to utilize a previously established session, after the call to **gsk_secure_socket_init()** to establish the connection, call **gsk_attribute_get_buffer()** to retrieve the session ID (GSK_SID_VALUE) information for the newly generated connection. For TLS V1.3 connections, the server must be enabled to send session tickets (GSK_SESSION_TICKET_SERVER_ENABLE is set to ON and GSK_SESSION_TICKET_SERVER_COUNT is non-zero or **gsk_secure_socket_misc()** with attribute GSK_SEND_SESSION_TICKET has been called) prior to calling **gsk_attribute_get_buffer()** to retrieve the session ID (GSK_SID_VALUE). Before calling **gsk_secure_socket_init()** for the next connection, call **gsk_attribute_set_buffer()** to set the session ID value.

If a cached session or session ticket is required, a server application does not require setting attribute GSK_REQ_CACHED_SESSION to ON to enforce the reusing of a cached session. By setting the GSK_SID_VALUE prior to calling **gsk_secure_socket_init()**, the server is required to locate that cached session for SSL V3 through TLS V1.2 sessions. For TLS V1.3 sessions, the client must present a session ticket that contains that session ID value on its resumption handshake.

For SSL V3 through TLS V1.2 sessions, GSK_SID_VALUE is used to locate a session entry in the server session cache that can be used to resume the session with a client if one exists and has not expired. **gsk_secure_socket_init()** fails if the cached entry cannot be found. If the session ID value from a client hello does not match the cached session ID, **gsk_secure_socket_init()** fails.

For TLS V1.3 sessions, the GSK_SID_VALUE is used by the server to check the session ID value received from the client in its session ticket. If the session ID in the session ticket does not match the GSK_SID_VALUE or the client does not present a session ticket on the client hello, **gsk_secure_socket_init()** fails. In addition, the server may not be able to decrypt a session ticket received from a client if the server's session keys have been refreshed based upon the GSK_SESSION_TICKET_SERVER_KEY_REFRESH environment attribute setting. In this case, the **gsk_secure_socket_init()** routine fails.

Related topics

- "gsk_environment_init()" on page 134
- "gsk_secure_socket_write()" on page 179
- "gsk_secure_socket_read()" on page 174
- "gsk_secure_socket_misc()" on page 170

- “gsk_secure_socket_close()” on page 156

gsk_secure_socket_misc()

Performs miscellaneous secure connection functions.

Format

```
#include <gskssl.h>

gsk_status gsk_secure_socket_misc (
                                   gsk_handle      soc_handle,
                                   GSK_MISC_ID     misc_id)
```

Parameters

soc_handle
Specifies the connection handle returned by the **gsk_secure_socket_open()** routine.

misc_id
Miscellaneous function identifier.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ERR_BAD_SESSION_TICKET_ATTRIBUTES]
Attempt to send session ticket failed due to invalid resumption attributes.

[GSK_ERR_CONNECTION_CLOSED]
A close notification alert has been sent for the connection.

[GSK_ERR_IO]
I/O error communicating with peer application.

[GSK_ERR_NO_NEGOTIATION]
An attempt was made to renegotiate a session when renegotiation is disabled.

[GSK_ERR_NOT_SSLV3]
The session is not using the SSL V3, TLS V1.0, or higher protocol.

[GSK_ERR_NOT_SUPPORTED_BY_PROTOCOL]
Specified function not supported by protocol version.

[GSK_ERR_SESSION_TICKET_EXPIRED]
The session ticket cannot be sent because the session has expired.

[GSK_ERR_SOCKET_CLOSED]
Socket connection closed by peer application.

[GSK_ERR_SYSPLEX_SESSION_TICKET_CACHE]
Session ticket information cannot be successfully cached.

[GSK_INVALID_HANDLE]
The connection handle is not valid.

[GSK_INVALID_STATE]
The connection is not in the initialized state.

[GSK_MISC_INVALID_ID]
The miscellaneous identifier is not valid.

Usage

The **gsk_secure_socket_misc()** routine performs miscellaneous function for an initialized secure connection.

These miscellaneous functions are provided:

GSK_RESET_CIPHER
This function generates new session keys for the connection. A full SSL handshake will be performed if the session has expired or has been reset by the GSK_RESET_SESSION function. Otherwise, a short SSL handshake will be performed for SSL V3 through TLS V1.2 secure connections. The GSK_RESET_CIPHER function can be performed only for a session using the SSL V3, TLS V1.0, or higher protocol. For SSL V3 through TLS V1.2 secure connections, the GSK_RESET_CIPHER function initiates the SSL handshake, but does not wait for it to complete. Any pending handshake messages will be processed when the **gsk_secure_socket_read()** routine is called to process incoming data.

For a TLS V1.3 secure connection, the read and write session keys are updated by both partners. Any messages sent after the request will be under the new write key. Any pending notification by the partner about the key changes will be processed when the **gsk_secure_socket_read()** routine is called to process incoming data.

GSK_RESET_SESSION
This function resets the session associated with the connection. A full SSL/TLS handshake will be performed for the next connection using the session. The current SSL V3 through TLS V1.2 secure connection is not affected unless the GSK_RESET_CIPHER function is performed after the GSK_RESET_SESSION function has completed. If using session ID caching, specifying GSK_RESET_SESSION causes the cache entry and all session tickets to be deleted for this session. If TLS V1.3 session tickets are cached on the client or sysplex session ticket caching is enabled on the server, specifying GSK_RESET_SESSION causes all session tickets to be deleted for the TLS V1.3 session.

Note: Caution should be taken when specifying GSK_RESET_SESSION and reusing cached sessions.

- Specifying GSK_RESET_CIPHER causes a new cache entry to be created from the session if one did not already exist for SSL V3 through TLS V1.2 secure connections.
- For a client application with multiple connections reusing a cached session entry, the connection using **gsk_secure_socket_misc()** as well as current reused connections will continue to function properly. However, a new connection requiring the use of the cached entry will fail if cache reuse is required and the cache entry or session ticket no longer exists. A new connection not requiring the use of the cached entry or session ticket will result in a full handshake.
- For a server application with multiple SSL V3 through TLS V1.2 connections or enabled for sysplex ticket caching for TLS V1.3 connections reusing a cached session entry, the connection utilizing **gsk_secure_socket_misc()** as well as current reused connections will continue to function properly. However, a new connection requiring the use of the cached entry will fail if cache reuse is required and the cache entry no longer exists.
- For a server application using SSL V3 through TLS V1.2 connections, specify GSK_RENEGOTIATION_ABBREVIATED to ensure successful GSK_RESET_CIPHER when GSK_SID_VALUE is specified.

GSK_RESET_WRITE_CIPHER
For a TLS V1.3 connection, the local write key and the remote partner's read key are updated. Any messages sent after the request will be under the new write key.

This operation is only supported by connections using TLS V1.3.

GSK_SEND_SESSION_TICKET
For a TLS V1.3 server connection, a single session ticket is sent to the client for the current connection. This option can only be used by the server after successful TLS V1.3 handshake negotiation. A session ticket from the server provides the client with the means to request session resumption for TLS V1.3. In order to use this option, GSK_SESSION_TICKET_SERVER_ENABLE must be ON and GSK_SESSION_TICKET_SERVER_COUNT must be set to 0. After the session has expired, the server will be unable to successfully send session tickets until a new full TLS V1.3 handshake has been negotiated.

This operation is only supported by connections using TLS V1.3.

Related topics

- "gsk_secure_socket_open()" on page 173
- "gsk_secure_socket_read()" on page 174
- "gsk_secure_socket_write()" on page 179

gsk_secure_socket_open()

Creates a secure socket connection.

Format

```
#include <gskssl.h>

gsk_status gsk_secure_socket_open (
                                   gsk_handle       env_handle,
                                   gsk_handle *     soc_handle)
```

Parameters

env_handle
Specifies the SSL environment handle returned by the **gsk_environment_open()** routine.

soc_handle
Returns the handle for the secure connection. The application should call the **gsk_secure_socket_close()** routine to release the connection when it is no longer needed.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_INSUFFICIENT_STORAGE]
Insufficient storage is available.

[GSK_INVALID_HANDLE]
The environment handle is not valid.

[GSK_INVALID_STATE]
The environment is not in the initialized state.

Usage

The **gsk_secure_socket_open()** routine creates a secure socket connection. The connection will be initialized with values obtained from the SSL environment. These values can be changed by the application using the appropriate **gsk_attribute_set_*()** routines. The **gsk_secure_socket_init()** routine should then be called to initialize the connection. This connection can then be used to send and receive data with the remote partner.

Related topics

- “gsk_secure_socket_close()” on page 156
- “gsk_secure_socket_init()” on page 157

gsk_secure_socket_read()

Reads data using a secure socket connection.

Format

```
#include <gskssl.h>

gsk_status gsk_secure_socket_read (
                                   gsk_handle     soc_handle,
                                   char *         buffer,
                                   int            size,
                                   int *          length)
```

Parameters

soc_handle
Specifies the connection handle returned by the **gsk_secure_socket_open()** routine.

buffer
Specifies the buffer to receive the data read from the secure socket connection. The maximum amount of data returned by **gsk_secure_socket_read()** is 16384 (16K) bytes. If the SSL V2 protocol is used, then the maximum length is 16384 minus the length of the SSL protocol headers.

size
Specifies the size of the supplied buffer.

length
Returns the length of the data read into the supplied buffer.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_CONNECTION_ACTIVE]
A read request is already active for the connection.

[GSK_ERR_BAD_MAC]
Message verification failed.

[GSK_ERR_BAD_MESSAGE]
Incorrectly-formatted message received from peer application.

[GSK_ERR_BAD_PEER]
Peer application has violated the SSL protocol.

[GSK_ERR_CERTIFICATE_REVOKED]
Peer certificate is revoked.

[GSK_ERR_CONNECTION_CLOSED]
Close notification received from peer application.

[GSK_ERR_CRYPTO]
Cryptographic error detected.

[GSK_ERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[GSK_ERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[GSK_ERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[GSK_ERR_IO]
I/O error communicating with peer application.

[GSK_ERR_MISMATCH_TLS_EXT_MASTER_SECRET]
Extended master secret extension mismatch detected on cached TLS handshake attempt.

[GSK_ERR_NO_NEGOTIATION]
An attempt was made to renegotiate a session when renegotiation is disabled or the peer rejected an attempted session renegotiation.

[GSK_ERR_REQUIRED_TLS_EXT_MASTER_SECRET]
Missing required TLS extended master secret extension from the remote partner.

[GSK_ERROR_RENEGOTIATION_INDICATION]
Peer did not signal support for TLS Renegotiation Indication.

[GSK_ERR_SOCKET_CLOSED]
Socket connection closed by peer application.

[GSK_INSUFFICIENT_STORAGE]
Insufficient storage is available.

[GSK_INVALID_BUFFER_SIZE]
The buffer address or buffer size is not valid.

[GSK_INVALID_HANDLE]
The connection handle is not valid.

[GSK_INVALID_STATE]
The connection is not in the initialized state.

[GSK_WOULD_BLOCK]
A complete SSL record is not available.

[GSK_WOULD_BLOCK_WRITE]
An SSL handshake is in progress but data cannot be written to the socket.

Usage

The **gsk_secure_socket_read()** routine reads data from a secure socket connection and returns it in the application buffer. SSL is a record-based protocol and a single call does not return more than a single SSL record. The maximum amount of data returned by **gsk_secure_socket_read()** is 16384 (16K) bytes. If the SSL V2 protocol is used, then the maximum length is 16384 minus the length of the SSL protocol headers. The application can read an entire SSL record in a single call by supplying a buffer large enough to contain the record. Otherwise, multiple calls will be required to retrieve the entire SSL record.

SSL supports multiple threads but only one thread at a time can call the **gsk_secure_socket_read()** routine for a given connection handle. Multiple concurrent threads can call **gsk_secure_socket_read()** if each thread has its own connection handle.

SSL supports sockets in blocking mode and in non-blocking mode. When a socket is in non-blocking mode and a complete SSL record is not available, **gsk_secure_socket_read()** will return with GSK_WOULD_BLOCK. No data will be returned in the application buffer when GSK_WOULD_BLOCK is returned. The application should call **gsk_secure_socket_read()** again when there is data available to be read from the socket.

The peer application can initiate an SSL handshake sequence after the connection is established. If this is done and the socket is in non-blocking mode, it is possible for **gsk_secure_socket_read()** to return with GSK_WOULD_BLOCK_WRITE. This indicates that an SSL handshake is in progress and the application should call **gsk_secure_socket_read()** again when data can be written to the socket. No data will be returned in the application buffer when GSK_WOULD_BLOCK_WRITE is returned.

The application should not read data directly from the socket since this can cause SSL protocol errors if the application inadvertently reads part of an SSL record. If the application must read data from the socket, it is responsible for synchronizing this activity with the peer application so that no SSL records are sent while the application is performing its own read operations.

Related topics

- “gsk_secure_socket_write()” on page 179
- “gsk_secure_socket_init()” on page 157

gsk_secure_socket_shutdown()

Shuts down a secure socket connection.

Format

```
    #include <gskssl.h>

     gsk_status gsk_secure_socket_shutdown (
                                          gsk_handle     soc_handle)
```

Parameters

soc_handle
Specifies the connection handle returned by the **gsk_secure_socket_open()** routine.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_CONNECTION_ACTIVE]
The connection has an active write request.

[GSK_ERR_CONNECTION_CLOSED]
The close notification alert has already been sent.

[GSK_ERR_IO]
I/O error communicating with peer application.

[GSK_ERR_NOT_SSLV3]
The session is not using the SSL V3, TLS V1.0, or higher protocol.

[GSK_ERR_SOCKET_CLOSED]
Socket connection closed by peer application.

[GSK_INVALID_HANDLE]
The connection handle is not valid.

[GSK_INVALID_STATE]
The connection is not in the initialized state.

[GSK_WOULD_BLOCK_WRITE]
An attempt to write pending data failed with EWOULDBLOCK.

Usage

The **gsk_secure_socket_shutdown()** routine will send a close notification alert to the peer application. Any subsequent calls to the **gsk_secure_socket_write()** routine will return GSK_ERR_CONNECTION_CLOSED. The **gsk_secure_socket_shutdown()** routine cannot be used with the SSL V2 protocol.

The application should call **gsk_secure_socket_shutdown()** before calling **gsk_secure_socket_close()** in order to comply with the SSL V3, TLS V1.0, or higher specifications, which require that a close notification alert be sent before closing the transport connection.

For a 1-step shutdown, the application should call the **gsk_secure_socket_shutdown()** routine and then call the **gsk_secure_socket_close()** routine. This sends the close notification alert and then closes the secure socket connection. The application does not wait for acknowledgement from the peer application to the close notification.

For a 2-step shutdown, the application should call the **gsk_secure_socket_shutdown()** routine to send the close notification alert and then call the **gsk_secure_socket_read()** routine to process any pending data sent by the peer application. The SSL run time on the peer system will send a close notification alert when it receives the close notification alert from the local system. The **gsk_secure_socket_read()** routine will return GSK_ERR_CONNECTION_CLOSED when it receives this close notification. The application should then call the **gsk_secure_socket_close()** routine to close the secure socket connection.

Related topics

- “gsk_secure_socket_close()” on page 156
- “gsk_secure_socket_open()” on page 173
- “gsk_secure_socket_read()” on page 174
- “gsk_secure_socket_write()” on page 179

gsk_secure_socket_write()

Writes data using a secure socket connection.

Format

```
#include <gskssl.h>

gsk_status gsk_secure_socket_write (
                                    gsk_handle     soc_handle,
                                    char *         buffer,
                                    int            size,
                                    int *          length)
```

Parameters

soc_handle
Specifies the connection handle returned by the **gsk_secure_socket_open()** routine.

buffer
Specifies the buffer containing the data to write to the secure socket connection.

size
Specifies the amount to write.

length
Returns the length of the data written.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_CONNECTION_ACTIVE]
A write request is already active for the connection.

[GSK_ERR_CONNECTION_CLOSED]
A close notification alert has been sent for the connection.

[GSK_ERR_CRYPTO]
Cryptographic error detected.

[GSK_ERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[GSK_ERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[GSK_ERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[GSK_ERR_IO]
I/O error communicating with peer application.

[GSK_ERR_SOCKET_CLOSED]
Socket connection closed by peer application.

[GSK_INSUFFICIENT_STORAGE]
Insufficient storage is available.

[GSK_INVALID_BUFFER_SIZE]
The buffer address or buffer size is not valid.

[GSK_INVALID_HANDLE]
The connection handle is not valid.

[GSK_INVALID_STATE]
The connection is not in the initialized state.

[GSK_WOULD_BLOCK]
The SSL record cannot be written to the socket because of an EWOULDBLOCK condition.

Usage

The **gsk_secure_socket_write()** routine writes data to a secure socket connection. SSL is a record-based protocol with a maximum record length of 16384 bytes. If the SSL V2 protocol is used, then the maximum length is 16384 minus the length of the SSL protocol headers. Application data larger than the size of an SSL record will be sent using multiple records.

SSL supports multiple threads but only one thread at a time can call the **gsk_secure_socket_write()** routine for a given connection handle. Multiple concurrent threads can call **gsk_secure_socket_write()** if each thread has its own connection handle.

SSL supports sockets in blocking mode and in non-blocking mode. When a socket is in non-blocking mode and the SSL record cannot be written to the socket, **gsk_secure_socket_write()** will return with GSK_WOULD_BLOCK. The application must call **gsk_secure_socket_write()** again when the socket is ready to accept more data, specifying the same buffer address and buffer size as the original request. A new write request must not be initiated until the pending write request has been completed as indicated by a return value of 0.

The application should not write data directly to the socket since this can cause SSL protocol errors if the application inadvertently intermixes its data with SSL protocol data. If the application must write data to the socket, it is responsible for synchronizing this activity with the peer application so that application data is not intermixed with SSL data.

To notify your partner application that you are done sending data on the secure connection, a call to **gsk_secure_socket_shutdown()** should be issued before the **gsk_secure_socket_close()** call.

Related topics

- "gsk_secure_socket_read()" on page 174
- "gsk_secure_socket_init()" on page 157

gsk_strerror()

Return a text string for an SSL error code

Format

```
#include <gskssl.h>

const char * gsk_strerror (
                            gsk_status          error_code)
```

Parameters

error_code
Specifies an error code returned by a Secure Sockets layer (SSL) routine or by a Certificate Management Services (CMS) routine.

Results

The function return value is the address of the text string. The return value is always a valid text string address even when the error code is not recognized (the return value is the string "N/A" in this case).

Usage

The **gsk_strerror()** routine returns a text string describing an error code returned by an SSL (Secure Sockets layer) or CMS (Certificate Management Services) routine. The **gsk_strerror()** routine cannot be used to return a text string for an error code returned by one of the deprecated SSL routines. The text string must not be modified or released by the application program.

If **gsk_strerror()** routine is called with an error code that is not recognized to be a valid value from either the SSL or CMS routines, one of the following messages is displayed to stderr:

- Message *xxx* not found in catalog gskmsgs.cat.
- Message identifier *xxx* is not defined.

Notes:

- Error codes from the CMS routines are defined to be in the range '03353001' through '03353FFF' and '014CE001' through '014CEFFF'.
- SSL error codes are defined internally to be '134CC*xxx*', where *xxx* is the hexadecimal representation of the SSL error code. The range is '134CC001' through '134CCFFF'. For example, SSL error code 414 would be '134CC19E'.

If the error code is within the recognized ranges, but not a currently defined error code, 'Message *xxx* not found in catalog gskmsgs.cat' is display indicating that the error code message in not in the catalog. If the error code is not within the recognized ranges, 'Message identifier *xxx* is not defined' is displayed.

If the System SSL message catalog cannot be opened or found, the following message is displayed to stderr:

- Unable to open message catalog gskmsgs.cat.

Chapter 8. Certificate Management Services (CMS) API reference

This topic describes the Certificate Management Services (CMS) APIs. These APIs can be used to create/manage your own key database files in a similar function to the SSL **gskkyman** utility, use certificates stored in the key database file or key ring for purposes other than SSL, and basic PKCS #7 message support.

System SSL supports X.509 certificates (V1, V2, or V3) and X.509 V2 Certificate Revocation Lists as described in RFC 5280 (tools.ietf.org/html/rfc5280), RFC 3280 (tools.ietf.org/html/rfc3280) and RFC 2459 (tools.ietf.org/html/rfc2459). RFC 5280 obsoletes RFC 3280 which obsoletes RFC 2459.

Note: You can use the **gsk_strerror()** routine to return a text string describing a CMS error code. See "gsk_strerror()" on page 181 for more information.

This is a list of the Certificate Management Services (CMS) APIs:

- **gsk_add_record()** (see "gsk_add_record()" on page 188)
- **gsk_change_database_password()** (see "gsk_change_database_password()" on page 191)
- **gsk_change_database_record_length()** (see "gsk_change_database_record_length()" on page 193)
- **gsk_close_database()** (see "gsk_close_database()" on page 194)
- **gsk_close_directory()** (see "gsk_close_directory()" on page 195)
- **gsk_construct_certificate()** (see "gsk_construct_certificate()" on page 196)
- **gsk_construct_private_key()** (see "gsk_construct_private_key()" on page 200)
- **gsk_construct_private_key_rsa()** (see "gsk_construct_private_key_rsa()" on page 202)
- **gsk_construct_public_key()** (see "gsk_construct_public_key()" on page 203)
- **gsk_construct_public_key_rsa()** (see "gsk_construct_public_key_rsa()" on page 205)
- **gsk_construct_renewal_request()** (see "gsk_construct_renewal_request()" on page 206)
- **gsk_construct_self_signed_certificate()** (see "gsk_construct_self_signed_certificate()" on page 209)
- **gsk_construct_signed_certificate()** (see "gsk_construct_signed_certificate()" on page 212)
- **gsk_construct_signed_crl()** (see "gsk_construct_signed_crl()" on page 216)
- **gsk_copy_attributes_signers()** (see "gsk_copy_attributes_signers()" on page 220)
- **gsk_copy_buffer()** (see "gsk_copy_buffer()" on page 221)
- **gsk_copy_certificate()** (see "gsk_copy_certificate()" on page 222)
- **gsk_copy_certificate_extension()** (see "gsk_copy_certificate_extension()" on page 223)
- **gsk_copy_certification_request()** (see "gsk_copy_certification_request()" on page 224)
- **gsk_copy_content_info()** (see "gsk_copy_content_info()" on page 225)
- **gsk_copy_crl()** (see "gsk_copy_crl()" on page 226)
- **gsk_copy_name()** (see "gsk_copy_name()" on page 227)
- **gsk_copy_private_key_info()** (see "gsk_copy_private_key_info()" on page 228)
- **gsk_copy_public_key_info()** (see "gsk_copy_public_key_info()" on page 229)
- **gsk_copy_record()** (see "gsk_copy_record()" on page 230)
- **gsk_create_certification_request()** (see "gsk_create_certification_request()" on page 231)
- **gsk_create_database()** (see "gsk_create_database()" on page 235)
- **gsk_create_database_renewal_request()** (see "gsk_create_database_renewal_request()" on page 237)

- **gsk_create_database_signed_certificate()** (see "gsk_create_database_signed_certificate()" on page 240)
- **gsk_create_renewal_request()** (see "gsk_create_renewal_request()" on page 246)
- **gsk_create_revocation_source()** (see "gsk_create_revocation_source()" on page 249)
- **gsk_create_self_signed_certificate()** (see "gsk_create_self_signed_certificate()" on page 255)
- **gsk_create_signed_certificate()** (see "gsk_create_signed_certificate()" on page 259)
- **gsk_create_signed_certificate_record()** (see "gsk_create_signed_certificate_record()" on page 262)
- **gsk_create_signed_certificate_set()** (see "gsk_create_signed_certificate_set()" on page 266)
- **gsk_create_signed_crl()** (see "gsk_create_signed_crl()" on page 271)
- **gsk_create_signed_crl_record()** (see "gsk_create_signed_crl_record()" on page 274)
- **gsk_decode_base64()** (see "gsk_decode_base64()" on page 278)
- **gsk_decode_certificate()** (see "gsk_decode_certificate()" on page 279)
- **gsk_decode_certificate_extension()** (see "gsk_decode_certificate_extension()" on page 280)
- **gsk_decode_certification_request()** (see "gsk_decode_certification_request()" on page 282)
- **gsk_decode_crl()** (see "gsk_decode_crl()" on page 283)
- **gsk_decode_import_certificate()** (see "gsk_decode_import_certificate()" on page 284)
- **gsk_decode_import_key()** (see "gsk_decode_import_key()" on page 285)
- **gsk_decode_issuer_and_serial_number()** (see "gsk_decode_issuer_and_serial_number()" on page 287)
- **gsk_decode_name()** (see "gsk_decode_name()" on page 288)
- **gsk_decode_private_key()** (see "gsk_decode_private key()" on page 289)
- **gsk_decode_public_key()** (see "gsk_decode_public key()" on page 290)
- **gsk_decode_signer_identifier()** (see "gsk_decode_signer_identifier()" on page 291)
- **gsk_delete_record()** (see "gsk_delete_record()" on page 292)
- **gsk_dn_to_name()** (see "gsk_dn_to_name()" on page 293)
- **gsk_encode_base64()** (see "gsk_encode_base64()" on page 296)
- **gsk_encode_certificate_extension()** (see "gsk_encode_certificate_extension()" on page 297)
- **gsk_encode_ec_parameters()** (see "gsk_encode_ec_parameters()" on page 299)
- **gsk_encode_export_certificate()** (see "gsk_encode_export_certificate()" on page 300)
- **gsk_encode_export_key()** (see "gsk_encode_export_key()" on page 302)
- **gsk_encode_export_request()** (see "gsk_encode_export_request()" on page 305)
- **gsk_encode_issuer_and_serial_number()** (see "gsk_encode_issuer_and_serial_number()" on page 306)
- **gsk_encode_name()** (see "gsk_encode_name()" on page 307)
- **gsk_encode_private_key()** (see "gsk_encode_private_key()" on page 308)
- **gsk_encode_public_key()** (see "gsk_encode_public_key()" on page 309)
- **gsk_encode_signature()** (see "gsk_encode_signature()" on page 310)
- **gsk_encode_signer_identifier()** (see "gsk_encode_signer_identifier()" on page 311)
- **gsk_export_certificate()** (see "gsk_export_certificate()" on page 312)
- **gsk_export_certification_request()** (see "gsk_export_certification_request()" on page 314)
- **gsk_export_key()** (see "gsk_export_key()" on page 315)
- **gsk_factor_private_key()** (see "gsk_factor_private_key()" on page 318)
- **gsk_factor_private_key_rsa()** (see "gsk_factor_private_key_rsa()" on page 319)
- **gsk_factor_public_key()** (see "gsk_factor_public_key()" on page 320)

- **gsk_factor_public_key_rsa()** (see "gsk_factor_public_key_rsa()" on page 321)
- **gsk_fips_state_query()** (see "gsk_fips_state_query()" on page 322)
- **gsk_fips_state_set()** (see "gsk_fips_state_set()" on page 323)
- **gsk_format_long_integer()** (see "gsk_format_long_integer()" on page 325)
- **gsk_format_time()** (see "gsk_format_time()" on page 326)
- **gsk_free_attributes_signers()** (see "gsk_free_attributes_signers()" on page 328)
- **gsk_free_buffer()** (see "gsk_free_buffer()" on page 329)
- **gsk_free_certificate()** (see "gsk_free_certificate()" on page 330)
- **gsk_free_certificates()** (see "gsk_free_certificates()" on page 331)
- **gsk_free_certificate_extension()** (see "gsk_free_certificate_extension()" on page 332)
- **gsk_free_certification_request()** (see "gsk_free_certification_request()" on page 333)
- **gsk_free_content_info()** (see "gsk_free_content_info()" on page 334)
- **gsk_free_crl()** (see "gsk_free_crl()" on page 335)
- **gsk_free_crls()** (see "gsk_free_crls()" on page 336)
- **gsk_free_decoded_extension()** (see "gsk_free_decoded_extension()" on page 337)
- **gsk_free_issuer_and_serial_number()** (see "gsk_free_issuer_and_serial_number()" on page 338)
- **gsk_free_name()** (see "gsk_free_name()" on page 339)
- **gsk_free_oid()** (see "gsk_free_oid()" on page 340)
- **gsk_free_private_key()** (see "gsk_free_private_key()" on page 341)
- **gsk_free_private_key_info()** (see "gsk_free_private_key_info()" on page 342)
- **gsk_free_public_key()** (see "gsk_free_public_key()" on page 343)
- **gsk_free_public_key_info()** (see "gsk_free_public_key_info()" on page 344)
- **gsk_free_record()** (see "gsk_free_record()" on page 345)
- **gsk_free_records()** (see "gsk_free_records()" on page 346)
- **gsk_free_revocation_source()** (see "gsk_free_revocation_source()" on page 347)
- **gsk_free_signer_identifier()** (see "gsk_free_signer_identifier()" on page 348)
- **gsk_free_string()** (see "gsk_free_string()" on page 349)
- **gsk_free_strings()** (see "gsk_free_strings()" on page 350)
- **gsk_free_x509_diagnostics()** (see "gsk_free_x509_diagnostics()" on page 351)
- **gsk_generate_key_agreement_pair()** (see "gsk_generate_key_agreement_pair()" on page 352)
- **gsk_generate_key_pair()** (see "gsk_generate_key_pair()" on page 353)
- **gsk_generate_key_parameters()** (see "gsk_generate_key_parameters()" on page 356)
- **gsk_generate_random_bytes()** (see "gsk_generate_random_bytes()" on page 358)
- **gsk_generate_secret()** (see "gsk_generate_secret()" on page 359)
- **gsk_get_certificate_algorithms()** (see "gsk_get_certificate_algorithms()" on page 360)
- **gsk_get_certificate_info()** (see "gsk_get_certificate_info() " on page 361)
- **gsk_get_cms_vector()** (see "gsk_get_cms_vector()" on page 362)
- **gsk_get_content_type_and_cms_version()** (see "gsk_get_content_type_and_cms_version()" on page 364)
- **gsk_get_default_key()** (see "gsk_get_default_key()" on page 365)
- **gsk_get_default_label()** (see "gsk_get_default_label()" on page 366)
- **gsk_get_directory_certificates()** (see "gsk_get_directory_certificates()" on page 367)
- **gsk_get_directory_crls()** (see "gsk_get_directory_crls()" on page 369)
- **gsk_get_directory_enum()** (see "gsk_get_directory_enum()" on page 371)

- **gsk_get_directory_numeric_value()** (see "gsk_get_directory_numeric_value()" on page 373)
- **gsk_get_ec_parameters_info()** (see "gsk_get_ec_parameters_info()" on page 374)
- **gsk_get_record_by_id()** (see "gsk_get_record_by_id()" on page 375)
- **gsk_get_record_by_index()** (see "gsk_get_record_by_index()" on page 376)
- **gsk_get_record_by_label()** (see "gsk_get_record_by_label()" on page 377)
- **gsk_get_record_by_subject()** (see "gsk_get_record_by_subject()" on page 378)
- **gsk_get_record_labels()** (see "gsk_get_record_labels()" on page 379)
- **gsk_get_update_code()** (see "gsk_get_update_code()" on page 380)
- **gsk_import_certificate()** (see "gsk_import_certificate()" on page 381)
- **gsk_import_key()** (see "gsk_import_key()" on page 384)
- **gsk_make_content_msg()** (see "gsk_make_content_msg()" on page 387)
- **gsk_make_data_content()** (see "gsk_make_data_content()" on page 388)
- **gsk_make_data_msg()** (see "gsk_make_data_msg()" on page 389)
- **gsk_make_encrypted_data_content()** (see "gsk_make_encrypted_data_content()" on page 390)
- **gsk_make_encrypted_data_msg()** (see "gsk_make_encrypted_data_msg()" on page 392)
- **gsk_make_enveloped_data_content()** (see "gsk_make_enveloped_data_content()" on page 394)
- **gsk_make_enveloped_data_content_extended()** (see "gsk_make_enveloped_data_content_extended()" on page 397)
- **gsk_make_enveloped_data_msg()** (see "gsk_make_enveloped_data_msg()" on page 400)
- **gsk_make_enveloped_data_msg_extended()** (see "gsk_make_enveloped_data_msg_extended()" on page 403)
- **gsk_make_enveloped_private_key_msg()** (see "gsk_make_enveloped_private_key_msg()" on page 406
- **gsk_make_signed_data_content()** (see "gsk_make_signed_data_content()" on page 408)
- **gsk_make_signed_data_content_extended()** (see "gsk_make_signed_data_content_extended()" on page 411)
- **gsk_make_signed_data_msg()** (see "gsk_make_signed_data_msg()" on page 415)
- **gsk_make_signed_data_msg_extended()** (see "gsk_make_signed_data_msg_extended()" on page 418)
- **gsk_make_wrapped_content()** (see "gsk_make_wrapped_content()" on page 422)
- **gsk_mktime()** (see "gsk_mktime()" on page 423)
- **gsk_modify_pkcs11_key_label()** (see "gsk_modify_pkcs11_key_label()" on page 424
- **gsk_name_compare()** (see "gsk_name_compare()" on page 425)
- **gsk_name_to_dn()**(see "gsk_name_to_dn()" on page 426)
- **gsk_open_database()** (see "gsk_open_database()" on page 428)
- **gsk_open_database_using_stash_file()** (see "gsk_open_database_using_stash_file()" on page 430)
- **gsk_open_directory()** (see "gsk_open_directory()" on page 432)
- **gsk_open_keyring()** (see "gsk_open_keyring()" on page 433)
- **gsk_perform_kat()** (see "gsk_perform_kat()" on page 435)
- **gsk_query_crypto_level()** (see "gsk_query_crypto_level()" on page 436)
- **gsk_query_database_label()** (see "gsk_query_database_label()" on page 437)
- **gsk_query_database_record_length()** (see "gsk_query_database_record_length()" on page 438)
- **gsk_rdtime()** (see "gsk_rdtime()" on page 439)
- **gsk_read_content_msg()** (see "gsk_read_content_msg()" on page 440)
- **gsk_read_data_content()** (see "gsk_read_data_content()" on page 441)

- **gsk_read_data_msg()** (see "gsk_read_data_msg()" on page 442)
- **gsk_read_encrypted_data_content()** (see "gsk_read_encrypted_data_content()" on page 443)
- **gsk_read_encrypted_data_msg()** (see "gsk_read_encrypted_data_msg()" on page 445)
- **gsk_read_enveloped_data_content()** (see "gsk_read_enveloped_data_content()" on page 447)
- **gsk_read_enveloped_data_content_extended()** (see "gsk_read_enveloped_data_content_extended()" on page 449)
- **gsk_read_enveloped_data_msg()** (see "gsk_read_enveloped_data_msg()" on page 451)
- **gsk_read_enveloped_data_msg_extended()** (see "gsk_read_enveloped_data_msg_extended()" on page 453)
- **gsk_read_signed_data_content()** (see "gsk_read_signed_data_content()" on page 455)
- **gsk_read_signed_data_content_extended()** (see "gsk_read_signed_data_content_extended()" on page 458)
- **gsk_read_signed_data_msg()** (see "gsk_read_signed_data_msg()" on page 461)
- **gsk_read_signed_data_msg_extended()** (see "gsk_read_signed_data_msg_extended()" on page 464)
- **gsk_read_wrapped_content()** (see "gsk_read_wrapped_content()" on page 467)
- **gsk_receive_certificate()** (see "gsk_receive_certificate()" on page 468)
- **gsk_replace_record()** (see "gsk_replace_record()" on page 469)
- **gsk_set_default_key()** (see "gsk_set_default_key()" on page 472)
- **gsk_set_directory_enum()** (see "gsk_set_directory_enum()" on page 474)
- **gsk_set_directory_numeric_value()** (see "gsk_set_directory_numeric_value()" on page 476)
- **gsk_sign_certificate()** (see "gsk_sign_certificate()" on page 477)
- **gsk_sign_crl()** (see "gsk_sign_crl()" on page 480)
- **gsk_sign_data()** (see "gsk_sign_data()" on page 483)
- **gsk_validate_certificate()** (see "gsk_validate_certificate()" on page 486)
- **gsk_validate_certificate_mode()** (see "gsk_validate_certificate_mode()" on page 491)
- **gsk_validate_extended_key_usage()** (see "gsk_validate_extended_key_usage()" on page 498)
- **gsk_validate_hostname()** (see "gsk_validate_hostname()" on page 500)
- **gsk_validate_server()** (see "gsk_validate_server()" on page 502)
- **gsk_verify_certificate_signature()** (see "gsk_verify_certificate_signature()" on page 503)
- **gsk_verify_crl_signature()** (see "gsk_verify_crl_signature()" on page 506)
- **gsk_verify_data_signature()** (see "gsk_verify_data_signature()" on page 509)

gsk_add_record()

Adds a record to a key or request database.

Format

```
#include <gskcms.h>

gsk_status gsk_add_record (
                            gsk_handle          db_handle,
                            gskdb_record *      record)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine.

record
Specifies the database record.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The key algorithm or signature algorithm is not supported.

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_BAD_RNG_OUTPUT]
In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_DUPLICATE_CERTIFICATE]
The database already contains the certificate.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The record type is not supported for the database type.

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
No private key is provided for a record type that requires a private key.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_RECTYPE_NOT_VALID]
The record type is not valid.

[CMSERR_UNSUPPORTED_EXPIRATION]
The expiration date exceeds February 6, 2106, at 23:59:59 UTC.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_add_record()** routine adds a record to a key or request database. The database must be open for update in order to add records. Unused and reserved fields in the gskdb_record structure must be initialized to zero. An error will be returned when adding a certificate to a key database if the database already contains the certificate. If the record has a private key, the encrypted private key will be generated from the private key supplied in the database record.

The *recordType* field identifies the database record type as follows:

gskdb_rectype_certificate
The record contains an X.509 certificate

gskdb_rectype_certKey
The record contains an X.509 certificate and private key

gskdb_rectype_keyPair
The record contains a PKCS #10 certification request and private key

The *recordFlags* field is a bit field with these values:

GSKDB_RECFLAG_TRUSTED
The certificate is trusted

GSKDB_RECFLAG_DEFAULT
This is the default key

A unique record identifier is assigned when the record is added to the database and will be returned to the application in the *recordId* field. If the record contains an X.509 certificate, the *issuerRecordId* field will be set to the record identifier of the certificate issuer.

The record label is used as a friendly name for the database entry and is in the local code page. It can be set to any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be set to an empty string.

If the record contains an X.509 certificate, the certificate will be validated and the record will not be added to the database if the validation check fails. If the database is a FIPS key database, then the certificate must use only FIPS algorithms and key sizes.

Except for the record label, all character strings are specified using UTF-8.

The database file is updated as part of the **gsk_add_record()** processing. A temporary database file is created using the same name as the database file with ".new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, "System SSL and FIPS 140-2," on page 19.

gsk_change_database_password()

Changes the database password.

Format

```
#include <gskcms.h>

gsk_status gsk_change_database_password (
                                          const char *         filename,
                                          const char *         old_password,
                                          const char *         new_password,
                                          gsk_time             pwd_expiration)
```

Parameters

filename
Specifies the database file name in the local code page. The length of the fully-qualified file name cannot exceed 251.

old_password
Specifies the current database password in the local code page. The user will be prompted to enter the password if NULL is specified for this parameter.

new_password
Specifies the new database password in the local code page. The user will be prompted to enter the password if NULL is specified for this parameter.

pwd_expiration
Specifies the new password expiration time as the number of seconds since the POSIX epoch. A value of 0 indicates the password does not expire.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ACCESS_DENIED]
The file permissions do not allow access.

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_FILENAME]
The database file name is not valid.

[CMSERR_DB_CORRUPTED]
The database file is not valid.

[CMSERR_DB_FIPS_MODE_ONLY]
Key database can only be opened for update if running in FIPS mode.

[CMSERR_DB_LOCKED]
The database is open for update by another process.

[CMSERR_DB_NOT_FIPS]
Key database is not a FIPS mode database.

[CMSERR_FILE_NOT_FOUND]
The database file is not found.

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_IO_CANCELED]
The user canceled the password prompt.

[CMSERR_IO_ERROR]
An input/output request failed.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_OPEN_FAILED]
Unable to open the database.

[CMSERR_PW_INCORRECT]
The password is not correct.

Usage

The **gsk_change_database_password()** routine will change the password for the database and set a new password expiration time. **gsk_mktime()** can be used to convert a year/month/day time value to the number of seconds since the POSIX epoch.

A FIPS database password may only be changed while executing in FIPS mode. A non-FIPS database password can only be changed if not executing in FIPS mode.

gsk_change_database_record_length()

Changes the database record length.

Format

```
#include <gskcms.h>

gsk_status gsk_change_record_length (
                                       gsk_handle     db_handle,
                                       gsk_size       record_length)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine.

record_length
Specifies the new database record length. The default record length will be used if zero is specified for this parameter. All records in the database will have this length. The minimum record length is 2500. The default record length is 5000.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_IO_ERROR]
An input/output request failed.

[CMSERR_LENGTH_TOO_SMALL]
The record length is less than the minimum value.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_TOO_BIG]
A record in the database is larger than the new record length.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_change_database_record_length()** routine will change the record length for the database. All records in the database have the same length and a database entry cannot span records. An error will be returned if the requested record length is smaller than the largest entry in the database.

gsk_close_database()

Closes a key or request database.

Format

```
#include <gskcms.h>

gsk_status gsk_close_database (
                                gsk_handle *          db_handle)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine. The handle will be set to NULL upon successful completion.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_BAD_HANDLE]
The database handle is not valid.

Usage

The **gsk_close_database()** routine will close a key or request database. The db_handle will not be valid upon return from the **gsk_close_database()** routine.

gsk_close_directory()

Closes an LDAP directory.

Format

```
#include <gskcms.h>

gsk_status gsk_close_directory (
                              gsk_handle *    directory_handle)
```

Parameters

directory_handle
Specifies the directory handle returned by the **gsk_open_directory()** routine. The handle will be set to NULL upon successful completion.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_BAD_HANDLE]
The directory handle is not valid.

Usage

The **gsk_close_directory()** routine closes an LDAP directory opened by the **gsk_open_directory()** routine. The directory_handle is not valid upon return from the **gsk_close_directory()** routine.

gsk_construct_certificate()

Constructs a signed certificate and returns it to the caller.

Format

```
#include <gskcms.h>

gsk_status gsk_construct_certificate (
                             pkcs_cert_key *            issuer_certificate,
                             x509_algorithm_type        signature_algorithm,
                             const char *               subject_name,
                             int                        num_days,
                             gsk_boolean                ca_certificate,
                             x509_extensions *          extensions,
                             x509_public_key_info *     public_key,
                             x509_certificate *         subject_certificate)
```

Parameters

issuer_certificate
Specifies the issuing CA certificate with private key.

signature_algorithm
Specifies the signature algorithm for the certificate.

subject_name
Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

num_days
Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

ca_certificate
Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

extensions
Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

public_key
Specifies the public key for the constructed certificate.

subject_certificate
Contains the constructed certificate.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not valid.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_SUBJECT_NAME]
The subject name is not valid.

[CMSERR_CA_NOT_SUPPLIED]
Signing Certificate Authority Certificate not supplied.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_EXPIRED]
The signer certificate is expired.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_KEY_TYPE]
Incorrect key algorithm.

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing certificates.

[CMSERR_ISSUER_NOT_CA]
The signer certificate is not for a certification authority.

[CMSERR_KEY_MISMATCH]
The signer certificate key cannot be used to sign a certificate or the key type is not supported for the requested signature algorithm.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_SUBJECT_IS_CA]
The requested subject name is the same as the signer name.

[CMSERR_UNSUPPORTED_EXPIRATION]
The expiration date exceeds February 6, 2106, at 23:59:59 UTC.

Usage

The **gsk_construct_certificate()** routine will construct an X.509 certificate as described in RFC 5280 (tools.ietf.org/html/rfc5280). The certificate will be signed using the certificate as supplied by the *issuer_certificate* parameter.

- If the supplied *public_key* contains a Diffie-Hellman key, the *issuer_certificate* must contain either an RSA or a DSA key.
- If the supplied *public_key* is an ECC key, the *issuer_certificate* cannot contain a DSA key.

A certification authority (CA) certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions as follows:

- An RSA key can be used for authentication, digital signature, and data encryption. An RSA key can be used for both CA certificates and end user certificates.
- A DSS key can be used for authentication and digital signature. A DSS key can be used for both CA certificates and end user certificates.
- A Diffie_Hellman key can be used for key agreement. A Diffie-Hellman key can be used only for end user certificates.
- An ECC key can be used for authentication, digital signature and key agreement. An ECC key can be used for both CA certificates and end user certificates.

The new certificate is returned in the supplied x509_certificate structure.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest – {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest – {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest – {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest – {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest – {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest – {1.2.840.10045.4.3.4}

These RSASSA-PSS [1.2.840.113549.1.1.10] signatures are supported:

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation function algorithm 1.

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An AuthorityKeyIdentifier extension will be created if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage or BasicConstraints extension provided by the application will replace the default extension constructed for the certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, "System SSL and FIPS 140-2," on page 19.

gsk_construct_private_key()

Constructs a private key from its component values.

Format

```
#include <gskcms.h>

gsk_status gsk_construct_private_key (
                             gsk_private_key *          private_key_factors,
                             pkcs_private_key_info *    private_key)
```

Parameters

private_key_factors
Specifies the private key structure containing the key algorithm type and private key components.

private_key
Returns the private key.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
Cryptographic algorithm is not supported.

[CMSERR_BASE_NOT_SUPPLIED]
Base not supplied.

[CMSERR_COEFFICIENT_NOT_SUPPLIED]
CRT Coefficient not supplied.

[CMSERR_EC_PARAMETERS_NOT_SUPPLIED]
EC parameters not supplied.

[CMSERR_MODULUS_NOT_SUPPLIED]
Modulus not supplied.

[CMSERR_PRIME_EXPONENT1_NOT_SUPPLIED]
First prime exponent not supplied.

[CMSERR_PRIME_EXPONENT2_NOT_SUPPLIED]
Second prime exponent not supplied.

[CMSERR_PRIME_NOT_SUPPLIED]
Prime not supplied.

[CMSERR_PRIME1_NOT_SUPPLIED]
First prime not supplied.

[CMSERR_PRIME2_NOT_SUPPLIED]
Second prime not supplied.

[CMSERR_PRIVATE_EXPONENT_NOT_SUPPLIED]
Private exponent not supplied.

[CMSERR_PRIVATE_KEY_INFO_NOT_SUPPLIED]
Private key information not supplied.

[CMSERR_PRIVATE_KEY_NOT_SUPPLIED]
Private key structure not supplied.

[CMSERR_PRIVATE_VALUE_NOT_SUPPLIED]
Private value not supplied.

[CMSERR_PUBLIC_EXPONENT_NOT_SUPPLIED]
Public exponent not supplied.

[CMSERR_STRUCTURE_TOO_SMALL]
Size specified for supplied structure is too small.

[CMSERR_SUB_PRIME_NOT_SUPPLIED]
Sub-prime not supplied.

Usage

The **gsk_construct_private_key()** function constructs the pkcs_private_key_info from the supplied private key components. The format of the supplied components is as stored in ICSF PKCS #11 tokens.

Before calling the function, the application must initialize the size field in *private_key_factors* to the size of the gsk_private_key structure. It must also prime *private_key_factors* with the x509_algorithm_identifier, including appropriate private key components for the private key type being constructed.

The x509_algorithm_identifier in *private_key* is set with the appropriate value for the private key type when returned.

gsk_construct_private_key_rsa()

Constructs an RSA private key from its component values.

Note: This function is deprecated. Use **gsk_construct_private_key()** instead.

Format

```
#include <gskcms.h>

gsk_status gsk_construct_private_key_rsa (
                                       gsk_buffer *            modulus,
                                       gsk_buffer *            public_exponent,
                                       gsk_buffer *            private_exponent,
                                       gsk_buffer *            prime1,
                                       gsk_buffer *            prime2,
                                       gsk_buffer *            prime_exponent1,
                                       gsk_buffer *            prime_exponent2,
                                       gsk_buffer *            coefficient,
                                       pkcs_private_key_info *  private_key)
```

Parameters

modulus
Specifies the modulus (n).

public_exponent
Specifies the public exponent (e).

private_exponent
Specifies the private exponent (d).

prime1
Specifies the 1st prime (p).

prime2
Specifies the 2nd prime (q).

prime_exponent1
Specifies the private exponent d modulo p-1

prime_exponent2
Specifies the private exponent d modulo q-1.

coefficient
Specifies the CRT coefficient q^{-1} mod p.

private_key
Returns the private key

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_ELEMENTS_MISSING]
Required data element is missing.

Usage

The **gsk_construct_private_key_rsa()** function constructs pkcs_private_key_info from its RSA private key components. The pkcs_private_key_info structures x509_algorithm_identifier is set with x509_alg_rsaEncryption, while version specifies 0.

gsk_construct_public_key()

Constructs a public key from its component values

Format

```
#include <gskcms.h>

gsk_status gsk_construct_public_key(
                           gsk_public_key *         public_key_factors,
                           x509_public_key_info *   public_key)
```

Parameters

public_key_factors
Specifies the public key structure containing the key algorithm type and public key components.

public_key
Returns the public key.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
Cryptographic algorithm not supported

[CMSERR_BASE_NOT_SUPPLIED]
Base not supplied

[CMSERR_EC_PARAMETERS_NOT_SUPPLIED]
EC parameters not supplied.

[CMSERR_MODULUS_NOT_SUPPLIED]
Modulus not supplied

[CMSERR_PRIME_NOT_SUPPLIED]
Prime not supplied

[CMSERR_PUBLIC_EXPONENT_NOT_SUPPLIED]
Public exponent not supplied

[CMSERR_PUBLIC_KEY_INFO_NOT_SUPPLIED]
Public key information not supplied

[CMSERR_PUBLIC_KEY_NOT_SUPPLIED]
Public key structure not supplied

[CMSERR_PUBLIC_VALUE_NOT_SUPPLIED]
Public value not supplied

[CMSERR_STRUCTURE_TOO_SMALL]
Size specified for supplied structure is too small

[CMSERR_SUB_PRIME_NOT_SUPPLIED]
Sub-prime not supplied

Usage

The **gsk_construct_public_key()** function constructs the x509_public_key_info from the supplied public key components. The format of the supplied components is as stored in ICSF PKCS #11 tokens.

Before calling the function, the application must initialize the size field in *public_key_factors* to the size of the gsk_public_key structure. It must also prime *public_key_factors* with the x509_algorithm_identifier, including appropriate public key components for the public key type being constructed.

The x509_algorithm_identifier in *public_key* is set with the appropriate value for the public key type when returned.

gsk_construct_public_key_rsa()

Constructs an RSA public key from its component values.

Note: This function is deprecated. Use **gsk_construct_public_key()** instead.

Format

```
#include <gskcms.h>

gsk_status gsk_construct_public_key_rsa (
                                       gsk_buffer *             modulus,
                                       gsk_buffer *             exponent,
                                       x509_public_key_info *   public_key)
```

Parameters

modulus
Specifies the modulus (n).

exponent
Specifies the public exponent (e).

public_key
Returns the public key.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_ELEMENTS_MISSING]
Required data element is missing.

Usage

The **gsk_construct_public_key_rsa()** function constructs pkcs_public_key_info from its RSA public key components. The x509_public_key_info structures x509_algorithm_identifier is set with x509_alg_rsaEncryption.

gsk_construct_renewal_request()

Constructs a certification renewal request as described in RFC 2986 (tools.ietf.org/html/rfc2986).

Format

```
#include <gskcms.h>

gsk_status gsk_construct_renewal_request (
                             x509_public_key_info *  public_key,
                             pkcs_private_key_info * private_key,
                             x509_algorithm_type     signature_algorithm,
                             const char *            subject_name,
                             x509_extensions *       extensions,
                             pkcs_cert_request *     request)
```

Parameters

public_key
Specifies the public key for the certification request.

private_key
Specifies the private key for the certification request.

signature_algorithm
Specifies the signature algorithm used to sign the constructed request.

subject_name
Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

extensions
Specifies certificate extensions to be included in the certification request. Specify NULL for this parameter if no certificate extensions are provided.

request
Returns the certification renewal request as a pkcs_cert_request structure.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_X500_NO_AVA_SEP]
An attribute value separator is missing.

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_KEY MISMATCH]
The signing key type is not supported by the requested signature algorithm.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_construct_renewal_request()** routine constructs a certification renewal request and returns the constructed request in the pkcs_cert_request structure *request*.

The **gsk_encode_export_request()** routine can be called to create an export file containing the request for transmission to the certification authority.

The certification request will be signed using the key specified by the *private_key* parameter and the signature algorithm specified by the *signature_algorithm* parameter.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest – {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest - {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest - {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest - {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest - {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest - {1.2.840.10045.4.3.4}

These RSASSA-PSS [1.2.840.113549.1.1.10] signatures are supported:

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation function algorithm 1.

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

The *extensions* parameter can be used to provide certificate extensions for inclusion in the certification request. Whether or not a particular certificate extension will be included in the new certificate is determined by the certification authority.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_construct_self_signed_certificate()

Constructs a self-signed certificate and returns it to the caller.

Format

```
#include <gskcms.h>

gsk_status gsk_construct_self_signed_certificate (
                             x509_algorithm_type     signature_algorithm,
                             const_char *            subject_name,
                             int                     num_days,
                             gsk_boolean             ca_certificate,
                             x509_extensions *       extensions,
                             x509_public_key_info *  public_key,
                             pkcs_private_key_info * private_key,
                             x509_certificate *      subject_certificate)
```

Parameters

signature_algorithm
Specifies the signature algorithm used to sign the constructed certificate.

subject_name
Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

num_days
Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

ca_certificate
Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

extensions
Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

public_key
Specifies the public key for the constructed certificate.

private_key
Specifies the private key for the constructed certificate.

subject_certificate
Contains the constructed certificate.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not valid.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_SUBJECT_NAME]
The subject name is not valid.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_KEY_MISMATCH]
The signer key cannot be used to sign a certificate or the key type is not supported for the requested signature algorithm.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_UNSUPPORTED_EXPIRATION]
The expiration date exceeds February 6, 2106, at 23:59:59 UTC.

Usage

The **gsk_construct_self_signed_certificate()** routine will construct an X.509 certificate as described in RFC 5280 (tools.ietf.org/html/rfc5280). A certification authority certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have no basic constraints limitations or key usage limitations. The constructed certificate is then returned in the x509_certificate structure *subject_certificate*.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest – {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest - {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest - {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest - {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest - {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest - {1.2.840.10045.4.3.4}

These RSASSA-PSS [1.2.840.113549.1.1.10] signatures are supported:

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation function algorithm 1.

When executing in non-FIPS mode, an RSA key size must be between 512 and 4096 bits. A DSA key size must be between 512 and 2048 bits. An ECC key can be a NIST recommended named curve with a key between 192 and 521 or a Brainpool curve with a key between 160 and 512.

When executing in FIPS mode:

- Signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.
- An RSA key size must be between 1024 and 4096 bits.
- A DSA key size must be either 1024 bits or 2048 bits.
- An NIST ECC key must be a NIST recommended named curve with a key size between 192 and 521.

For more information about FIPS supported algorithms and key sizes, see "Algorithms and key sizes" on page 19.

A DSA key size of 1024 or less should specify signature algorithm x509_alg_dsaWithSha1, while a key size of 2048 bits should specify either x509_alg_dsaWithSha224 or x509_alg_dsaWithSha256 as the signature algorithm.

Note: A self-signed end-entity certificate (server or client certificate) is not recommended for use in production environments and should only be used to facilitate test environments before production. Self-signed certificates do not imply any level of security or authenticity of the certificate because, as their name implies, they are signed by the same key that is contained in the certificate. However, certificates that are signed by a certificate authority indicate that, at least at the time of signature, the certificate authority approved the information contained in the certificate.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, "System SSL and FIPS 140-2," on page 19.

gsk_construct_signed_certificate()

Constructs a signed certificate for a certificate request.

Format

```
#include <gskcms.h>

gsk_status gsk_construct_signed_certificate (
                                       pkcs_cert_key *      signer_certificate,
                                       pkcs_cert_request *  request,
                                       x509_algorithm_type  signature_algorithm,
                                       int                  num_days,
                                       gsk_boolean          ca_certificate,
                                       x509_extensions *    extensions,
                                       x509_certificate *   certificate)
```

Parameters

signer_certificate
Specifies the signing certificate with private key.

request
Specifies the PKCS #10 certification request stream in either binary DERencoded format or in Base64 format. A Base64 stream is in the local code page.

signature_algorithm
Specifies the signature algorithm used to sign the constructed certificate.

num_days
Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

ca_certificate
Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

extensions
Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

certificate
Contains the constructed signed certificate.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The key algorithm or the signature algorithm is not valid.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_ENCODING]
The certificate request stream is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_SIGNATURE]
The request signature is not correct.

CMSERR_CA_NOT_SUPPLIED[]
CA certificate is not supplied.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_EXPIRED]
The signer certificate is expired.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing certificates.

[CMSERR_ISSUER_NOT_CA]
The signer certificate is not for a certification authority.

[CMSERR_KEY_MISMATCH]
The signer certificate key cannot be used to sign a certificate or the key type is not supported for the requested signature algorithm.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_REQUEST_NOT_SUPPLIED]
Certificate request not supplied.

[CMSERR_SUBJECT_IS_CA]
The requested subject name is the same as the signer name.

[CMSERR_UNSUPPORTED_EXPIRATION]
The expiration date exceeds February 6, 2106, at 23:59:59 UTC.

Usage

The **gsk_construct_signed_certificate()** routine will construct an X.509 certificate as described in RFC 5280 (tools.ietf.org/html/rfc5280). The new certificate will be signed using the certificate specified by the *signer_certificate* parameter. A certification authority certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions which allow the certificate to be used for authentication, digital signatures, and data encryption (except for a DSA key which cannot be used for data encryption). The certificate expiration will be set to the earlier of the requested expiration date and the expiration date of the signing certificate.

The signing certificate must have an associated private key, the Basic Constraints extension must either be omitted or must have the CA indicator set, and the KeyUsage extension must either be omitted or must allow signing certificates.

A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An AuthorityKeyIdentifier extension will be created if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage or BasicConstraints extension provided by the application will replace the default extension constructed for the certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.

Certificate extensions can also be contained within the certification request. A certificate extension supplied by the application will override a certificate extension of the same type contained in the certification request. The certificate extension found in the certification request will be copied unmodified to the new certificate with these exceptions:

- The AuthorityInfoAccess, AuthorityKeyIdentifier, BasicConstraints, CrlDistributionPoints, IssuerAltName, NameConstraints, PolicyConstraints, PolicyMappings, and PrivateKeyUsagePeriod extensions will not be copied.
- The keyCertSign and crlSign flags in the KeyUsage extension will be modified based upon the value of the *ca_certificate* parameter.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest – {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest – {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest – {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest – {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest – {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest – {1.2.840.10045.4.3.4}

These RSASSA-PSS [1.2.840.113549.1.1.10] signatures are supported:

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation function algorithm 1.

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

No certification path validation is performed by the **gsk_construct_signed_certificate()** routine. An error will be returned if the requested subject name is the same as the subject name in the signing certificate.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_construct_signed_crl()

Creates a signed certificate revocation list in DER or base64 encoded formats.

Format

```
#include <gskcms.h>

gsk_status gsk_construct_signed_crl (
                            gskdb_record *                record,
                            gsk_time                      this_update,
                            gsk_time                      next_update,
                            x509_algorithm_type           signature_algorithm,
                            int                           crl_number,
                            x509_revoked_certificates *   revoked_certificates,
                            x509_extensions *             extensions,
                            gsk_crl_format                crl_format,
                            gsk_buffer *                  signed_crl)
```

Parameters

record
Specifies the database record of the signing certificate to be used to sign the certificate revocation list.

this_update
Specifies the time as the number of seconds since the POSIX epoch to use for the thisUpdate field in the signed certificate revocation list.

next_update
Specifies the time as the number of seconds since the POSIX epoch to use for the nextUpdate field in the signed certificate revocation list. A value of 0 indicates that the signed certificate revocation list does not contain the optional nextUpdate time.

signature_algorithm
Specifies the signature algorithm to be used for the CRL signature.

crl_number
Specifies the CRL number. Each CRL is numbered and each successive revocation list has a larger CRL number than all previous revocation lists.

revoked_certificates
Specifies the revoked list of certificates to be included in the CRL. This list consists of the certificate serial numbers and not the actual certificates.

extensions
Specifies the CRL extensions for the new CRL. Specify NULL for this parameter if no CRL extensions are supplied.

crl_format
Specifies the format of the signed certificate revocation list that is returned in the *signed_crl* parameter.

- If set to **gsk_crl_format_base64_encode**, the signed CRL is returned in base64 encoded format.
- If set to **gsk_crl_format_der_encode**, the signed certificate revocation list is returned in DER encoded format.

signed_crl
Returns the signed certificate revocation list in the format that is specified by *crl_format*.

- If *crl_format* is set to **gsk_crl_format_base64_encode**, the Base64 stream is in the local code page.
- If *crl_format* is set to **gsk_crl_format_der_encode**, the stream is in binary.

The application should call the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve is not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_EXPIRED]
The signer certificate is expired.

[CMSERR_FORMAT_NOT_VALID]
An unsupported certificate revocation list encoding format is specified.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing a CRL.

[CMSERR_INVALID_NUMERIC_VALUE]
The *this_update* time must be less than the *next_update* time or the signing certificate's expiration date.

[CMSERR_ISSUER_NOT_CA]
The signer certificate is not for a certification authority.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

Usage

The **gsk_construct_signed_crl()** routine generates an X.509 certificate revocation list (CRL) as described in RFC 5280 (tools.ietf.org/html/rfc5280). The new CRL is signed by using the certificate that is specified by the *record* parameter and the signature algorithm that is specified by the *signature_algorithm* parameter. The format of the signed CRL is specified by the *crl_format* parameter.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest – {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest – {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest – {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest – {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest – {1.2.840.10045.4.3.4}

These RSASSA-PSS [1.2.840.113549.1.1.10] signatures are supported:

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation function algorithm 1.

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

The *this_update* parameter indicates the time as the number of seconds since the POSIX epoch (January 1, 1970), at which time the signed CRL is valid. The time in seconds specified for the thisUpdate in the signed CRL cannot exceed February 6, 2106, at 23:59:59 UTC.

The *next_update* parameter indicates the time as the number of seconds since the POSIX epoch (January 1, 1970), at which time the signed CRL is no longer valid. The time in seconds specified for the nextUpdate in the signed CRL cannot exceed February 6, 2106, at 23:59:59 UTC. If the *next_update* parameter is set to a non-zero number, the expiration time of the signed CRL is set to the earlier of the requested date and time and the expiration of the signing certificate. If the *next_update* parameter is set to 0, the signed CRL does not contain an expiration time. The CRL expiration time must be later than the time specified in the *this_update* parameter.

The signing certificate that is specified by the *record* parameter must have an associated private key, the BasicConstraints extension must either be omitted or must have the CA indicator set, and the KeyUsage extension must either be omitted or must allow signing certificate revocation lists.

The CRL has a CRLNumber extension that contains the value that is specified by the *crl_number* parameter. It also has an AuthorityKeyIdentifier extension if the signing certificate has a SubjectKeyIdentifier extension. The application can supply extra extensions through the *extensions* parameter. An AuthorityKeyIdentifier or CRLNumber extension that is provided by the application replaces the default extension that is created for the CRL.

The *crl_format* parameter indicates the format of the returned signed CRL.

- If *crl_format* is set to **gsk_crl_format_base64_encode**, the *signed_crl* is base64 encoded in the local code page and includes the encoding header and footer lines.
- If *crl_format* is set to **gsk_crl_format_der_encode**, the *signed_crl* contains the CRL in binary format without the encoding header and footer lines.

No certification path validation is performed by the **gsk_construct_signed_crl()** routine.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_copy_attributes_signers()

Copies a gsk_attributes_signers structure.

Format

```
#include <gskcms.h>

gsk_status gsk_copy_attributes_signers (
                               gsk_attributes_signers *  in_attributesSigners,
                               gsk_attributes_signers *  out_attributesSigners)
```

Parameters

in_attributesSigners
Specifies the source gsk_attributes_signers structure.

out_attributesSigners
Specifies the destination gsk_attributes_signers structure. The application should call the **gsk_free_attributes_signers()** routine when the gsk_attributes_signers structure is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_copy_attributes_signers()** routine will allocate the output gsk_attributes_signers structure and then copy the input gsk_attributes_signers structure to the output gsk_attributes_signers structure. Storage for the base gsk_attributes_signers structure (*in_attributesSigners*) is provided by the application.

gsk_copy_buffer()

Copies a buffer.

Format

```
#include <gskcms.h>

gsk_status gsk_copy_buffer (
                             gsk_buffer *    in_buffer,
                             gsk_buffer *    out_buffer)
```

Parameters

in_buffer
Specifies the source buffer.

out_buffer
Specifies the destination buffer. The application should call the **gsk_free_buffer()** routine when the buffer is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_copy_buffer()** routine will allocate the output buffer and then copy the input buffer to the output buffer. Storage for the base gsk_buffer structure is provided by the caller.

gsk_copy_certificate()

Copies an X.509 certificate.

Format

```
#include <gskcms.h>

gsk_status gsk_copy_certificate (
                                  x509_certificate *         in_certificate,
                                  x509_certificate *         out_certificate)
```

Parameters

in_certificate
Specifies the source certificate.

out_certificate
Specifies the destination certificate. The application should call the **gsk_free_certificate()** routine when the certificate is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_copy_certificate()** routine will allocate the output certificate and then copy the input certificate to the output certificate. Storage for the base x509_certificate structure is provided by the caller.

gsk_copy_certificate_extension()

Copies an X.509 certificate extension.

Format

```
#include <gskcms.h>

gsk_status gsk_copy_certificate_extension (
                                 x509_extension *        in_extension,
                                 x509_extension *        out_extension)
```

Parameters

in_extension
Specifies the source certificate extension.

out_extension
Specifies the destination certificate extension. The application should call the **gsk_free_certificate_extension()** routine when the extension is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_copy_certificate_extension()** routine will allocate the output certificate extension and then copy the input certificate extension to the output certificate extension. Storage for the base x509_extension structure is provided by the caller.

gsk_copy_certification_request()

Copies a PKCS #10 certification request.

Format

```
#include <gskcms.h>

gsk_status gsk_copy_certification_request (
                                  pkcs_cert_request *        in_request,
                                  pkcs_cert_request *        out_request)
```

Parameters

in_request
Specifies the source certification request.

out_request
Specifies the destination certification request. The application should call the **gsk_free_certification_request()** routine when the certification request is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_copy_certification_request()** routine will allocate the output certification request and then copy the input certification request to the output certification request. Storage for the base pkcs_cert_request structure is provided by the application.

gsk_copy_content_info()

Copies PKCS #7 content information.

Format

```
#include <gskcms.h>

gsk_status gsk_copy_content_info (
                                   pkcs_content_info *         in_info,
                                   pkcs_content_info *         out_info)
```

Parameters

in_info
Specifies the source content information.

out_info
Specifies the destination content information. The application should call the **gsk_free_content_info()** routine when the content information is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_copy_content_info()** routine will allocate the output content information and then copy the input content information to the output content information. Storage for the base pkcs_content_info structure is provided by the application.

gsk_copy_crl()

Copies an X.509 certificate revocation list.

Format

```
#include <gskcms.h>

gsk_status gsk_copy_crl (
                          x509_crl *          in_crl,
                          x509_crl *          out_crl)
```

Parameters

in_crl
Specifies the source certificate revocation list.

out_crl
Specifies the destination certificate revocation list. The application should call the **gsk_free_crl()** routine when the certificate revocation list is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_copy_crl()** routine will allocate the output certificate revocation list and then copy the input list to the output list. Storage for the base x509_crl structure is provided by the caller.

gsk_copy_name()

Copies an X.509 name.

Format

```
#include <gskcms.h>

gsk_status gsk_copy_name (
                           x509_name *          in_name,
                           x509_name *          out_name)
```

Parameters

in_name
Specifies the source name.

out_name
Specifies the destination name. The application should call the **gsk_free_name()** routine when the name is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_copy_name()** routine will allocate the output name and then copy the input name to the output name. Storage for the base x509_name structure is provided by the caller.

gsk_copy_private_key_info()

Copies the private key information.

Format

```
#include <gskcms.h>

gsk_status gsk_copy_private_key_info (
                                       pkcs_private_key_info *         in_info,
                                       pkcs_private_key_info *         out_info)
```

Parameters

in_info
Specifies the source private key information.

out_info
Specifies the destination private key information. The application should call the **gsk_free_private_key_info()** routine when the private key is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_copy_private_key_info()** routine will allocate the output private key and then copy the input key to the output key. Storage for the base pkcs_private_key_info structure is provided by the caller.

gsk_copy_public_key_info()

Copies the public key information.

Format

```
#include <gskcms.h>

gsk_status gsk_copy_public_key_info (
                                      x509_public_key_info *        in_info,
                                      x509_public_key_info *        out_info)
```

Parameters

in_info
Specifies the source public key information.

out_info
Specifies the destination public key information. The application should call the **gsk_free_public_key_info()** routine when the public key is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_copy_public_key_info()** routine will allocate the output public key and then copy the input key to the output key. Storage for the base x509_public_key_info structure is provided by the caller.

gsk_copy_record()

Copies a database record.

Format

```
#include <gskcms.h>

gsk_status gsk_copy_record (
                             gskdb_record *      in_record,
                             gskdb_record **     out_record)
```

Parameters

in_record
Specifies the source record.

out_record
Returns the copied record. The application should call the **gsk_free_record()** routine when the record is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_copy_record()** routine will allocate the output record and then copy the input record to the output record. The address of the copied record will then be returned to the application.

gsk_create_certification_request()

Creates a PKCS #10 certification request as described in RFC 2986 (tools.ietf.org/html/rfc2986).

Format

```
#include <gskcms.h>

gsk_status gsk_create_certification_request (
                                 gsk_handle            db_handle,
                                 const char *          label,
                                 x509_algorithm_type   signature_algorithm,
                                 int                   key_size,
                                 const char *          subject_name,
                                 x509_extensions *     extensions)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine. This must be a request database and not a key database.

label
Specifies the label for the new database record. The label is specified in the local code page.

signature_algorithm
Specifies the signature algorithm for the certificate.

key_size
Specifies the key size in bits.

subject_name
Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

extensions
Specifies certificate extensions to be included in the certification request. Specify NULL for this parameter if no certificate extensions are provided.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_FIPS_KEY_PAIR_CONSISTENCY]
FIPS mode key generation failed pair-wise consistency check.

[CMSERR_ICSF_CLEAR_KEY_SUPPORT_NOT_AVAILABLE]
Clear key support not available due to ICSF key policy.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certification requests.

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_create_certification_request()** routine creates a PKCS #10 certification request. The request is then stored in the request database. The **gsk_export_certification_request()** routine can be called to create an export file containing the request for transmission to the certification authority.

The **gsk_create_certification_request()** routine is similar to the **gsk_create_renewal_request()** routine. Both routines create a PKCS #10 certification request. The difference is the **gsk_create_certification_request()** routine generates a new public/private key pair while the **gsk_create_renewal_request()** routine uses the public/private key pair provided by the application.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest – {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest - {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest - {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest - {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest - {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest - {1.2.840.10045.4.3.4}

These RSASSA-PSS [1.2.840.113549.1.1.10] signatures are supported:

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation function algorithm 1.

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

If not in FIPS mode, an RSA key size must be between 512 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be between 512 and 2048 bits. Key sizes of between 512 and 1024 bits are rounded up to a multiple of 64, key size 2048 must be explicitly specified as such. A key size of 1024 or less should specify signature algorithm x509_alg_dsaWithSha1, while a key size of 2048 bits should specify either x509_alg_dsaWithSha224 or x509_alg_dsaWithSha256 as the signature algorithm.

In FIPS mode, an RSA key size must be between 1024 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be either 1024 bits or 2048 bits. A key size of 1024 bits should specify signature algorithm x509_alg_dsaWithSha1, while a key size of 2048 bits should specify either x509_alg_dsaWithSha224 or x509_alg_dsaWithSha256 as the signature algorithm.

For an ECC key the key size will determine the default named curve that will be used for the public/private key pair, as specified in Table 3 on page 13. In FIPS mode, only NIST recommended that curves are supported. To specify a specific supported elliptic curve, use **gsk_construct_renewal_request()** to create a certificate request.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

The *extensions* parameter can be used to provide certificate extensions for inclusion in the certification request. Whether or not a particular certificate extension will be included in the new certificate is determined by the certification authority.

The database must be open for update in order to add the new request. The database file is updated as part of the **gsk_create_certification_request()** processing. A temporary database file is created using the same name as the database file with ".new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_create_database()

Creates a key or request database.

Format

```
#include <gskcms.h>

gsk_status gsk_create_database (
                            char *                  filename,
                            char *                  password,
                            gskdb_database_type     db_type,
                            gsk_size                record_length,
                            gsk_time                pwd_expiration,
                            gsk_handle *            db_handle)
```

Parameters

filename
: Specifies the database file name in the local code page. The length of the fully-qualified file name cannot exceed 251.

password
: Specifies the database password in the local code page. The password must consist of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string. The user will be prompted to enter the password if NULL is specified for this parameter.

db_type
: Specifies the database type and it must be gskdb_dbtype_key for a key database populated with common CA certificates, gskdb_dbtype_key_empty for an empty key database, or gskdb_dbtype_request for a certification request database.

record_length
: Specifies the database record length. The default record length will be used if zero is specified for this parameter. All records in the database will have this length. The minimum record length is 2500. The default record length is 5000.

pwd_expiration
: Specifies the database password expiration time as the number of seconds since the POSIX epoch. A value of 0 indicates that the password does not expire.

db_handle
: Returns the database handle. The application should call the **gsk_close_database()** routine when it no longer needs access to the database.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_FILENAME]
: The database file name is not valid.

[CMSERR_DB_EXISTS]
: The database already exists.

[CMSERR_INCORRECT_DBTYPE]
: The database type is not valid.

[CMSERR_IO_CANCELED]
: The user canceled the password prompt.

[CMSERR_IO_ERROR]
: An input/output request failed.

[CMSERR_LENGTH_TOO_SMALL]
The record length is less than the minimum value.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_OPEN_FAILED]
Unable to open the key database.

Usage

The **gsk_create_database()** routine will create a key or request database. The database must not already exist. When gskdb_dbtype_key_empty is specified, an empty new key database is created. When gskdb_dbtype_key is specified, a new key database containing an initial set of Certificate Authority certificates for use in validating certificate signatures is created. If this function is called while in FIPS mode with the FIPS state set to either level 2 or level 3, the new key database will not be populated with any initial Certificate Authority certificates, even if gskdb_dbtype_key is specified.

If this function is called while executing in FIPS mode, the new database will meet FIPS 140-2 criteria. Such a database:

- Can be read while executing in FIPS mode and when not in FIPS mode.
- Can be updated only when executing in FIPS mode.

A database created while not executing in FIPS mode:

- Can be updated or read when not in FIPS mode
- Cannot be used while executing in FIPS mode

gsk_create_database_renewal_request()

Creates a PKCS #10 certification renewal request.

Format

```
#include <gskcms.h>

gsk_status gsk_create_database_renewal_request (
                                 gsk_handle                db_handle,
                                 const char *              label,
                                 x509_public_key_info *    public_key,
                                 pkcs_private_key_info *   private_key,
                                 x509_algorithm_type       signature_algorithm,
                                 const char *              subject_name,
                                 x509_extensions *         extensions)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine. This must be a request database and not a key database.

label
Specifies the label for the request database record. The label is specified in the local code page.

public_key
Specifies the public key for the certification request.

private_key
Specifies the private key for the certification request.

signature_algorithm
Specifies the signature algorithm to be used for the request signature.

subject_name
Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

extensions
Specifies certificate extensions to be included in the certification request. Specify NULL for this parameter if no certificate extensions are provided.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not valid.

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certification requests.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_KEY_MISMATCH]
The supplied private key cannot be used to sign a certificate or the private key type is not supported for the requested signature algorithm.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_PRIVATE_KEY_INFO_NOT_SUPPLIED]
Private key information not supplied.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_create_database_renewal_request()** routine creates a certification request as described in RFC 2986 (tools.ietf.org/html/rfc2986). The request is then stored in the request database. The **gsk_export_certification_request()** routine can be called to create an export file containing the request for transmission to the certification authority.

The **gsk_create_database_renewal_request()** routine is similar to the **gsk_create_certification_request()** routine. Both routines create a PKCS #10 certification request. The difference is the **gsk_create_certification_request()** routine generates a new public/private key pair while the **gsk_create_database_renewal_request()** routine uses the public/private key pair provided by the application.

The renewal request will be signed using the key specified by the *private_key* parameter and the signature algorithm specified by the *signature_algorithm* parameter.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest – {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest - {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest - {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest - {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest - {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest - {1.2.840.10045.4.3.4}

These RSASSA-PSS [1.2.840.113549.1.1.10] signatures are supported:

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation function algorithm 1.

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_create_database_signed_certificate()

Creates a signed certificate as part of a set of certificates.

Format

```
#include <gskcms.h>

gsk_status gsk_create_database_signed_certificate (
                                  gsk_handle              db_handle,
                                  const char *            ca_label,
                                  const char *            record_label,
                                  x509_algorithm_type     key_algorithm,
                                  int                     key_size,
                                  gsk_buffer *            key_parameters,
                                  x509_algorithm_type     signature_algorithm,
                                  const char *            subject_name,
                                  int                     num_days,
                                  gsk_boolean             ca_certificate,
                                  x509_extensions *       extensions)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine. This must be a key database and not a request database.

ca_label
Specifies the label of the certificate to be used to sign the new certificate. The key usage for the certificate must allow certificate signing. The label is specified in the local code page.

record_label
Specifies the label for the new database record. The label is specified in the local code page.

key_algorithm
Specifies the certificate key algorithm.

key_size
Specifies the certificate key size in bits.

key_parameters
Specifies the key generation parameters. Specify NULL for this parameter if the key algorithm does not require any key parameters.

signature_algorithm
Specifies the signature algorithm used for the certificate signature.

subject_name
Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

num_days
Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

ca_certificate
Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

extensions
Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The key algorithm or the signature algorithm is not valid.

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_LABEL]
The record label or CA certificate label is not valid.

[CMSERR_BAD_SUBJECT_NAME]
The subject name is not valid.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_EXPIRED]
The signer certificate is expired.

[CMSERR_FIPS_KEY_PAIR_CONSISTENCY]
FIPS mode key generation failed pair-wise consistency check.

[CMSERR_ICSF_CLEAR_KEY_SUPPORT_NOT_AVAILABLE]
Clear key support not available due to ICSF key policy.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_TYPE]
Incorrect key algorithm

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing certificates.

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_IO_ERROR]
Unable to read or write a database record.

[CMSERR_ISSUER_NOT_CA]
The signer certificate is not for a certification authority.

[CMSERR_KEY_MISMATCH]
The signer certificate key cannot be used to sign a certificate.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_SUBJECT_IS_CA]
The requested subject name is the same as the signer name.

[CMSERR_UNSUPPORTED_EXPIRATION]
The expiration date exceeds February 6, 2106, at 23:59:59 UTC.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_create_database_signed_certificate()** routine will generate an X.509 certificate as described in RFC 5280 (tools.ietf.org/html/rfc5280). The certificate will be signed using an existing certificate as specified by the *ca_label* parameter and the signature algorithm specified by the *signature_algorithm* parameter.

- If the specified certificate key is a Diffie-Hellman key, the *signature_algorithm* must specify either an RSA or a DSA signature.
- If the specified certificate key is an ECC key, the *signature_algorithm* cannot specify a DSA signature.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest – {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest – {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest – {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest – {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest – {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest – {1.2.840.10045.4.3.4}

These RSASSA-PSS [1.2.840.113549.1.1.10] signatures are supported:

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation function algorithm 1.

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

A certification authority (CA) certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions as follows:

- An RSA key can be used for authentication, digital signature, and data encryption.
- A DSS key can be used for authentication and digital signature.
- A Diffie-Hellman key can be used for key agreement.
- An ECC key can be used for authentication, digital signature, and key agreement.

The new certificate will be stored in the key database using the supplied record label. The **gsk_export_certificate()** routine can be called to create an export file containing the certificate for transmission to another system.

The following key algorithms are supported:

x509_alg_rsaEncryption
RSA encryption - {1.2.840.113549.1.1.1}

x509_alg_idDsa
Digital Signature Standard (DSS) - {1.2.840.10040.4.1}

x509_alg_dhPublicNumber
Diffie-Hellman (DH) - {1.2.840.10046.2.1}

x509_alg_ecPublicKey
Elliptic Curve Public Key (ECC) - {1.2.840.10045.2.1}

RSA keys

- Can be used for both CA certificates and end user certificates
- Key size when not in FIPS mode is between 512 and 4096 bits rounded up to a multiple of 16
- Key size in FIPS mode is between 1024 and 4096 bits rounded up to a multiple of 16
- No key parameters

DSS keys

- Can be used for both CA certificates and end user certificates.
- Key sizes of between 512 and 1024 bits when in non-FIPS mode are rounded up to a multiple of 64.

- Key sizes less than 1024 bits can only be generated in non-FIPS mode and are generated according to FIPS 186-2.
- Key sizes of 1024 or 2048 bits are generated according to FIPS 186-4 in both FIPS mode and non-FIPS mode. These are the only valid key sizes in FIPS mode.
- A key size of 1024 or less should specify x509_alg_dsaWithSha1 as the signature algorithm, while a key size of 2048 bits should specify either x509_alg_dsaWithSha224 or x509_alg_dsaWithSha256 as the signature algorithm.
- Key parameters encoded as an ASN.1 sequence consisting of the prime p, the prime divisor q, and the generator g. For 1024-bit and 2048-bit keys, see FIPS 186-4: Digital Signature Standard (DSS) (nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf) for more information about the key parameters, for smaller key sizes see FIPS 186-2: Digital Signature Standard (DSS) (csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf). Note that key parameters that contain a p of 2048 bits and a q of 160 bits do not conform to FIPS 186-4 and are not supported. The **gsk_generate_key_parameters()** routine can be used to generate the key parameters.

DH keys

- Can be used only for end user certificates
- Can only be signed using a certificate containing either an RSA or DSA key
- Key size when not in FIPS mode is between 512 and 2048 bits rounded up to a multiple of 64
- Key size in FIPS mode of 2048 bits
- Key parameters encoded as an ASN.1 sequence consisting of the prime P, the base G, and optionally the subprime Q and the subgroup factor J. See RFC 2631 (tools.ietf.org/html/rfc2631) for more information about the key parameters for non-FIPS mode, and see *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for FIPS mode. The **gsk_generate_key_parameters()** routine can be used to generate the key parameters.

ECC keys

- Can be used for both CA certificates and end user certificates.
- The ECC named curve used to generate the ECC key pair can be specified using either the *key_parameters* buffer or the *key_size* parameter. If the *key_parameters* buffer is supplied the *key_size* parameter will be ignored.
- The *key_parameters* buffer must contain ASN.1 encoded ECC parameters, or be NULL.
- If the *key_parameters* buffer is not supplied, the *key_size* parameter will be rounded up to the nearest supported key size and the default EC named curve for that key size will be used, as specified in Table 3 on page 13.
- In FIPS mode, only NIST recommended curves are supported.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

A CA certificate will have SubjectKeyIdentifier, KeyUsage, and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An AuthorityKeyIdentifier extension will be created if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage, or BasicConstraints extension provided by the application will replace the default extension constructed for the certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.

The database must be open for update in order to add the new certificate. The database file is updated as part of the **gsk_create_database_signed_certificate()** processing. A temporary database file is created using the same name as the database file with ".new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_create_renewal_request()

Creates a PKCS #10 certification renewal request.

This function is deprecated. Use **gsk_create_database_renewal_request()** instead.

Format

```
#include <gskcms.h>

gsk_status gsk_create_renewal_request (
                                 gsk_handle                  db_handle,
                                 const char *                label,
                                 x509_public_key_info *      public_key,
                                 pkcs_private_key_info *     private_key,
                                 const char *                subject_name,
                                 x509_extentions *           extensions)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine. This must be a request database and not a key database.

label
Specifies the label for the request database record. The label is specified in the local code page.

public_key
Specifies the public key for the certification request.

private_key
Specifies the private key for the certification request.

subject_name
Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

extensions
Specifies certificate extensions to be included in the certification request. Specify NULL for this parameter if no certificate extensions are provided.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certification requests.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_PRIVATE_KEY_INFO_NOT_SUPPLIED]
Private key information not supplied.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_create_renewal_request()** routine creates a certification request as described in RFC 2986 (tools.ietf.org/html/rfc2986). The request is then stored in the request database. The **gsk_export_certification_request()** routine can be called to create an export file containing the request for transmission to the certification authority.

The **gsk_create_renewal_request()** routine is similar to the **gsk_create_certification_request()** routine. Both routines create a PKCS #10 certification request. The difference is the **gsk_create_certification_request()** routine generates a new public/private key pair while the **gsk_create_renewal_request()** routine uses the public/private key pair provided by the application.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

The extensions parameter can be used to provide certificate extensions for inclusion in the certification request. Whether or not a particular certificate extension will be included in the new certificate is determined by the certification authority.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

The signature algorithm used when signing the certificate request is derived from the key algorithm of the provided private key.

- For a RSA key algorithm, the signature digest type is SHA-1 and the signature is based on RSA encryption.
- For a DSA key algorithm, when using a 1024-bit DSA key, the digest type is SHA-1. When using a 2048-bit DSA key, the digest type is SHA-256.

- For a ECC key algorithm, the signature digest type is the suggested digest for the key size, as specified in Table 2 on page 13.

gsk_create_revocation_source()

Creates an OCSP, HTTP CRL, or an extended LDAP CRL revocation source.

Format

```
#include <gskcms.h>

gsk_status gsk_create_revocation_source (
                                   gskdb_source *        source,
                                   gsk_handle *          revocation_handle)
```

Parameters

source
Specifies the parameters needed for the revocation source handle to be created.

revocation_handle
Returns the revocation data source handle. The application should call the **gsk_free_revocation_source()** routine when it no longer needs access to the revocation source.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ATTRIBUTE_INVALID_ENUMERATION]
The enumeration value specified in the source is not valid.

[CMSERR_BAD_HANDLE]
The source handle is NULL or the type field within the source handle is not valid.

[CMSERR_BAD_SIG_ALG_PAIR]
Signature algorithms pairs list is not valid.

[CMSERR_INVALID_NUMERIC_VALUE]
Numeric value is not valid.

[CMSERR_MUTEX_ERROR]
Mutex request failed.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_OCSP_REQUEST_SIGALG_NOT_VALID]
OCSP request signature algorithm pair is not valid.

[CMSERR_REQUIRED_PARAMETER_NOTSET]
Required parameter is not set.

Usage

The **gsk_create_revocation_source()** routine creates a revocation data source to be used in the **gsk_validate_certificate()** or **gsk_validate_certificate_mode()** routines.

The source field has support for three different sources:

LDAP extended CRL support
This allows for a CRL to be obtained from an LDAP server.

HTTP CDP (Certificate Distribution Point) CRL support
This allows for a CRL to be obtained from an HTTP server.

OCSP support
This allows for certificate revocation information to be obtained from an OCSP responder.

The *gskdb_source* structure is defined in the **gskcms.h** include file. This structure contains a source type and a union which contains an embedded source structure. The *source_type* parameter defines the revocation source type and the embedded source structure contains the enablement information. The embedded source structure should be set to binary zeros and then populated with relevant information for the data source. Unused and reserved fields must contain binary zeros.

Extended LDAP directory source

If an extended LDAP directory source is specified in the *gskdb_source* structure, this indicates that an extended LDAP directory source is to be created. The parameters in the *gskdb_extended_directory_source* structure are in Table 17 on page 250.

Table 17. gskdb_extended_directory_source parameters

Parameter name	Description
crlCacheEntryMaxSize	Specifies the maximum size in bytes of a CRL that is allowed to be stored in the LDAP CRL cache. The valid cache entry sizes are 0 through 32000. A size of 0 means there is no limit on the size of the CRL.
crlCacheSize	Specifies the maximum number of CRLs that may reside at one time in the LDAP CRL cache. The valid cache sizes are -1 through 32000. If caching is desired, this parameter should be set to a value greater than 0 or -1. When set to a non-zero value, CRLs are only cached if they contain an nextUpdate value that is later than the current time. The cached CRL will stay in the cache based upon its nextUpdate time value. A value of -1 means there is no limit on the cache size. A value of 0 disables caching.
crlCacheTempCRL	Specifies if a temporary CRL is to be added to the LDAP CRL cache when the CRL does not reside on the LDAP server: **TRUE** Add entry. **FALSE** Do not add entry.
crlCacheTempCRLTimeout	Specifies the amount of time in hours that a temporary CRL can reside in the LDAP CRL cache. Valid hours are 1 through 720. This parameter should be set to a non-zero value when *crlCacheTempCRL* is set to TRUE.
crlSecurityLevel	Specifies the security level when attempting to contact an LDAP server for the certificate revocation list (CRL). Supported security level are: • GSKCMS_CRL_SECURITY_LEVEL_LOW • GSKCMS_CRL_SECURITY_LEVEL_MEDIUM • GSKCMS_CRL_SECURITY_LEVEL_HIGH See "gsk_set_directory_enum()" on page 474 for more information about the supported security levels.
ldapPassword	Specifies the password for the LDAP user (if one was specified).
ldapPort	Specifies the LDAP port for the LDAP server. Valid ports are 0 through 65535. A value of 0 means the default LDAP port of 389 is used.
ldapResponseTimeout	Specifies the time limit in seconds to wait for a response from the LDAP server. The valid time limits are 0 through 43200 seconds (12 hours). A value of 0 means there is no time limit.

Table 17. gskdb_extended_directory_source parameters (continued)

Parameter name	Description
ldapServerName	Specifies one or more blank-separated LDAP server host names.
ldapUser	Specifies the LDAP bind distinguished name (DN) used to authenticate with the LDAP server. If this is set to NULL, an anonymous authentication is used to authenticate with the LDAP server.

CDP source

If an CDP source is specified in the *gskdb_source* structure, this indicates that an HTTP CDP data source is to be created. The parameters in the *gskdb_cdp_source* structure are in Table 18 on page 251.

Table 18. gskdb_cdp_source structure parameters

Parameter name	Description
cdpEnableFlags	Enables the usage of the CDP extension for HTTP URIs. *cdpEnableFlags* must be set to either GSKCMS_CDP_ENABLE_HTTP or GSKCMS_CDP_ENABLE_ANY.
httpCdpCacheEntryMaxSize	Specifies the maximum size in bytes of a CRL that is allowed to be stored in the HTTP CDP CRL cache. The valid sizes are 0 through 2147483647. A size of 0 means there is no limit on the size of the CRL.
httpCdpCacheSize	Specifies the maximum number of CRLs that are allowed to be stored in the HTTP CDP CRL cache. The valid sizes are 0 through 32000. If caching is desired, this parameter should be set to a value greater than 0. When set to a non-zero value, CRLs are only cached if they contain an nextUpdate value that is later than the current time. The cached CRL will stay in the cache based upon its nextUpdate time value. A value of 0 disables caching.
httpCdpMaxResponseSize	Specifies the maximum response size in bytes that the application will accept from an HTTP server. The valid sizes are 0 through 2147483647. A size of 0 means there is no limit on the size of the response.
httpCdpProxyServerName	Specifies an HTTP proxy server to use when making a connection to obtain an HTTP CRL. A value of NULL indicates no proxy server. When non-NULL, the proxy server hostname or IP address must be specified. When specified, the *httpCdpProxyServerPort* must be set to a value between 1 and 65535.
httpCdpProxyServerPort	Specifies the HTTP proxy server port to use when making a connection to obtain an HTTP CRL. This port is only used when *httpCdpProxyServerName* is not NULL. Valid ports are 1 through 65535.
httpCdpResponseTimeout	Specifies the time in seconds to wait for a response from an HTTP server. The valid time limits are 0 through 43200 seconds (12 hours). A value of 0 means there is no time limit.

OCSP source

If an OCSP source is specified in the *gskdb_source* structure, this indicates that an OCSP data source is to be created. The *ocspEnable* parameter must be set to TRUE or *ocspURL* must be specified for an OCSP revocation source to be successfully created. The parameters in the *gskdb_ocsp_source* structure are in Table 19 on page 252.

Table 19. gskdb_ocsp_source structure parameters

Parameter name	Description
ocspCacheEntryMaxSize	Specifies the maximum number of cached certificate statuses to store under an OCSP cache entry block. An OCSP cache entry block is used to store the certificate statuses of certificates with the same issuer's name, issuer's key name hash, and the URL of the OCSP responder contacted. The size must be between 0 and 32000 and must be less than or equal to the size specified for *ocspCacheSize*. This size is rounded up to the nearest power of 16. The size must greater than or equal to 0. A size of 0 means there is no limit on the number of entries stored under an OCSP cache entry block.
ocspCacheSize	Specifies the maximum number of OCSP certificate statuses to store in the OCSP cache. This number is rounded up to the nearest power of 16. The size must be between 0 and 32000. If caching is desired, this parameter should be set to a value greater than 0. A value of 0 disables caching.
ocspCheckNonce	Specifies if there should be a nonce value included in the OCSP response: **TRUE** Nonce in the OCSP response is checked to verify that it is the same as the one that was sent on the OCSP request. **FALSE** No nonce checking is performed. If set to TRUE, *ocspNonceSize* must be greater than or equal to 8.
ocspDbHandle	Specifies the database handle returned by the **gsk_open_database()** routine or the **gsk_open_keyring()** routine that contains the certificate to be used to sign OCSP requests. This field is required when a non-NULL label is provided in the *ocspReqLabel* field.
ocspEnable	Specifies if the certificate's AIA extension is queried to obtain the URI values for the OCSP responder to contact: **TRUE** The URI of the OCSP responder is retrieved from the certificate's AIA extension. **FALSE** The OCSP responder is not obtained from the AIA extension.
ocspGenerateNonce	Specifies if a nonce should be generated and sent in the OCSP request: **TRUE** The nonce is sent. **FALSE** The nonce is not sent. If set to TRUE, *ocspNonceSize* must be greater than or equal to 8.
ocspMaxResponseSize	Specifies the maximum response size in bytes to be accepted from an OCSP responder. The valid response sizes are 0 through 2147483647 and defaults to 20480. A size of 0 means there is no limit on the size of the response.
ocspNonceSize	Specifies the size of the nonce that will be sent to the OCSP responder when nonce support is enabled (*ocspGenerateNonce* is TRUE). The valid nonce sizes are 8 through 256 and defaults to 8.

Table 19. gskdb_ocsp_source structure parameters (continued)

Parameter name	Description
ocspProxyServerName	Specifies an OCSP proxy server to use when making a connection to obtain an OCSP response. A value of NULL indicates no proxy server. When non-NULL, the proxy server hostname or IP address must be specified. When specified, the *ocspProxyServerPort* must be set to a value between 1 and 65535.
ocspProxyServerPort	Specifies the OCSP proxy server port to use when making a connection to obtain an OCSP response. This field is only used when *ocspProxyServerName* is not NULL. Valid ports are 1 through 65535.
ocspReqLabel	Specifies the label of the certificate to be used to sign OCSP requests. The label is specified in the local code page. When specified, the *ocspDbHandle* parameter is required.
ocspReqSignatureAlgorithm	Specifies the signature algorithm to be used to sign OCSP requests. The supported signature algorithms are: • x509_alg_md5WithRsaEncryption • x509_alg_sha1WithRsaEncryption • x509_alg_dsaWithSha1 • x509_alg_ecdsaWithSha1 • x509_alg_sha224WithRsaEncryption • x509_alg_dsaWithSha224 • x509_alg_ecdsaWithSha224 • x509_alg_sha256WithRsaEncryption • x509_alg_dsaWithSha256 • x509_alg_ecdsaWithSha256 • x509_alg_sha384WithRsaEncryption • x509_alg_ecdsaWithSha384 • x509_alg_sha512WithRsaEncryption • x509_alg_ecdsaWithSha512 • x509_alg_mgf1Sha256WithRsaSsaPss • x509_alg_mgf1Sha384WithRsaSsaPss • x509_alg_mgf1Sha512WithRsaSsaPss To have the default signature algorithm used (RSA with SHA-256), set to 0. **Note:** When executing in FIPS mode, signature algorithm x509_alg_md5WithRSAEncryption is not supported.

Table 19. gskdb_ocsp_source structure parameters (continued)

Parameter name	Description
ocspResponseSigAlgPairs	Specifies a preference ordered list of hash and signature algorithm pair specifications that are sent on the OCSP request and may be used by the OCSP responder to select an appropriate algorithm for signing the OCSP response. If set to NULL, a preference ordered list of hash and signature algorithm pairs are not sent to the OCSP responder. If set to a non-NULL value, the string consists of one or more 4-character values in order of preference for use. The OCSP response must be signed with one of these hash and signature algorithm pairs and if it is not, the OCSP response is rejected. It should be noted that not all OCSP responders support the preference ordered list and the OCSP response may be signed by a signature algorithm that was not specified. See Table 31 on page 805 for a list of valid 4-character signature algorithm pair specifications.
ocspResponseTimeout	Specifies the time in seconds to wait for a response from an OCSP responder. The valid time limits are 0 through 43200 seconds (12 hours) and defaults to 30 seconds. A value of 0 means there is no time limit.

ocspURL	Specifies the default URL that is used when contacting an OCSP responder.
	
	The URL must conform to the definition of an HTTP url:
	
	`http_URL = "http:" "//" host [ ":" port ]`
	`[ abs_path [ "?" query ]]`
	
	where host can be an IPv4 address, an IPv6 address, or a domain name.

ocspURLPriority	Specifies the order of precedence for contacting OCSP responder locations if *ocspURL* is non-NULL and *ocspEnable* is TRUE. **TRUE** The URL specified in *ocspURL* has priority over the certificate's AIA extension. **FALSE** The URL specified in the AIA extension has priority over the *ocspURL* value.
ocspUseGetMethod	Specifies the OCSP HTTP method to be used: **TRUE** HTTP GET method is used. **FALSE** HTTP POST method is used.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_create_self_signed_certificate()

Creates a self-signed certificate.

Format

```
#include <gskcms.h>

gsk_status gsk_create_self_signed_certificate (
                        gsk_handle               db_handle,
                        const char *             label,
                        x509_algorithm_type      signature_algorithm,
                        int                      key_size,
                        const char *             subject_name,
                        int                      num_days,
                        gsk_boolean              ca_certificate,
                        x509_extensions *        extensions)
```

Parameters

db_handle
: Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine. This must be a key database and not a request database.

label
: Specifies the label for the new database record. The label is specified in the local code page.

signature_algorithm
: Specifies the certificate signature algorithm.

key_size
: Specifies the key size in bits.

subject_name
: Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

num_days
: Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

ca_certificate
: Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

extensions
: Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
: The signature algorithm is not valid.

[CMSERR_BACKUP_EXISTS]
: The backup file already exists.

[CMSERR_BAD_HANDLE]
: The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
: The key size is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_BAD_SUBJECT_NAME]
The subject name is not valid.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_ICSF_CLEAR_KEY_SUPPORT_NOT_AVAILABLE]
Clear key support not available due to ICSF key policy.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_FIPS_KEY_PAIR_CONSISTENCY]
FIPS mode key generation failed pair-wise consistency check.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_KEY_MISMATCH]
The signer certificate key cannot be used to sign a certificate.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UNSUPPORTED_EXPIRATION]
The expiration date exceeds February 6, 2106, at 23:59:59 UTC.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_create_self_signed_certificate()** routine will generate a self-signed X.509 certificate as described in RFC 5280 (tools.ietf.org/html/rfc5280). A certification authority certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user self-signed certificate contains the basic constraints extension marked as a CA and sets the digital signature, non-repudiation, key encipherment, data encipherment, certificate signature, and CRL signature bits in the key usage extension. The new certificate is then stored in the key database. The **gsk_export_certificate()** routine can be called to create an export file containing the certificate for transmission to another system.

These signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest - {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest - {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest - {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest - {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest - {1.2.840.10045.4.3.4}

These RSASSA-PSS [1.2.840.113549.1.1.10] signatures are supported:

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation function algorithm 1.

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

If not in FIPS mode, an RSA key size must be between 512 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be between 512 and 2048 bits. Key sizes of between 512 and 1024 bits are rounded up to a multiple of 64. A key size of 2048 must be explicitly specified as such. A key size of 1024 or less should specify signature algorithm x509_alg_dsaWithSha1, while a key size of 2048 bits should specify either x509_alg_dsaWithSha224 or x509_alg_dsaWithSha256 as the signature algorithm.

In FIPS mode, an RSA key size must be between 1024 and 4096 bits and will be rounded up to a multiple of 16 bits. A DSA key size must be either 1024 bits or 2048 bits. A key size of 1024 bits should specify signature algorithm x509_alg_dsaWithSha1, while a key size of 2048 bits should specify either x509_alg_dsaWithSha224 or x509_alg_dsaWithSha256 as the signature algorithm.

For an ECC key, the key size will determine the default namedCurve that will be used for the public/private key pair, as specified in Table 3 on page 13. In FIPS mode, only NIST recommended curves are supported. To specify a specific supported elliptic curve, use **gsk_construct_self_signed_certificate()** to create a self-signed certificate.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

Both a CA certificate and an end user certificate will have SubjectKeyIdentifier, AuthorityKeyIdentifier, KeyUsage and BasicConstraints extensions. The application can supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage or BasicContraints extension provided by the application will replace the default extension created for the certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.

The database must be open for update in order to add the new certificate. The database file is updated as part of the **gsk_create_self_signed_certificate()** processing. A temporary database file is created using the same name as the database file with ".new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

Note: A self-signed end-entity certificate (server or client certificate) is not recommended for use in production environments and should only be used to facilitate test environments before production. Self-signed certificates do not imply any level of security or authenticity of the certificate because, as their name implies, they are signed by the same key that is contained in the certificate. However, certificates that are signed by a certificate authority indicate that, at least at the time of signature, the certificate authority approved the information contained in the certificate.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_create_signed_certificate()

Creates a signed certificate.

This function is deprecated. Use **gsk_create_signed_certificate_record()** instead.

Format

```
#include <gskcms.h>

gsk_status gsk_create_signed_certificate (
                            gsk_handle              db_handle,
                            const char *            label,
                            int                     num_days,
                            gsk_boolean             ca_certificate,
                            x509_extensions *       extensions,
                            gsk_buffer *            cert_request,
                            gsk_buffer *            signed_certificate)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine. This must be a key database and not a request database.

label
Specifies the label for the certificate to be used to sign the new certificate. The label is specified in the local code page.

num_days
Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

ca_certificate
Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

extensions
Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

cert_request
Specifies the PKCS #10 certification request stream in either binary DER-encoded format or in Base64 format. A Base64 stream is in the local code page.

signed_certificate
Returns the signed certificate in Base64 format. The Base64 stream will be in the local code page. The application should call the **gsk_free_buffer()** routine to release the certificate stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not valid.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_ENCODING]
The certificate request stream is not valid.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_BAD_SIGNATURE]
The request signature is not correct.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_EXPIRED]
The signer certificate is expired.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_TYPE]
Incorrect key algorithm

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing certificates.

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_ISSUER_NOT_CA]
The signer certificate is not for a certification authority.

[CMSERR_KEY_MISTMATCH]
The signer certificate key cannot be used to sign a certificate.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_RECORD_NOT_FOUND]
The signer certificate is not found in the key database.

[CMSERR_SUBJECT_IS_CA]
The requested subject name is the same as the signer name.

[CMSERR_UNSUPPORTED_EXPIRATION]
The expiration date exceeds February 6, 2106, at 23:59:59 UTC.

Usage

The **gsk_create_signed_certificate()** routine will generate an X.509 certificate as described in RFC 5280 (tools.ietf.org/html/rfc5280). The new certificate will be signed using the certificate specified by the *label* parameter.

If the certificate request contains an ECC key, the signing certificate cannot contain a DSA key.

A certification authority certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions which allow the certificate to be used as follows:

- An RSA key can be used for authentication, digital signature, and data encryption.
- A DSS key can be used for authentication and digital signature.
- A Diffie-Hellman key can be used for key agreement.
- An ECC key can be used for authentication, digital signature and key agreement.

The certificate expiration date will be set to the earlier of the requested expiration date and the expiration date of the signing certificate.

The signing certificate must have an associated private key, the BasicConstraints extension must either be omitted or must have the CA indicator set, and the KeyUsage extension must either be omitted or must allow signing certificates.

A CA certificate will have SubjectKeyIdentifier, KeyUsage, and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An AuthorityKeyIdentifier extension will be created if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage, or BasicConstraints extension provided by the application will replace the default extension created for the certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.

Certificate extensions can also be contained within the certification request. A certificate extension supplied by the application will override a certificate extension of the same type contained in the certification request. The certificate extensions found in the certification request will be copied unmodified to the new certificate with these exceptions:

- The AuthorityInfoAccess, AuthorityKeyIdentifier, BasicConstraints, CrlDistributionPoints, IssuerAltName, NameConstraints, PolicyConstraints, PolicyMappings, and PrivateKeyUsagePeriod extensions will not be copied
- The keyCertSign and crlSign flags in the KeyUsage extension will be modified based upon the value of the *ca_certificate* parameter.

No certification path validation is performed by the **gsk_create_signed_certificate()** routine. An error will be returned if the requested subject name is the same as the subject name in the signing certificate.

The signature algorithm used when signing the certificate is derived from the key algorithm of the provided CA certificate's private key.

- For a RSA key algorithm, the signature digest type is SHA-1 and the signature is based on RSA encryption.
- For a DSA key algorithm, when using a 1024-bit DSA key, the digest type is SHA-1. When using a 2048-bit DSA key, the digest type is SHA-256.
- For a ECC key algorithm, the signature digest type is the suggested digest for the key size, as specified in Table 2 on page 13.

gsk_create_signed_certificate_record()

Creates a signed certificate.

Format

```
#include <gskcms.h>

gsk_status gsk_create_signed_certificate_record (
                           gsk_handle            db_handle,
                           const char *          label,
                           int                   num_days,
                           gsk_boolean           ca_certificate,
                           x509_algorithm_type   signature_algorithm,
                           x509_extensions *     extensions,
                           gsk_buffer *          cert_request,
                           gsk_buffer *          signed_certificate)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine. This must be a key database and not a request database.

label
Specifies the label for the certificate to be used to sign the new certificate. The label is specified in the local code page.

num_days
Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

ca_certificate
Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

signature_algorithm
Specifies the signature algorithm to be used for the certificate signature.

extensions
Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

cert_request
Specifies the PKCS #10 certification request stream in either binary DER-encoded format or in Base64 format. A Base64 stream is in the local code page.

signed_certificate
Returns the signed certificate in Base64 format. The Base64 stream will be in the local code page. The application should call the **gsk_free_buffer()** routine to release the certificate stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not valid.

[CMSERR_BACKUP EXISTS]
The backup file already exists.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
Encryption key size is not supported.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_BAD_SUBJECT_NAME]
The subject name is not valid.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_RSA_PRIVATE_KEY_BAD_TYPE]
PKDS RSA private key type not valid for RSASSA-PSS signature generation.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_TYPE]
Incorrect key algorithm.

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing certificates

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_KEY_MISMATCH]
The signer certificate key cannot be used to sign a certificate or the signers key type is not supported for the requested signature algorithm.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_MASK_ALG_NOT_SUPPORTED]
RSASSA-PSS mask generation algorithm is not supported.

[CMSERR_UNSUPPORTED_EXPIRATION]
The expiration date exceeds February 6, 2106, at 23:59:59 UTC.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_create_signed_certificate_record()** routine will generate an X.509 certificate as described in RFC 5280 (tools.ietf.org/html/rfc5280). The new certificate will be signed using the certificate specified by the *label* parameter and the signature algorithm specified by the *signature_algorithm* parameter.

If the certificate request contains an ECC key, the signing certificate cannot contain a DSA key.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation algorithm 1.

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest - {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest - {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest - {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest - {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest - {1.2.840.10045.4.3.4}

When executing in non-FIPS mode, an RSA key size must be between 512 and 4096 bits. A DSA key size must be between 512 and 2048 bits. An ECC key can be a NIST recommended named curve with a key between 192 and 521 or a Brainpool curve with a key between 160 and 512.

When executing in FIPS mode:

- Signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.
- An RSA key size must be between 1024 and 4096 bits.
- A DSA key size must be either 1024 bits or 2048 bits.
- An NIST ECC key must be a NIST recommended named curve with a key size between 192 and 521.

For more information about FIPS supported algorithms and key sizes, see "Algorithms and key sizes" on page 19.

A DSA key size of 1024 or less should specify signature algorithm x509_alg_dsaWithSha1, while a key size of 2048 bits should specify either x509_alg_dsaWithSha224 or x509_alg_dsaWithSha256 as the signature algorithm.

A certification authority certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions which allow the certificate to be used as follows:

- An RSA key can be used for authentication, digital signature, and data encryption.
- A DSA key can be used for authentication and digital signature.
- A Diffie-Hellman key can be used for key agreement.
- An ECC key can be used for authentication, digital signature and key agreement.

The certificate expiration date will be set to the earlier of the requested expiration date and the expiration date of the signing certificate.

The signing certificate must have an associated private key, the BasicConstraints extension must either be omitted or must have the CA indicator set, and the KeyUsage extension must either be omitted or must allow signing certificates.

A CA certificate will have SubjectKeyIdentifier, KeyUsage, and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An AuthorityKeyIdentifier extension will be created if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage, or BasicConstraints extension provided by the application will replace the default extension created for the certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.

Certificate extensions can also be contained within the certification request. A certificate extension supplied by the application will override a certificate extension of the same type contained in the certification request. The certificate extensions found in the certification request will be copied unmodified to the new certificate with these exceptions:

- The AuthorityInfoAccess, AuthorityKeyIdentifier, BasicConstraints, CrlDistributionPoints, IssuerAltName, NameConstraints, PolicyConstraints, PolicyMappings, and PrivateKeyUsagePeriod extensions will not be copied
- The keyCertSign and crlSign flags in the KeyUsage extension will be modified based upon the value of the *ca_certificate* parameter.

No certification path validation is performed by the **gsk_create_signed_certificate_record()** routine. An error will be returned if the requested subject name is the same as the subject name in the signing certificate.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, "System SSL and FIPS 140-2," on page 19.

gsk_create_signed_certificate_set()

Creates a signed certificate as part of a set of certificates.

This function is deprecated. Use **gsk_create_database_signed_certificate()** instead.

Format

```
#include <gskcms.h>

gsk_status gsk_create_signed_certificate_set (
                                 gsk_handle                db_handle,
                                 const char *              ca_label,
                                 const char *              record_label,
                                 x509_algorithm_type       key_algorithm,
                                 int                       key_size,
                                 gsk_buffer *              key_parameters,
                                 const char *              subject_name,
                                 int                       num_days,
                                 gsk_boolean               ca_certificate,
                                 x509_extensions           extensions)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine. This must be a key database and not a request database.

ca_label
Specifies the label of the certificate to be used to sign the new certificate. The key usage for the certificate must allow certificate signing. The label is specified in the local code page.

record_label
Specifies the label for the new database record. The label is specified in the local code page.

key_algorithm
Specifies the certificate key algorithm.

key_size
Specifies the certificate key size in bits.

key_parameters
Specifies the key generation parameters. Specify NULL for this parameter if the key algorithm does not require any key parameters.

subject_name
Specifies the distinguished name for the certificate subject. The distinguished name is specified in the local code page and consists of one or more relative distinguished name components separated by commas.

num_days
Specifies the number of days for the certificate validity period as a value between 1 and 9999 (the maximum of 9999 will be used if a larger value is specified and the minimum of 1 will be used if a smaller value is specified).

ca_certificate
Specify TRUE if this is a certification authority certificate or FALSE if this is an end user certificate.

extensions
Specifies the certificate extensions for the new certificate. Specify NULL for this parameter if no certificate extensions are supplied.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
: The key algorithm or the signature algorithm is not valid.

[CMSERR_BACKUP_EXISTS]
: The backup file already exists.

[CMSERR_BAD_EC_PARAMS]
: Elliptic Curve parameters are not valid.

[CMSERR_BAD_HANDLE]
: The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
: The key size is not valid.

[CMSERR_BAD_LABEL]
: The record label or CA certificate label is not valid.

[CMSERR_BAD_SUBJECT_NAME]
: The subject name is not valid.

[CMSERR_CRYPTO_FAILED]
: Unexpected cryptographic request failure.

[CMSERR_DUPLICATE_EXTENSION]
: Supplied extensions contain a duplicate extension.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
: Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
: Elliptic Curve is not supported.

[CMSERR_EXPIRED]
: The signer certificate is expired.

[CMSERR_ICSF_CLEAR_KEY_SUPPORT_NOT_AVAILABLE]
: Clear key support not available due to ICSF key policy.

[CMSERR_ICSF_FIPS_DISABLED]
: ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
: ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
: ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
: ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
: The database type does not support certificates.

[CMSERR_INCORRECT_KEY_TYPE]
: Incorrect key algorithm

[CMSERR_INCORRECT_KEY_USAGE]
: The signer certificate key usage does not allow signing certificates.

[CMSERR_INTERNAL_ERROR]
: An internal processing error has occurred.

[CMSERR_IO_ERROR]
: Unable to read or write a database record.

[CMSERR_ISSUER_NOT_CA]
: The signer certificate is not for a certification authority.

[CMSERR_KEY_MISMATCH]
The signer certificate key cannot be used to sign a certificate.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_SUBJECT_IS_CA]
The requested subject name is the same as the signer name.

[CMSERR_UNSUPPORTED_EXPIRATION]
The expiration date exceeds February 6, 2106, at 23:59:59 UTC.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_create_signed_certificate_set()** routine will generate an X.509 certificate as described in RFC 5280 (tools.ietf.org/html/rfc5280). The certificate will be signed using an existing certificate as specified by the *ca_label* parameter.

- If the specified certificate key is a Diffie-Hellman key, the signing certificate must contain either an RSA or a DSA key.
- If the specified certificate key is an ECC key, the signing certificate cannot contain a DSA key.

A certification authority (CA) certificate will have basic constraints and key usage extensions which allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate will have basic constraints and key usage extensions as follows:

- An RSA key can be used for authentication, digital signature, and data encryption
- A DSS key can be used for authentication and digital signature
- A Diffie-Hellman key can be used for key agreement
- An ECC key can be used for authentication, digital signature, and key agreement.

The new certificate will be stored in the key database using the supplied record label. The **gsk_export_certificate()** routine can be called to create an export file containing the certificate for transmission to another system.

These key algorithms are supported:

x509_alg_rsaEncryption
RSA encryption - {1.2.840.113549.1.1.1}

x509_alg_idDsa
Digital Signature Standard (DSS) - {1.2.840.10040.4.1}

x509_alg_dhPublicNumber
Diffie-Hellman (DH) - {1.2.840.10046.2.1}

x509_alg_ecPublicKey
Elliptic Curve Public Key (ECC) - {1.2.840.10045.2.1}

RSA keys

- Can be used for both CA certificates and end user certificates
- Key size when not in FIPS mode is between 512 and 4096 bits rounded up to a multiple of 16

- Key size in FIPS mode is between 1024 and 4096 bits rounded up to a multiple of 16
- No key parameters

DSS keys

- Can be used for both CA certificates and end user certificates.
- Key sizes of between 512 and 1024 bits when in non-FIPS mode are rounded up to a multiple of 64.
- Key sizes less than 1024 bits can only be generated in non-FIPS mode and are generated according to FIPS 186-2.
- Key sizes of 1024 or 2048 bits are generated according to FIPS 186-4 in both FIPS mode and non-FIPS mode. These are the only valid key sizes in FIPS mode.
- A key size of 1024 bits or less will use SHA1 digest, while a key size of 2048 bits will use SHA256 digest.
- Key parameters encoded as an ASN.1 sequence consisting of the prime p, the prime divisor q, and the generator g. For 1024-bit and 2048-bit keys, see FIPS 186-4: Digital Signature Standard (DSS) (nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf) for more information about the key parameters. For smaller key sizes, see FIPS 186-2: Digital Signature Standard (DSS) (csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf). Note that key parameters that contain a p of 2048 bits and a q of 160 bits do not conform to FIPS 186-4 and are not supported. The **gsk_generate_key_parameters()** routine can be used to generate the key parameters.

DH keys

- Can be used only for end user certificates
- Can only be signed using a certificate containing either an RSA or a DSA key
- Key size when not in FIPS mode is between 512 and 2048 bits rounded up to a multiple of 64
- Key size in FIPS mode of 2048 bits
- Key parameters encoded as an ASN.1 sequence consisting of the prime P, the base G, the subprime Q and the subgroup factor J. See RFC 2631 (tools.ietf.org/html/rfc2631) for more information about the key parameters for non-FIPS mode, and see *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for FIPS mode. The **gsk_generate_key_parameters()** routine can be used to generate the key parameters.

ECC keys

- Can be used for both CA certificates and end user certificates.
- The ECC named curve used to generate the ECC key pair can be specified using either the *key_parameters* buffer or the *key_size* parameter. If the *key_parameters* buffer is supplied the *key_size* parameter will be ignored.
- The *key_parameters* buffer must contain ASN.1 encoded EC domain parameters, or be NULL.
- If the *key_parameters* buffer is not supplied, the *key_size* parameter will be rounded up to the nearest supported key size and the default EC named curve for that key size will be used, as specified in Table 3 on page 13.
- In FIPS mode only NIST recommended curves are supported.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

A CA certificate will have SubjectKeyIdentifier, KeyUsage and BasicConstraints extensions while an end user certificate will have SubjectKeyIdentifier and KeyUsage extensions. An AuthorityKeyIdentifier extension will be created if the signing certificate has a SubjectKeyIdentifier extension. The application can supply additional extensions through the extensions parameter. An AuthorityKeyIdentifier, KeyUsage or BasicConstraints extension provided by the application will replace the default extension created for the certificate, however a SubjectKeyIdentifier extension provided by the application will be ignored.

The database must be open for update in order to add the new certificate. The database file is updated as part of the **gsk_create_signed_certificate_set()** processing. A temporary database file is created using the same name as the database file with ".new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

The signature algorithm used when signing the certificate is derived from the key algorithm of the provided CA certificate's private key.

- For a RSA key algorithm, the signature digest type is SHA-1 and the signature is based on RSA encryption.
- For a DSA key algorithm, when using a 1024-bit DSA key, the digest type is SHA-1. When using a 2048-bit DSA key, the digest type is SHA-256.
- For a ECC key algorithm, the signature digest type is the suggested digest for the key size, as specified in Table 2 on page 13.

gsk_create_signed_crl()

Creates a signed certificate revocation list.

This function is deprecated. Use **gsk_create_signed_crl_record()** instead.

Format

```
#include <gskcms.h>

gsk_status gsk_create_signed_crl (
                        gsk_handle                       db_handle,
                        const char *                     label,
                        gsk_int32                        crl_number,
                        int                              num_days,
                        x509_revoked_certificates *      revoked_certificates,
                        x509_extensions *                extensions,
                        gsk_buffer *                     signed_crl)
```

Parameters

db_handle
Specifies the database handle that is returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine. This database handle must be a key database and not a request database.

label
Specifies the label for the certificate to be used to sign the certificate revocation list. The label is specified in the local code page.

crl_number
Specifies the CRL number. Each CRL is numbered and each successive revocation list has a larger CRL number than all previous revocation lists.

num_days
Specifies the number of days until the next CRL is issued and is specified as a value between 1 and 9999 (the maximum of 9999 is used if a larger value is specified and the minimum of 1 is used if a smaller value is specified).

revoked_certificates
Specifies the revoked list of certificates to be included in the CRL. This list consists of the certificate serial numbers and not the actual certificates.

extensions
Specifies the CRL extensions for the new CRL. Specify NULL for this parameter if no CRL extensions are supplied.

signed_crl
Returns the signed certificate revocation list in Base64 format. The Base64 stream is in the local code page. The application should call the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve is not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_EXPIRED]
The signer certificate is expired.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing a CRL.

[CMSERR_ISSUER_NOT_CA]
The signer certificate is not for a certification authority.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_RECORD_NOT_FOUND]
The signer certificate is not found in the key database.

[CMSERR_UNSUPPORTED_EXPIRATION]
The expiration date exceeds February 6, 2106, at 23:59:59 UTC.

Usage

The **gsk_create_signed_crl()** routine generates an X.509 certificate revocation list (CRL) as described in RFC 5280 (tools.ietf.org/html/rfc5280). The new CRL is signed by using the certificate that is specified by the *label* parameter. The number of days until the next CRL is issued is set to the earlier of the requested date and the expiration of the signing certificate.

The signing certificate must have an associated private key, the BasicConstraints extension must either be omitted or must have the CA indicator set, and the KeyUsage extension must either be omitted or must allow signing certificate revocation lists.

The CRL has a CRLNumber extension that contains the value that is specified by the *crl_number* parameter. It also has an AuthorityKeyIdentifier extension if the signing certificate has a SubjectKeyIdentifier extension. The application can supply extra extensions through the *extensions* parameter. An AuthorityKeyIdentifier or CRLNumber extension that is provided by the application replaces the default extension that is created for the CRL.

No certification path validation is performed by the **gsk_create_signed_crl()** routine.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

The signature algorithm used when signing the CRL is derived from the key algorithm of the provided CA certificate's private key.

- For a RSA key algorithm, the signature digest type is SHA-1 and the signature is based on RSA encryption.
- For a DSA key algorithm, when using a 1024-bit DSA key, the digest type is SHA-1. When using a 2048-bit DSA key, the digest type is SHA-256.
- For a ECC key algorithm, the signature digest type is the suggested digest for the key size, as specified in Table 2 on page 13.

gsk_create_signed_crl_record()

Creates a signed certificate revocation list.

Format

```
#include <gskcms.h>

gsk_status gsk_create_signed_crl_record (
                            gsk_handle                   db_handle,
                            const char *                 label,
                            x509_algorithm_type          signature_algorithm,
                            gsk_int32                    crl_number,
                            int                          num_days,
                            x509_revoked_certificates *  revoked_certificates,
                            x509_extensions *            extensions,
                            gsk_buffer *                 signed_crl)
```

Parameters

db_handle
Specifies the database handle that is returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine. This database handle must be a key database and not a request database.

label
Specifies the label for the certificate to be used to sign the certificate revocation list. The label is specified in the local code page.

signature_algorithm
Specifies the signature algorithm to be used for the crl signature.

crl_number
Specifies the CRL number. Each CRL is numbered and each successive revocation list has a larger CRL number than all previous revocation lists.

num_days
Specifies the number of days until the next CRL is issued and is specified as a value between 1 and 9999 (the maximum of 9999 is used if a larger value is specified and the minimum of 1 is used if a smaller value is specified).

revoked_certificates
Specifies the revoked list of certificates to be included in the CRL. This list consists of the certificate serial numbers and not the actual certificates.

extensions
Specifies the CRL extensions for the new CRL. Specify NULL for this parameter if no CRL extensions are supplied.

signed_crl
Returns the signed certificate revocation list in Base64 format. The Base64 stream is in the local code page. The application should call the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
Encryption key size is not supported.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve is not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_EXPIRED]
The signer certificate is expired.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_RSA_PRIVATE_KEY_BAD_TYPE]
PKDS RSA private key type not valid for RSASSA-PSS signature generation.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The signer certificate key usage does not allow signing a CRL.

[CMSERR_ISSUER_NOT_CA]
The signer certificate is not for a certification authority.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_RECORD_NOT_FOUND]
The signer certificate is not found in the key database.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_MASK_ALG_NOT_SUPPORTED]
RSASSA-PSS mask generation algorithm is not supported.

[CMSERR_UNSUPPORTED_EXPIRATION]
The expiration date exceeds February 6, 2106, at 23:59:59 UTC.

Usage

The **gsk_create_signed_crl_record()** routine generates an X.509 certificate revocation list (CRL) as described in RFC 5280 (tools.ietf.org/html/rfc5280). The new CRL is signed by using the certificate that is specified by the *label* parameter and the signature algorithm that is specified by the *signature_algorithm* parameter.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation algorithm 1.

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest - {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest - {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest – {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest – {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest – {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest – {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest – {1.2.840.10045.4.3.4}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

The number of days until the next CRL is issued is set to the earlier of the requested date and the expiration of the signing certificate.

The signing certificate must have an associated private key, the BasicConstraints extension must either be omitted or must have the CA indicator set, and the KeyUsage extension must either be omitted or must allow signing certificate revocation lists.

The CRL has a CRLNumber extension that contains the value that is specified by the *crl_number* parameter. It also has an AuthorityKeyIdentifier extension if the signing certificate has a SubjectKeyIdentifier extension. The application can supply extra extensions through the *extensions* parameter. An AuthorityKeyIdentifier or CRLNumber extension that is provided by the application replaces the default extension that is created for the CRL.

No certification path validation is performed by the **gsk_create_signed_crl_record()** routine.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_decode_base64()

Decodes a Base64-encoded stream.

Format

```
#include <gskcms.h>

gsk_status gsk_decode_base64 (
                              gsk_buffer *     encoded_stream,
                              gsk_buffer *     decoded_stream)
```

Parameters

encoded_stream
Specifies the Base64-encoded stream. The encoded data must be in the local code page.

decoded_stream
Returns the decoded stream. The application should call the **gsk_free_buffer()** routine to release the decoded stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_BASE64_ENCODING]
Incorrect Base64 encoding.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_decode_base64()** routine will decode a Base64-encoded stream created by the **gsk_encode_base64()** routine. The encoded stream must be in the local code page and must not include any header or trailer lines added by the application to identify the stream contents (such as '-----BEGIN CERTIFICATE-----' or '-----END CERTIFICATE-----'). New line characters and whitespace characters (tabs and spaces) are ignored.

gsk_decode_certificate()

Decodes an X.509 certificate.

Format

```
#include <gskcms.h>

gsk_status gsk_decode_certificate (
                                   gsk_buffer *              stream,
                                   x509_certificate *        certificate)
```

Parameters

stream
Specifies the encoded certificate.

certificate
Returns the decoded certificate information. The application should call the **gsk_free_certificate()** routine to release the decoded certificate when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[ASN_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_decode_certificate()** routine decodes an X.509 certificate and returns the decoded information to the application. The certificate must have been encoded as described in RFC 5280 (tools.ietf.org/html/rfc5280). The *derCertificate* field will contain the undecoded TBSCertificate ASN.1 sequence for use in verifying the certificate signature, the *tbsCertificate* field will contain the decoded TBSCertificate ASN.1 sequence, and the *signatureAlgorithm* and *signatureValue* fields will contain the certificate signature. The **gsk_encode_signature()** routine can be used to re-create the encoded certificate from the x509_certificate structure returned by the **gsk_decode_certificate()** routine.

Character strings contained in the certificate will be returned using UTF-8 encoding. The application can call **iconv()** to convert the string to a different encoding as needed.

The certificate extensions will be returned with the extension values in ASN.1 encoded format. The **gsk_decode_certificate_extension()** routine can be called to decode a particular certificate extension. This allows all of the certificate extensions to be returned even when one or more extensions cannot be processed by the System SSL runtime.

gsk_decode_certificate_extension()

Decodes an X.509 certificate extension.

Format

```
#include <gskcms.h>

gsk_status gsk_decode_certificate_extension (
                                    x509_extension *           encoded_extension,
                                    x509_decoded_extension *   decoded_extension)
```

Parameters

encoded_extension
Specifies the encoded X.509 extension as returned by the **gsk_decode_certificate()** or **gsk_decode_crl()** routine.

decoded_extension
Returns the decoded extension data. The application should call the **gsk_free_decoded_extension()** routine to release the decoded extension when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_NO_MEMORY]
Insufficient memory is available.

[CMSERR_EXT_NOT_SUPPORTED]
The certificate extension is not supported.

[CMSERR_NO_MEMORY]
Insufficient memory is available.

Usage

The **gsk_decode_certificate()** and **gsk_decode_crl()** routines returns all of the certificate extensions in the x509_extensions structure with the extension values still in ASN.1 encoded format. The application then calls the **gsk_decode_certificate_extension()** routine to decode a specific certificate extension.

The **gsk_decode_certificate_extension()** routine returns character strings using UTF-8 encoding. If necessary, the application can call the **iconv()** routine to convert the strings to a different encoding.

These certificate extensions are supported:

- AuthorityInfoAccess
- AuthorityKeyIdentifier
- BasicConstraints
- CertificateIssuer
- CertificatePolicies
- CrlDistributionPoints
- CrlNumber
- CrlReasonCode
- DeltaCrlIndicator
- ExtKeyUsage
- FreshestCRL

- HoldInstructionCode
- HostIDMapping (z/OS specific extension 1.3.18.0.2.18.1)
- InhibitAnyPolicy
- InvalidityDate
- IssuerAltName
- IssuingDistributionPoint
- KeyUsage
- NameConstraints
- OCSPNoCheck
- PolicyConstraints
- PolicyMappings
- PrivateKeyUsagePeriod (not supported in RFC 5280)
- SubjectAltName
- SubjectDirectoryAttributes
- SubjectInfoAccess
- SubjectKeyIdentifier

These general name types are supported:

- DirectoryName
- DnsName
- IpAddress
- RegisteredId
- Rfc822Name
- UniformResourceIdentifier

These general name types are not supported and will be copied to the decoded extension data as an ASN.1-encoded sequence:

- otherName
- x400Address
- ediPartyName

See RFC 5280 (tools.ietf.org/html/rfc5280) for more information about the various certificate extensions.

gsk_decode_certification_request()

Decodes a PKCS #10 certification request.

Format

```
#include <gskcms.h>

gsk_status gsk_decode_certification_request (
                                     gsk_buffer *              stream,
                                     pkcs_cert_request *       request)
```

Parameters

stream
Specifies the encoded certification request.

request
Returns the decoded certification request. The application should call the **gsk_free_certification_request()** routine to release the decoded certification request when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[ASN_NO_MEMORY]
Insufficient memory is available.

Usage

The **gsk_decode_certification_request()** routine decodes a Public Key Cryptography Standards (PKCS) certification request and returns the decoded information to the application. The request must have been encoded as described in RFC 2986 (tools.ietf.org/html/rfc2986). The *derRequestInfo* field will contain the undecoded CertificationRequestInfo ASN.1 sequence for use in verifying the request signature, the *certificationRequestInfo* field will contain the decoded CertificationRequestInfo ASN.1 sequence, and the *signatureAlgorithm* and *signatureValue* fields will contain the request signature. The **gsk_encode_signature()** routine can be used to re-create the encoded certification request from the pkcs_cert_request structure returned by the **gsk_decode_certification_request()** routine.

Character strings contained in the request will be returned using UTF-8 encoding. If necessary, the application can call **iconv()** to convert the string to a different encoding.

gsk_decode_crl()

Decodes an X.509 certificate revocation list.

Format

```
#include <gskcms.h>

gsk_status gsk_decode_crl (
                            gsk_buffer *          stream,
                            x509_crl *            crl)
```

Parameters

stream
Specifies the encoded certificate revocation list.

crl
Returns the decoded information. The application should call the **gsk_free_crl()** routine to release the decoded certificate revocation list when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[ASN_NO_MEMORY]
Insufficient memory is available.

Usage

The **gsk_decode_crl()** routine decodes an X.509 certificate revocation list (CRL) and returns the decoded information to the application. The CRL must have been encoded as described in RFC 5280 (tools.ietf.org/html/rfc5280). The *derCertList* field will contain the undecoded TBSCertList ASN.1 sequence for use in verifying the certificate signature, the *tbsCertList* field will contain the decoded TBSCertList ASN.1 sequence, and the *signatureAlgorithm* and *signatureValue* fields will contain the certificate signature. The time values that return in the *next_update* field of the *x509_tbs_crl* structure within the *x509_crl* structure are *tm_year*, *tm_mon*, *tm_mday*, *tm_hour*, *tm_min*, and *tm_sec*. The **gsk_encode_signature()** routine can be used to re-create the encoded CRL from the x509_crl structure returned by the **gsk_decode_crl()** routine.

Character strings will be returned using UTF-8 encoding. If necessary, the application can call **iconv()** to convert the string to a different encoding.

The certificate extensions will be returned with the extension values in ASN.1 encoded format. The **gsk_decode_certificate_extension()** routine can be called to decode a particular certificate extension. This allows all of the certificate extensions to be returned even when one or more extensions cannot be processed by the System SSL runtime.

gsk_decode_import_certificate()

Decodes certificate from DERencoded or PKCS #7encoded data stream.

Format

```
#include <gskcms.h>

gsk_status gsk_decode_import_certificate (
                                gsk_buffer *         stream,
                                pkcs_certificate *   subject_certificate,
                                pkcs_certificates *  issuer_certificates)
```

Parameters

stream
Specifies the byte stream of the encoded certificate.

subject_certificate
Returns the decoded certificate.

issuer_certificates
Returns the decoded certificate chain for the subject certificate.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_BASE64_ENCODING]
The Base64 encoding of the import stream is not correct.

[CMSERR_BAD_ENCODING]
The certificate request stream is not valid.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_IMPORT_CERTIFICATE]
No certificate in import file.

Usage

The **gsk_decode_import_certificate()** function decodes a data stream into a pkcs_certificate structure. The pkcs_certificate structure *subject_certificate* returns the subject certificate, and the pkcs_certificates structure *issuer_certificates* returns the certificate chain for the subject certificate (all other certificates not part of the subject certificates chain are discarded). The root certificate for the chain is the final entry in the array.

The supplied stream can represent either the ASN.1 DER encoding for the certificate or the Cryptographic Message Syntax (PKCS #7) encoding for the certificate. This can be either the binary value or the Base64 encoding of the binary value. A Base64 Encoded stream must be in the local code page and must include the encoding header and footer lines.

The **gsk_decode_import_certificate()** function decodes a single certificate. If the PKCS #7 message contains multiple certificates, only the first certificate and its certificate chain will be decoded.

gsk_decode_import_key()

Decodes certificate and key from PKCS #12-encoded data stream.

Format

```
#include <gskcms.h>

gsk_status gsk_decode_import_key (
                                  gsk_buffer *          stream,
                                  const char *          password,
                                  pkcs_cert_key *       subject_certificate,
                                  pkcs_certificates *   issuer_certificates)
```

Parameters

stream
Specifies the byte stream of the encoded certificate.

password
Specifies the password for the import file. The password is single-byte EBCDIC in the local code page and must consist of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

subject_certificate
Returns the decoded certificate and key.

issuer_certificates
Returns the decoded certificate chain for the subject certificate.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The decryption algorithm is not valid.

[CMSERR_BAD_ENCODING]
The certificate request stream is not valid.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_IMPORT_CERTIFICATE]
No certificate in input stream.

[CMSERR_PW_INCORRECT]
The password is not correct.

Usage

The **gsk_decode_import_key()** function decodes a data stream into a pkcs_cert_key structure. The pkcs_cert_key structure *subject_certificate* returns the subject certificate and key, while the pkcs_certificates structure *issuer_certificates* returns the certificate chain for the subject certificate (all other certificates not part of the subject certificates chain are discarded). The root certificate for the chain is the final entry in the array.

The certificate and key must have been encoded according to the Personal Information Exchange Syntax (PKCS #12). The supplied stream can be the binary ASN.1 sequence or the Base64 encoding of the ASN.1 sequence. A Base64 encoded stream is assumed to be in the local code page and must include the encoding header and footer lines.

In FIPS mode, the only supported decryption algorithm for the import file is:

- **x509_alg_pbeWithSha1And3DesCbc** - Triple DES with SHA-1 digest.

gsk_decode_issuer_and_serial_number()

Decodes a PKCS #7 *IssuerAndSerialNumber*.

Format

```
#include <gskcms.h>

gsk_status gsk_decode_issuer_and_serial_number (
                                    gsk_buffer *            stream,
                                    pkcs_issuer_serial *    issuer_serial)
```

Parameters

stream
Specifies an ASN.1 DER-encoded stream containing a PKCS #7 *IssuerAndSerialNumber*.

issuer_serial
Returns the decoded issuer name and serial number. The application should call the **gsk_free_issuer_and_serial_number()** routine to release the storage when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_NO_MORE_DATA]
Input data is incomplete.

[ASN_SELECTION_OUT_OF_RANGE]
Selection is not within the valid range.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_decode_issuer_and_serial_number()** routine decodes a PKCS #7 (Cryptographic Message Syntax) ASN.1 DER-encoded *IssuerAndSerialNumber* and returns the issuer name and serial number.

gsk_decode_name()

Decodes an X.509 name.

Format

```
#include <gskcms.h>

gsk_status gsk_decode_name (
                            gsk_buffer *          stream,
                            x509_name *           name)
```

Parameters

stream
Specifies the ASN.1 stream for the name.

name
Returns the decoded X.509 name. The application should release the name when it is no longer needed by calling the **gsk_free_name()** routine.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[ASN_NO_MEMORY]
Insufficient memory is available.

Usage

The **gsk_decode_name()** routine will decode an ASN.1 DER-encoded X.509 name. The name must have been encoded as described in RFC 5280 (tools.ietf.org/html/rfc5280). Character strings will be stored in UTF-8 format and the *stringType* field in the x509_rdn_attribute structure will be set to indicate the ASN.1 encoded string type.

gsk_decode_private key()

Decodes a private key.

Format

```
#include <gskcms.h>

gsk_status gsk_decode_private_key (
                                   gsk_buffer *                  stream,
                                   pkcs_private_key_info *       private_key)
```

Parameters

stream
Specifies the ASN.1 stream for the encoded private key.

private_key
Returns the decoded private key. The application should release the private key when it is no longer needed by calling the **gsk_free_private_key_info()** routine.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[ASN_NO_MEMORY]
Insufficient memory is available.

Usage

The **gsk_decode_private_key()** routine will decode an ASN.1 DER-encoded private key. The private key must have been encoded as described in RFC 5208 (tools.ietf.org/html/rfc5208).

gsk_decode_public key()

Decodes a public key.

Format

```
#include <gskcms.h>

gsk_status gsk_decode_public_key (
                                   gsk_buffer *                  stream,
                                   x509_public_key_info *        public_key)
```

Parameters

stream
Specifies the ASN.1 stream for the encoded public key.

public_key
Returns the decoded public key. The application should release the public key when it is no longer needed by calling the **gsk_free_public_key_info()** routine.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[ASN_NO_MEMORY]
Insufficient memory is available.

Usage

The **gsk_decode_public_key()** routine will decode an ASN.1 DER-encoded public key. The public key must have been encoded as described in RFC 5280 (tools.ietf.org/html/rfc5280).

gsk_decode_signer_identifier()

Decodes a PKCS #7 signer identifier.

Format

```
#include <gskcms.h>

gsk_status gsk_decode_signer_identifier (
                                    int                 signer_info_version,
                                    gsk_buffer *        encoded_signer_identifier,
                                    pkcs_signer_id *    decoded_signer_identifier)
```

Parameters

signer_info_version
Specifies the PKCS #7 Cryptographic Message Syntax version that was used to create the signer identifier as described in RFC 3852:

- Specify 1 to use the certificate's *IssuerAndSerialNumber*.
- Specify 3 to use the certificate's *SubjectKeyIdentifier*.

encoded_signer_identifier
Specifies ASN.1 DER-encoded stream containing a PKCS #7 signer identifier.

decoded_signer_identifier
Returns the decoded *pkcs_signer_id* containing either an *IssuerAndSerialNumber* or *SubjectKeyIdentifier*.

The application should call the **gsk_free_signer_identifier()** routine to release the storage when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_NO_MORE_DATA]
Input data is incomplete.

[CMSERR_PKCS7_CMSVERSION_NOT_SUPPORTED]
PKCS #7 CMS version is not supported.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_decode_signer_identifier()** routine reads a PKCS #7 (Cryptographic Message Syntax) ASN.1 DER-encoded signer identifier and returns a *pkcs_signer_id* containing either the *IssuerAndSerialNumber* or the *SubjectKeyIdentifier*.

gsk_delete_record()

Deletes a record from a key or request database.

Format

```
#include <gskcms.h>

gsk_status gsk_delete_record (
                              gsk_handle         db_handle,
                              gsk_int32          record_id)
```

Parameters

db_handle
Specifies the database handle return by the **gsk_create_database()** routine or the **gsk_open_database()** routine.

record_id
Specifies the database record to be deleted.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_NOT_FOUND]
Record is not found.

[CMSERR_SIGNED_CERTS]
The database contains records signed using the certificate.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_delete_record()** routine deletes a record from a key or request database. The database must be open for update in order to delete records. The unique record identifier identifies the record to be deleted. A certificate record cannot be deleted from a key database if the database contains records that were signed using the certificate.

The database file is updated as part of the **gsk_delete_record()** processing. A temporary database file is created using the same name as the database file with ".new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

gsk_dn_to_name()

Converts a DN string to an X.509 name.

Format

```
#include <gskcms.h>

gsk_status gsk_dn_to_name (
                            const char *         dn,
                            x509_name *          name)
```

Parameters

dn
Specifies the distinguished name in the local code page.

name
Returns the X.509 name. The X.509 strings use UTF-8 encoding. The application should call the **gsk_free_name()** routine to release the name when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_ATTR_NOT_FOUND]
An attribute type is not recognized.

[ASN_CANT_CONVERT]
An encoded attribute value contains characters from the wrong character set.

[ASN_INVALID_VALUE]
An attribute value is not valid.

[ASN_NO_MEMORY]
Insufficient storage is available.

[ASN_WRONG_TYPE]
An encoded attribute value does not represent a character string.

[ASN_X500_NO_AVA_SEP]
An attribute value separator is missing.

[ASN_X500_OID_SYNTAX_ERROR]
An object identifier is not valid.

[ASN_X500_SYNTAX_ERROR]
The DN string format is not valid.

Usage

The **gsk_dn_to_name()** routine converts a distinguished name (DN) string to an X.509 name in accordance with RFC 2253 (tools.ietf.org/html/rfc2253). The input string consists of single-byte characters in the local code page. A double-byte character is represented using the escaped UTF-8 encoding of the double-byte character in the Unicode character set.

Attribute types may be specified using either attribute names or numeric object identifiers. Attribute values must represent string values.

These DN attribute names are recognized by the System SSL run time. An error is returned if the DN contains an unrecognized attribute name.

Table 20. DN attribute names

	Name
C	Country
CN	Common name
DC	Domain component
DNQUALIFIER	Distinguished name qualifier
E	E-mail address
EMAIL	E-mail address (preferred)
EMAILADDRESS	E-mail address
GENERATIONQUALIFIER	Generation qualifier
GIVENNAME	Given name
INITIALS	Initials
L	Locality
MAIL	RFC 822 style address
NAME	Name
O	Organization name
OU	Organizational unit name
PC	Postal code
S	State or province
SERIALNUMBER	Serial number
SN	Surname
SP	State or province
ST	State or province (preferred)
STREET	Street
T	Title

This is an example of a DN using attribute names and string values:

```
CN=Ronald Hoffman,OU=Endicott,O=IBM,C=US
```

This is the same DN using object identifiers and encoded string values. The encoded string values represent the ASN.1 DER encoding of the string. The System SSL run time supports these ASN.1 string types: PRINTABLE, VISIBLE, TELETEX, IA5, UTF8, BMP, and UCS.

```
2.5.4.3=#130E526F6E616C6420486F66666D616E,2.5.4.11=#1308456E6469636F7474,
2.5.4.10=#130349424D,2.5.4.6=13025553
```

Individual characters can be represented using escape sequences. This is useful when the character cannot be represented in a single-byte character set. The hexadecimal value for the escape sequence is the UTF-8 encoding of the character in the Unicode character set.

```
Unicode Letter Description         10646 code    UTF-8     Quoted
===============================    ==========    ======    =======
LATIN CAPITAL LETTER L             U0000004C     0x4C      L
LATIN SMALL LETTER U               U00000075     0x75      u
LATIN SMALL LETTER C WITH CARON    U0000010D     0xC48D    \C4\8D
LATIN SMALL LETTER I               U00000069     0x69      i
LATIN SMALL LETTER C WITH ACUTE    U00000107     0xC487    \C4\87

SN=Lu\C4\8Di\C4\87
```

An escape sequence can also be used for special characters which are part of the name and are not to be interpreted as delimiters. For example:

```
CN=L. Eagle,OU=Jones\, Dale and Mian,O=IBM,C=US
```

gsk_encode_base64()

Encodes binary data using Base64 encoding.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_base64 (
                               gsk_buffer *          input_data,
                               gsk_buffer *          encoded_data)
```

Parameters

input_data
Specifies the data to be encoded.

encoded_data
Returns the encoded stream in the local code page. The application should call the **gsk_free_buffer()** routine to release the encoded stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_encode_base64()** routine will encode binary data using Base64 encoding. The encoded stream will consist of printable characters in the local code page. A new line will be inserted after each group of 64 encoded characters with a final new line at the end of the encoded stream. The **gsk_decode_base64()** routine can be used to decode the data.

gsk_encode_certificate_extension()

Encodes an X.509 certificate extension.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_certificate_extension (
                                 x509_decoded_extension *    decoded_extension,
                                 gsk_boolean                 critical,
                                 x509_extension *            encoded_extension)
```

Parameters

decoded_extension
Specifies the decoded extension data.

critical
Specify TRUE if this is a critical extension or FALSE if it is not a critical extension.

encoded_extension
Returns the encoded X.509 extension. The application should call the **gsk_free_certificate_extension()** routine to release the extension when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_NO_MEMORY]
Insufficient memory is available.

[CMSERR_EXT_NOT_SUPPORTED]
The certificate extension is not supported.

[CMSERR_NO_MEMORY]
Insufficient memory is available.

Usage

The **gsk_encode_certificate_extension()** routine encodes a certificate extension and returns the encoded extension in a format that can be used as input to the **gsk_encode_certificate()** routine.

The **gsk_encode_certificate_extension()** routine assumes character strings use UTF-8 encoding. The application is responsible for providing character data in this format.

These certificate extensions are supported:

- AuthorityInfoAccess
- AuthorityKeyIdentifier
- BasicConstraints
- CertificateIssuer
- CertificatePolicies
- CrlDistributionPoints
- CrlNumber
- CrlReasonCode
- DeltaCrlIndicator
- ExtKeyUsage

- FreshestCRL
- HoldInstructionCode
- HostIDMapping (z/OS specific extension 1.3.18.0.2.18.1)
- InhibitAnyPolicy
- InvalidityDate
- IssuerAltName
- IssuingDistributionPoint
- KeyUsage
- NameConstraints
- OCSPNoCheck
- PolicyConstraints
- PolicyMappings
- PrivateKeyUsagePeriod (not supported in RFC 5280)
- SubjectAltName
- SubjectDirectoryAttributes
- SubjectInfoAccess
- SubjectKeyIdentifier

These general name types are supported:

- DirectoryName
- DnsName
- IpAddress
- RegisteredId
- Rfc822Name
- UniformResourceIdentifier

See RFC 5280 (tools.ietf.org/html/rfc5280) for more information about the various certificate extensions.

gsk_encode_ec_parameters()

Encodes the EC domain parameters for an ECC key

Format

```
#include <gskcms.h>

gsk_status gsk_encode_ec_parameters (
                                        int               arg_count,
                                        x509_ecurve_type  ec_curve,
                                        gsk_buffer *      key_params,
                                        ...)
```

Parameters

arg_count
Specifies the number of parameters following the *arg_count* parameter. Currently, *arg_count* must be set to 2.

ec_curve
Specifies the EC named curve

key_params
Returns the ASN.1 stream for the EC domain parameters. The application should call the **gsk_free_buffe**r function to release the storage when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_NO_MEMORY]
Insufficient memory is available.

[CMSERR_BAD_ARG_COUNT]
Variable argument count is not valid.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported

Usage

The **gsk_encode_ec_parameters()** routine will encode the EC domain parameters of an elliptic curve as an ASN.1 stream. The EC domain parameters will be encoded as described in SEC1 (Elliptic Curve Cryptography).

gsk_encode_export_certificate()

Encodes an X.509 certificate into a DER or PKCS #7 data stream.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_export_certificate (
                                       pkcs_certificate *   subject_certificate,
                                       pkcs_certificates *  issuer_certificates,
                                       gskdb_export_format  format,
                                       gsk_buffer *         stream)
```

Parameters

subject_certificate
Specifies the certificate.

issuer_certificates
Specifies the certificate chain for the subject certificate.

format
Specifies the export format. These values may be specified:

gskdb_export_der_binary
Binary ASN.1 DER-encoded

gskdb_export_der_base64
Base64 ASN.1 DER-encoded

gskdb_export_pkcs7_binary
Binary PKCS #7 Cryptographic Message Syntax

gskdb_export_pkcs7_base64
Base64 PKCS #7 Cryptographic Message Syntax

stream
Returns the byte stream for the encoded certificate. The application should call the **gsk_free_buffer** function to release the storage when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_RNG_OUTPUT]
In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_FMT_NOT_SUPPORTED]
An unsupported export file stream format is specified.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_encode_export_certificate()** function encodes an X.509 certificate using either the ASN.1 DER encoding for the certificate or the Cryptographic Message Syntax (PKCS #7) encoding for the certificate. This can be either the binary value or the Base64 encoding of the binary value. A Base64 encoded stream will be in the local code page and will include the encoding header and footer lines.

The export data stream contains just the requested certificate when the DER format is selected. The export data stream contains the requested certificate and its certification chain when the PKCS #7 format is selected. The certificate chain for the subject certificate is supplied from the pkcs_certificates structure *issuer_certificates* with the root certificate being the final entry in the array. A partial certification chain will be exported if the complete chain is not supplied in *issuer_certificates*.

gsk_encode_export_key()

Encodes an X.509 certificate and its private key into a PKCS #12 data stream.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_export_key (
                                       pkcs_cert_key *       subject_certificate,
                                       pkcs_certificates *   issuer_certificates,
                                       gskdb_export_format   format,
                                       x509_algorithm_type   algorithm,
                                       const char *          password,
                                       const char *          nickname,
                                       gsk_buffer *          stream)
```

Parameters

subject_certificate
Specifies the certificate and key. The private key must not be stored in ICSF's PKDS.

issuer_certificates
Specifies the certificate chain for the subject certificate.

format
Specifies the export format. These values may be specified:

gskdb_export_pkcs12v1_binary
Binary PKCS #12 Version 1.

gskdb_export_pkcs12v1_base64
Base64 PKCS #12 Version 1.

gskdb_export_pkcs12v3_binary
Binary PKCS #12 Version 3.

gskdb_export_pkcs12v3_base64
Base64 PKCS #12 Version 3.

algorithm
Specifies the encryption algorithm for the export file. The strong encryption algorithms may not be available depending upon government export regulations. These values may be specified:

x509_alg_pbeWithSha1And40BitRc2Cbc
40bit RC2 with SHA-1 digest.

x509_alg_pbeWithSha1And128BitRc2Cbc
128-bit RC2 with SHA-1 digest.

x509_alg_pbeWithSha1And40BitRc4
40bit RC4 with SHA-1 digest.

x509_alg_pbeWithSha1And128BitRc4
128-bit RC4 with SHA-1 digest.

x509_alg_pbeWithSha1And3DesCbc
Triple DES with SHA-1 digest.

In FIPS mode, the only supported encryption algorithm for the export file is:

x509_alg_pbeWithSha1And3DesCbc
Triple DES with SHA-1 digest.

password
Specifies the password for the export file. The password is in the local code page and must consist of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string. The user is prompted to enter the password if NULL is specified for

this parameter. If the key that is being encoded for export is a secure private key in the TKDS, the maximum password length is 63 bytes.

nickname
Specifies the nickname assigned to the exported key in the bagAttributes field for a PKCS #12 Version 1 format file. The nickname is in the local code page. It may not be an empty string. If a PKCS #12 Version 3 export file format is specified, this parameter is ignored.

stream
Returns the byte stream for the encoded certificate. The application should call the **gsk_free_buffer()** function to release the storage when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not valid.

[CMSERR_CRYPTO_HARDWARE_NOT_AVAILABLE]
Cryptographic hardware does not support service or algorithm.

[CMSERR_FMT_NOT_SUPPORTED]
An unsupported export file format is specified.

[CMSERR_ICSF_FIPS_BAD_ALG_OR_KEY_SIZE]
The algorithm or key size is not supported by ICSF in FIPS mode.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_FIPS]
ICSF is not operating in FIPS mode.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_ATTRIBUTE]
Parameter contents or key attribute value is incorrect.

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_KEY_CANNOT_BE_EXTRACTED]
PKCS #11 key cannot be extracted.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_PW_INCORRECT]
The password is not correct.

Usage

The **gsk_encode_export_key()** function encodes an X.509 certificate and its private key into a PKCS #12 data stream. The certificate chain for the subject certificate is supplied from the pkcs_certificates structure *issuer_certificates*, with the root certificate being the final entry in the array.

The export byte stream contains the requested certificate, its private key, and the certification chain. A partial certification chain is exported if the complete chain is not supplied in *issuer_certificates*.

If the certificate's private key is stored as a secure TKDS private key label:

- Only formats **gskdb_export_pkcs12v3_binary** and **gskdb_export_pkcs12v3_base64**, along with algorithm **x509_alg_pbeWithSha1And3DesCbc**, are supported.

- When the private key was created in the TKDS, it was created with the extractable attribute.
- When using this API, you must have the correct access to the CRYPTOZ class. See Chapter 3, “Using cryptographic features with System SSL,” on page 9 for more information.

gsk_encode_export_request()

Encodes a certification renewal request as described in RFC 2986 (tools.ietf.org/html/rfc2986).

Format

```
#include <gskcms.h>

gsk_status gsk_encode_export_request (
                                       pkcs_cert_request *  request,
                                       gskdb_export_format  format,
                                       gsk_buffer *         stream)
```

Parameters

request
Specifies the certification renewal request.

format
Specifies the export format. These values may be specified:

gskdb_export_der_binary
Binary ASN.1 DERencoded.

gskdb_export_der_base64
Base64 ASN.1 DERencoded.

stream
Returns the byte stream for the encoded certification request. The application should call the **gsk_free_buffer()** routine to release the storage when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_encode_export_request()** routine exports a PKCS #10 certification request. The request can be exported using either the ASN.1 DER encoding for the request or the Base64 encoding of the binary value. A Base64 encoded stream will be in the local code page and will include the encoding header and footer lines.

gsk_encode_issuer_and_serial_number()

Encodes a PKCS #7 *IssuerAndSerialNumber*.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_issuer_and_serial_number (
                                 pkcs_certificate *        certificate,
                                 gsk_buffer *              stream)
```

Parameters

certificate
Specifies an x.509 certificate containing the issuer name and serial number to be encoded.

stream
Returns the ASN.1 DER-encoded stream containing the *IssuerAndSerialNumber* sequence. The application should call the **gsk_free_buffer()** routine to release the storage when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_SELECTION_OUT_OF_RANGE]
Selection is not within the valid range.

[CMSERR_CERTIFICATE_NOT_PROVIDED]
Input certificate is missing.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_encode_issuer_and_serial_number()** routine returns a PKCS #7 (Cryptographic Message Syntax) ASN.1 DER-encoded *IssuerAndSerialNumber* from the input certificate. The *IssuerAndSerialNumber* created by calling **gsk_encode_issuer_and_serial_number()** can be decoded using **gsk_decode_issuer_and_serial_number()**.

gsk_encode_name()

Encodes an X.509 name.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_name (
                             x509_name *          name,
                             gsk_buffer *         stream)
```

Parameters

name
Specifies X.509 name.

stream
Returns the ASN.1 stream for the name. The application should release the stream when it is no longer needed by calling the **gsk_free_buffer()** routine.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_CANT_CONVERT]
A character string contains characters not allowed for the string type.

[ASN_NO_MEMORY]
Insufficient memory is available.

Usage

The **gsk_encode_name()** routine will encode an X.509 name as an ASN.1 stream. The name will be encoded as described in RFC 5280 (tools.ietf.org/html/rfc5280).

The *stringType* field in the x509_rdn_attribute structure will be used to determine the format for an encoded directory string. If it is set to x509_string_unknown, the **gsk_encode_name()** routine attempts to encode the string as an ASN.1 printable string. If the string contains characters not included in the printable string set, the string will be encoded as an ASN.1 UTF-8 string. There are a couple of mandatory exceptions:

- The countryName attribute is always encoded as a printable string
- The dnQualifier attribute is always encoded as a printable string
- The emailAddress attribute is always encoded as an IA5 string
- The domainComponent attribute is always encoded as an IA5 string

gsk_encode_private_key()

Encode a private key.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_private_key (
                                   pkcs_private_key_info *          private_key,
                                   gsk_buffer *                     stream)
```

Parameters

private_key
Specifies the private key.

stream
Returns the ASN.1 stream for the private key. The application should release the stream when it is no longer needed by calling the **gsk_free_buffer()** routine.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_NO_MEMORY]
Insufficient memory is available.

Usage

The **gsk_encode_private_key()** routine will encode a private key as an ASN.1 stream. The name will be encoded as described in RFC 5208 (tools.ietf.org/html/rfc5208). The encoded private key will not be usable on another system if the private key information contains an ICSF key token.

gsk_encode_public_key()

Encode a public key.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_public_key (
                                  x509_public_key_info *          public_key,
                                  gsk_buffer *                    stream)
```

Parameters

public_key
Specifies the public key.

stream
Returns the ASN.1 stream for the public key. The application should release the stream when it is no longer needed by calling the **gsk_free_buffer()** routine.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_NO_MEMORY]
Insufficient memory is available.

Usage

The **gsk_encode_public_key()** routine will encode a public key as an ASN.1 stream. The name will be encoded as described in RFC 5280 (tools.ietf.org/html/rfc5280).

gsk_encode_signature()

Encodes an ASN.1 stream and the accompanying signature.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_signature (
                         gsk_buffer *                    unsigned_stream,
                         x509_algorithm_identifier *     algorithm,
                         gsk_bitstring *                 signature,
                         gsk_buffer *                    signed_stream)
```

Parameters

unsigned_stream
Specifies the unsigned ASN.1 stream.

algorithm
Specifies the algorithm used to compute the signature.

signature
Specifies the signature for the ASN.1 stream.

signed_stream
Returns the encoded signature stream. The application should call the **gsk_free_buffer()** routine to release the encoded stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. This is a possible error:

[ASN_NO_MEMORY]
Insufficient memory is available.

Usage

The **gsk_encode_signature()** routine is used to encode an unsigned ASN.1 stream and the digital signature generated for the stream. The signature is encoded using ASN.1 DER (Distinguished Encoding Rules). The application is responsible for ensuring the validity of the supplied information.

gsk_encode_signer_identifier()

Encodes a PKCS #7 *SignerIdentifier* from a signer certificate.

Format

```
#include <gskcms.h>

gsk_status gsk_encode_signer_identifier (
                                  int *                cms_version,
                                  pkcs_certificate *   signer_certificate,
                                  gsk_buffer *         stream)
```

Parameters

cms_version
Specifies the PKCS #7 Cryptographic Message Syntax version that was used to create the signer identifier as described in RFC 3852:

- Specify 1 to use the certificate's *IssuerAndSerialNumber*.
- Specify 3 to use the certificate's *SubjectKeyIdentifier*.

signer_certificate
Specifies an x.509 certificate that is a signer certificate.

stream
Returns the ASN.1 DER-encoded stream. The application should call the **gsk_free_buffer()** routine to release the storage when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_SELECTION_OUT_OF_RANGE]
Selection is not within the valid range.

[CMSERR_CERTIFICATE_NOT_PROVIDED]
Input certificate not provided.

[CMSERR_PKCS7_CMSVERSION_NOT_SUPPORTED]
PKCS #7 CMS version is not supported.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_encode_signer_identifier()** routine creates a PKCS #7 (Cryptographic Message Syntax) signer identifier according to RFC 3852 and returns the ASN.1 DER-encoded signer identifier. The signer identifier created by calling **gsk_encode_signer_identifier()** can be decoded using **gsk_decode_signer_identifier()**.

gsk_export_certificate()

Exports a certificate.

Format

```
#include <gskcms.h>

gsk_status gsk_export_certificate (
                                  gsk_handle                  db_handle,
                                  const char *                label,
                                  gskdb_export_format         format,
                                  gsk_buffer *                stream)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine. The database must be a key database and not a request database.

label
Specifies the label for the database record. The label is specified in the local code page.

format
Specifies the export format. These values may be specified:

gskdb_export_der_binary
Binary ASN.1 DER-encoded

gskdb_export_der_base64
Base64 ASN.1 DER-encoded

gskdb_export_pkcs7_binary
Binary PKCS #7 Cryptographic Message Syntax

gskdb_export_pkcs7_base64
Base64 PKCS #7 Cryptographic Message Syntax

stream
Return the byte stream for the encoded certificate. The application should call the **gsk_free_buffer()** routine to release the storage when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_LABEL]
No database record label is supplied.

[CMSERR_FMT_NOT_SUPPORTED]
An unsupported export file format is specified.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_NOT_FOUND]
The requested record is not found.

Usage

The **gsk_export_certificate()** routine exports an X.509 certificate. The certificate can be exported using either the ASN.1 DER encoding for the certificate or the Cryptographic Message Syntax (PKCS #7) encoding for the certificate. This can be either the binary value or the Base64 encoding of the binary value. A Base64 encoded stream will be in the local code page and will include the encoding header and footer lines.

The export file will contain just the requested certificate when the DER format is selected. The export file will contain the requested certificate and its certification chain when the PKCS #7 format is selected. A partial certification chain will be exported if the complete chain is not in the database.

gsk_export_certification_request()

Exports a PKCS #10 certification request.

Format

```
#include <gskcms.h>

gsk_status gsk_export_certification_request (
                                     gsk_handle                db_handle,
                                     const char *              label,
                                     gskdb_export_format       format,
                                     gsk_buffer *              stream)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine. The database must be a request database and not a key database.

label
Specifies the label for the database record. The label is specified in the local code page.

format
Specifies the export format. These values may be specified:

gskdb_export_der_binary
Binary ASN.1 DER-encoded

gskdb_export_der_base64
Base64 ASN.1 DER-encoded

stream
Return the byte stream for the encoded certificatation request. The application should call the **gsk_free_buffer()** routine to release the storage when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_LABEL]
No database record label is supplied.

[CMSERR_FMT_NOT_SUPPORTED]
An unsupported export file format is specified.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certification requests.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_NOT_FOUND]
The requested record is not found.

Usage

The **gsk_export_certification_request()** routine exports a PKCS #10 certification request. The request can be exported using either the ASN.1 DER encoding for the request or the Base64 encoding of the binary value. A Base64 encoded stream will be in the local code page and will include the encoding header and footer lines.

gsk_export_key()

Exports a certificate and the associated private key.

Format

```
#include <gskcms.h>

gsk_status gsk_export_key (
                                    gsk_handle           db_handle,
                                    const char *         label,
                                    gskdb_export_format  format,
                                    x509_algorithm_type  algorithm,
                                    const char *         password,
                                    gsk_buffer *         stream)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine. The database must be a key database and not a request database. For a SAF key ring database, the private key may be stored in the SAF database or ICSF's TKDS as either a clear or secure key. It cannot be stored in ICSF's PKDS. For a PKCS #11 token, the private key may be stored in ICSF'S TKDS as either a clear or secure key. When stored in ICSF's TKDS as a secure key, the key must be extractable.

label
Specifies the label for the database record. The label is specified in the local code page.

format
Specifies the export format. These values may be specified:

gskdb_export_pkcs12v1_binary
Binary PKCS #12 Version 1

gskdb_export_pkcs12v1_base64
Base64 PKCS #12 Version 1

gskdb_export_pkcs12v3_binary
Binary PKCS #12 Version 3

gskdb_export_pkcs12v3_base64
Base64 PKCS #12 Version 3

algorithm
Specifies the encryption algorithm for the export file. The strong encryption algorithms may not be available depending upon government export regulations.

These values may be specified for the PKCS #12 Version 1 format:

x509_alg_pb1WithSha1And40BitRc2Cbc
40-bit RC2 with SHA-1 digest

x509_alg_pb1WithSha1And128 BitRc2Cbc
128bit RC2 with SHA-1 digest

x509_alg_pb1WithSha1And40BitRc4
40-bit RC4 with SHA-1 digest

x509_alg_pb1WithSha1And128BitRc4
128-bit RC4 with SHA-1 digest

x509_alg_pb1WithSha1And3DesCbc
Triple DES with SHA-1 digest

These values may be specified for the PKCS #12 Version 3 format:

x509_alg_pbeWithSha1And40BitRc2Cbc
40-bit RC2 with SHA-1 digest

x509_alg_pbeWithSha1And128 BitRc2Cbc
128bit RC2 with SHA-1 digest

x509_alg_pbeWithSha1And40BitRc4
40-bit RC4 with SHA-1 digest

x509_alg_pbeWithSha1And128BitRc4
128-bit RC4 with SHA-1 digest

x509_alg_pbeWithSha1And3DesCbc
Triple DES with SHA-1 digest

In FIPS mode, the export format must be PKCS #12 Version 3 and the encryption algorithm must be **x509_alg_pbeWithSha1And3DesCbc**.

password
Specifies the password for the export file. The password is in the local code page and must consist of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string. The user is prompted to enter the password if NULL is specified for this parameter. If the key that is being exported is a secure private key in the TKDS, the maximum password length is 63 bytes.

stream
Return the byte stream for the encoded certificate. The application should call the **gsk_free_buffer()** routine to release the storage when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_LABEL]
The record label or CA certificate label is not valid.

[CMSERR_BAD_RNG_OUTPUT]
In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_CRYPTO_HARDWARE_NOT_AVAILABLE]
Cryptographic hardware does not support service or algorithm.

[CMSERR_FMT_NOT_SUPPORTED]
An unsupported export file format is specified.

[CMSERR_ICSF_FIPS_BAD_ALG_OR_KEY_SIZE]
The algorithm or key size is not supported by ICSF in FIPS mode.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_FIPS]
ICSF is not operating in FIPS mode.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_ATTRIBUTE]
Parameter contents or key attribute value is incorrect.

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_KEY_CANNOT_BE_EXTRACTED]
PKCS #11 key cannot be extracted.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The signer certificate does not have a private key.

[CMSERR_RECORD_NOT_FOUND]
The requested record is not found.

Usage

The **gsk_export_key()** routine exports an X.509 certificate and the associated private key. The certificate can be exported using either the PKCS #12 Version 1 format or the PKCS #12 Version 3 format. This can be either the binary value or the Base64 encoding of the binary value. A Base64 encoded stream will be in the local code page and will include the encoding header and footer lines.

The PKCS #12 Version 1 format is obsolete. However, it is the only format supported by some SSL implementations and must be used when moving a certificate and key to one of those systems. If not running in FIPS mode, you should use either x509_alg_pb1WithSha1And40BitRc2Cbc or x509_alg_pb1withSha1And3DesCbc for interoperability with these older SSL implementations.

The export file will contain the requested certificate, its private key, and the certification chain. A partial certification chain will be exported if the complete chain is not in the database.

If the certificate's private key is stored as a secure TKDS private key label:

- Only formats **gskdb_export_pkcs12v3_binary** and **gskdb_export_pkcs12v3_base64**, along with algorithm **x509_alg_pbeWithSha1And3DesCbc**, are supported.
- When the private key was created in the TKDS, it must be created with the extractable attribute.
- When using this API, you must have the correct access to the CRYPTOZ class. See Chapter 3, “Using cryptographic features with System SSL,” on page 9 for more information.

gsk_factor_private_key()

Factorizes a private key into its component values.

Format

```
#include <gskcms.h>

gsk_status gsk_factor_private_key(
                         pkcs_private_key_info *    private_key,
                         gsk_private_key *          private_key_factors)
```

Parameters

private_key
Specifies the private key.

private_key_factors
Returns the private key components.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_ELEMENTS_MISSING]
Required data element is missing.

[CMSERR_ALG_NOT_SUPPORTED]
Cryptographic algorithm is not supported.

[CMSERR_PRIVATE_KEY_INFO_NOT_SUPPLIED]
Private key information not supplied.

[CMSERR_PRIVATE_KEY_NOT_SUPPLIED]
Private key structure not supplied.

[CMSERR_STRUCTURE_TOO_SMALL]
Size specified for supplied structure is too small.

Usage

The **gsk_factor_private_key()** function deconstructs the private key into its private key components, formatted for use with ICSF PKCS #11 tokens.

Before calling the function, the application must initialize the size field in *private_key_factors* to the size of the gsk_private_key structure. It must also prime *private_key* with the appropriate private key to be factorized before calling the routine.

The routine will return the factorized components of the private key in *private_key_factors*. The x509_algorithm_identifier is set with the appropriate value for the private key type when returned.

gsk_factor_private_key_rsa()

Factorizes an RSA private key into its component values.

Note: This function is deprecated. Use **gsk_factor_private_key()** instead.

Format

```
#include <gskcms.h>

gsk_status gsk_factor_private_key_rsa (
                                       pkcs_private_key_info * private_key,
                                       gsk_buffer *            modulus,
                                       gsk_buffer *            public_exponent,
                                       gsk_buffer *            private_exponent,
                                       gsk_buffer *            prime1,
                                       gsk_buffer *            prime2,
                                       gsk_buffer *            prime_exponent1,
                                       gsk_buffer *            prime_exponent2,
                                       gsk_buffer *            coefficient)
```

Parameters

private_key
Specifies the private key.

modulus
Returns the modulus (n).

public_exponent
Returns the public exponent (e).

private_exponent
Returns the private exponent (d).

prime1
Returns the 1st prime (p).

prime2
Returns the 2nd prime (q).

prime_exponent1
Returns the private exponent d modulo p-1.

prime_exponent2
Returns the private exponent d modulo q-1.

coefficient
Returns the CRT coefficient q^{-1} mod p.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_ELEMENTS_MISSING]
Required data element is missing.

Usage

The **gsk_factor_private_key_rsa()** function deconstructs the pkcs_private_key_info into its RSA private key components.

gsk_factor_public_key()

Factorizes a public key into its component values.

Format

```
#include <gskcms.h>

gsk_status gsk_factor_public_key(
                                x509_public_key_info *     public_key,
                                gsk_public_key *           public_key_factors)
```

Parameters

public_key
Specifies the public key.

public_key_factors
Returns the public key components.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_ELEMENTS_MISSING]
Required data element is missing.

[CMSERR_ALG_NOT_SUPPORTED]
Cryptographic algorithm not supported.

[CMSERR_PUBLIC_KEY_INFO_NOT_SUPPLIED]
Public key information not supplied.

[CMSERR_PUBLIC_KEY_NOT_SUPPLIED]
Public key structure not supplied.

[CMSERR_STRUCTURE_TOO_SMALL]
Size specified for supplied structure is too small.

Usage

The **gsk_factor_public_key()** function deconstructs the public key into its public key components, formatted for use with ICSF PKCS #11 tokens.

Before calling the function, the application must initialize the size field in *public_key_factors* to the size of the gsk_public_key structure. It must also prime *public_key* with the appropriate public key to be factorized before calling the routine.

The routine will return the factorized component of the public key in *public_key_factors*. The x509_algorithm_identifier is set with the appropriate value for the public key type when returned.

gsk_factor_public_key_rsa()

Factorizes an RSA public key into its component values.

Note: This function is deprecated. Use **gsk_factor_public_key()** instead.

Format

```
#include <gskcms.h>

gsk_status gsk_factor_public_key_rsa (
                                       x509_public_key_info *  public_key,
                                       gsk_uint32 *            modulus_bits,
                                       gsk_buffer *            modulus,
                                       gsk_buffer *            exponent)
```

Parameters

public_key
Specifies the public key.

modulus_bits
Returns the length of the modulus in bits.

modulus
Returns the modulus (n).

exponent
Returns the public exponent (e).

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_ELEMENTS_MISSING]
Required data element is missing.

Usage

The **gsk_factor_public_key_rsa()** function deconstructs the pkcs_public_key_info into its RSA public key components.

gsk_fips_state_query()

Queries the current state of FIPS mode.

Format

```
gsk_status gsk_fips_state_query(
                                GSK_FIPS_STATE_ENUM_VALUE *  enum_value)
```

Parameters

enum_value
Returns the FIPS state enumeration value.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file.

Usage

The **gsk_fips_state query** function returns an enumerated value indicating the current FIPS mode state of System SSL. One of the following enumerated values will be returned:

GSK_FIPS_STATE_NOTSET
FIPS mode state has not yet been set.

GSK_FIPS_STATE_ON
FIPS mode state has been set to FIPS mode.

GSK_FIPS_STATE_OFF
FIPS mode state has been set to non-FIPS mode.

GSK_FIPS_STATE_LEVEL1
FIPS mode state has been set to FIPS LEVEL1 mode.

GSK_FIPS_STATE_LEVEL2
FIPS mode state has been set to FIPS LEVEL2 mode.

GSK_FIPS_STATE_LEVEL3
FIPS mode state has been set to FIPS LEVEL3 mode.

gsk_fips_state_set()

Sets the state of FIPS mode for System SSL.

Format

```
gsk_status gsk_fips_state_set(
                              GSK_FIPS_STATE_ENUM_VALUE  enum_value)
```

Parameters

enum_value
Specifies the FIPS state enumeration value.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. The following are some possible errors:

[CMSERR_ATTRIBUTE_INVALID_ENUMERATION]
The enumeration value is not valid or it cannot be set because of the current state.

[CMSERR_FIPS_MODE_EXECUTE_FAILED]
The request to execute in FIPS mode failed because the Cryptographic Services Security Level 3 FMID is not installed so that the required System SSL DLLs could not be loaded.

[CMSERR_FIPS_MODE_SWITCH]
The System SSL FIPS mode state cannot be changed to FIPS mode because it is currently not in FIPS mode.

[CMSERR_FIPS_MODE_SWITCH_LEVEL]
The System SSL FIPS mode LEVEL state cannot be changed to another FIPS mode LEVEL.

[CMSERR_KATPW_FAILED]
The power-on known answer tests failed. FIPS mode cannot be set.

[CMSERR_KATPW_ICSF_FAILED]
The power-on known answer tests failed. Either ICSF was not available or FIPS mode was disabled. FIPS mode cannot be set.

Usage

The **gsk_fips_state_set()** routine sets the enumerated value for the System SSL FIPS mode state.

The FIPS mode setting applies to the entire process. Once set, then all threads of the same process execute in FIPS mode. If any thread switches to non-FIPS mode, then all threads in the same process execute in non-FIPS mode.

In order to set FIPS mode, this function must be executed before all other System SSL API functions except for **gsk_get_cms_vector(), gsk_get_ssl_vector()**, and **gsk_fips_state_query()**. It is possible to switch to a non-FIPS mode at a later time. It is not possible to switch from non-FIPS mode to FIPS mode at any time.

The following enumerated values are supported:

GSK_FIPS_STATE_ON
FIPS mode state has been set to FIPS mode.

GSK_FIPS_STATE_OFF
FIPS mode state has been set to non-FIPS mode.

GSK_FIPS_STATE_LEVEL1
FIPS mode state has been set to FIPS LEVEL1 mode.

GSK_FIPS_STATE_LEVEL2
FIPS mode state has been set to FIPS LEVEL2 mode.

GSK_FIPS_STATE_LEVEL3
FIPS mode state has been set to FIPS LEVEL3 mode.

The meaning of level in the preceding enumerator values does not correspond to the different FIPS 140-2 Validation Levels. Level is being used by System SSL to indicate different security key enforcements performed by System SSL for FIPS 140-2 Level 1 (Pre and Post NIST SP800-131Ar1/NIST SP800-131Ar2).

If using GSK_FIPS_STATE_ON or GSK_FIPS_STATE_LEVEL1, the default minimum supported peer key sizes are:

- 2048 for Diffie-Hellman.
- 1024 for DSA.
- NIST 192 for Elliptic Curve.
- 1024 for RSA.

If setting FIPS mode to GSK_FIPS_STATE_LEVEL2 or GSK_FIPS_STATE_LEVEL3, set the following System SSL environment variables to the supported key sizes for that security level:

Diffie-Hellman
Set GSK_PEER_DH_MIN_KEY_SIZE to 2048.

DSA
Set GSK_PEER_DSA_MIN_KEY_SIZE to 2048.

Elliptic Curve
Set GSK_PEER_ECC_MIN_KEY_SIZE to:

- 192 for GSK_FIPS_STATE_LEVEL2.
- 224 for GSK_FIPS_STATE_LEVEL3.

RSA
Set GSK_PEER_RSA_MIN_KEY_SIZE to 2048.

If a FIPS mode LEVEL has been set, changing the FIPS mode state to anything other than GSK_FIPS_STATE_OFF is not supported.

When calling the **gsk_fips_state_set()** API to set FIPS mode, each cryptographic algorithm implemented through System SSL software goes through a validation test. The validation test, also known as a Known Answer Test, involves exercising the algorithm and comparing the results for correctness. See the **gsk_perform_kat()** API for the list of algorithms validated. If an issue is encountered during the validation, **gsk_fips_state_set()** will fail with return code CMSERR_KATPW_FAILED or CMSERR_KATPW_ICSF_FAILED.

For more information about FIPS level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_format_long_integer()

Formats a long integer value into a printable string.

Format

```
#include <gskcms.h>

gsk_status gsk_format_long_integer (
                                    gsk_octet *     value_data,
                                    gsk_size        value_length,
                                    char **         formatted_string)
```

Parameters

value_data
Specifies the data containing the long integer value that will be formatted into a string.

value_length
Specifies the data value’s length in bytes. The valid values are 1 through 100.

formatted_string
Returns the formatted long integer string.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_INVALID_INPUT_PARAMETER]
Parameter *value_data* is not valid.

[CMSERR_INVALID_NUMERIC_VALUE]
Parameter *value_length* is not valid.

Usage

The **gsk_format_long_integer()** routine formats the long integer contained in *value_data* for the length of *value_length* and returns the result in *formatted_string*. The application should call the **gsk_free_string()** routine to release the string when it is no longer needed.

gsk_format_time()

Formats the time value into a Universal Coordinated Time (UTC) or Generalized Time character string.

Format

```
#include <gskcms.h>

gsk_status gsk_format_time (
                            gsk_timeval *        timeVal,
                            gsk_time_format *    timeFormat,
                            char **              timeString)
```

Parameters

timeVal
Specifies the time to be converted into a character string. The specified time must already have been converted to Greenwich Mean time or Zulu time.

timeFormat
Returns the time format of the *timeString* character string. The supported time formats are:

gsk_utc_time
YYMMDDhhmmssZ

gsk_generalized_time
YYYYMMDDhhmmssZ

timeString
Returns the formatted time string. The application should call the **gsk_free_string()** routine to release the string when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_TIME_VALUE]
Specified time structure contents are not valid.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_INVALID_OUTPUT_PARAMETER]
The time string specified or the time format specified or both the time string and the time format specified is null.

Usage

The **gsk_format_time()** routine formats the time specified in a *gsk_timeval* variable into a character string based on its value. If the year value of the time specified is less than 2050, a null-terminated string in UTC format is returned in the *timeString* field. If the year is 2050 or greater, a null-terminated string in Generalized Time format is returned in the *timeString* field.

The year of the *timeVal* that is passed in must be 1970 - 9999 and is the actual year minus 1900. Therefore:

- *tm_year* must be 70 - 8099.
- *tm_mon* must be 0 - 11.
- *tm_day* must be 1 - 31.
- *tm_hour* must be 0 - 23.

- *tm_min* must be 0 - 59.
- *tm_sec* must be 0 - 59.

The *timeType* is a returned value specifying the type of time returned. The value is *gsk_utc_time* if the time returned is in UTC format and is *gsk_generalized_time* if the time returned is in Generalized Time format.

No conversions between time zones is performed. No validation is performed to ensure that dates and times are valid beyond the requirements described previously.

gsk_free_attributes_signers()

Releases storage allocated for gsk_attributes_signers structure.

Format

```
#include <gskcms.h>

void gsk_free_attributes_signers (
                                   gsk_attributes_signers *   attributesSigners)
```

Parameters

attributesSigners
Specifies the gsk_attributes_signers structure to be released. The gsk_attributes_signers structure will be initialized to zero upon completion.

Usage

The **gsk_free_attributes_signers()** routine is used to release storage allocated for gsk_attributes_signers structure.

gsk_free_buffer()

Releases storage allocated for a buffer.

Format

```
#include <gskcms.h>

void gsk_free_buffer (
                          gsk_buffer *          buffer)
```

Parameters

buffer
Specifies the buffer to be released. The gsk_buffer structure will be initialized to zero upon completion.

Usage

The **gsk_free_buffer()** routine is used to release storage allocated for a buffer.

gsk_free_certificate()

Releases storage allocated for an X.509 certificate.

Format

```
#include <gskcms.h>

void gsk_free_certificate (
                          x509 certificate *         certificate)
```

Parameters

certificate
Specifies the certificate to be released. The x509_certificate structure will be initialized to zero upon completion.

Usage

The **gsk_free_certificate()** routine is used to release storage allocated for an X.509 certificate.

gsk_free_certificates()

Releases storage allocated for an array of certificates.

Format

```
#include <gskcms.h>

void gsk_free_certificates (
                                pkcs_certificates *        certificates)
```

Parameters

certificates
Specifies the certificates to be released. The pkcs_certificates structure will be initialized to zero upon completion.

Usage

The **gsk_free_certificates()** routine is used to release storage allocated for an array of certificates.

gsk_free_certificate_extension()

Releases storage allocated for an X.509 certificate extension.

Format

```
#include <gskcms.h>

void gsk_free_certificate_extension (
                                      x509_extension *    extension)
```

Parameters

extension

Specifies the certificate extension to be released. The x509_extension structure will be initialized to zero upon completion.

Usage

The **gsk_free_certificate_extension()** routine is used to release storage allocated for an X.509 certificate extension.

gsk_free_certification_request()

Releases storage allocated for a PKCS certification request.

Format

```
#include <gskcms.h>

void gsk_free_certification_request (
                                      pkcs_cert_request *        request)
```

Parameters

request

Specifies the certification request to be released. The pkcs_cert_request structure will be initialized to zero upon completion.

Usage

The **gsk_free_certification_request()** routine is used to release storage allocated for a Public Key Cryptography Standards (PKCS) certification request.

gsk_free_content_info()

Releases storage allocated for PKCS #7 content information.

Format

```
#include <gskcms.h>

void gsk_free_content_info (
                             pkcs_content_info *         content_info)
```

Parameters

content_info
Specifies the content information to be released. The pkcs_content_info structure will be initialized to zero upon completion.

Usage

The **gsk_free_content_info()** routine is used to release storage allocated for a Public Key Cryptography Standards (PKCS) content information.

gsk_free_crl()

Releases storage allocated for an X.509 certificate revocation list.

Format

```
#include <gskcms.h>

void gsk_free_crl (
                    x509_crl *          crl)
```

Parameters

crl
Specifies the certificate revocation list to be released. The x509_crl structure will be initialized to zero upon completion.

Usage

The **gsk_free_crl()** routine is used to release storage allocated for an X.509 certificate revocation list.

gsk_free_crls()

Releases storage allocated for an array of X.509 certificate revocation lists.

Format

```
#include <gskcms.h>

void gsk_free_crls (
                     x509_crls *         crls)
```

Parameters

crls

Specifies the array of certificate revocation lists to be released. The x509_crls structure will be initialized to zero upon completion.

Usage

The **gsk_free_crls()** routine is used to release storage allocated for an array of X.509 certificate revocation lists.

gsk_free_decoded_extension()

Frees a decoded certificate extension.

Format

```
#include <gskcms.h>

void gsk_free_decoded_extension (
                        x509_decoded_extension *        decoded_extension)
```

Parameters

decoded_extension
Specifies the certificate extension to be released. The x509_decoded_extension structure will be initialized to zero upon completion.

Usage

The **gsk_free_decoded_extension()** routine is used to release storage allocated for a decoded X.509 certificate extension.

gsk_free_issuer_and_serial_number()

Releases storage allocated for PKCS #7 issuer and serial number information.

Format

```
#include <gskcms.h>

void gsk_free_issuer_and_serial_number (
                                    pkcs_issuer_serial *      issuer_serial)
```

Parameters

issuer_serial
Specifies the issuer and serial number structure *pkcs_issuer_serial* to be released.

Usage

The **gsk_free_issuer_and_serial_number()** routine is used to release storage allocated for PKCS #7 issuer and serial information.

gsk_free_name()

Releases storage allocated for an X.509 name.

Format

```
#include <gskcms.h>

void gsk_free_name (
                     x509_name *         name)
```

Parameters

name
Specifies the name to be released. The x509_name structure will be initialized to zero upon completion.

Usage

The **gsk_free_name()** routine is used to release storage allocated for an X.509 name.

gsk_free_oid()

Releases storage allocated for OID information.

Format

```
#include <gskcms.h>

void gsk_free_oid (
                    gsk_oid *        oid)
```

Parameters

oid
Specifies the OID to be released.

Usage

The **gsk_free_oid()** routine is used to release storage allocated for OID.

gsk_free_private_key()

Releases storage allocated for private key information.

Format

```
#include <gskcms.h>

gsk_status gsk_free_private_key(
                                 gsk_private_key *          private_key_factors)
```

Parameters

private_key_factors
Specifies the private key components. The gsk_private_key structure will be initialized to zero upon completion.

Usage

The **gsk_free_private_key()** routine is used to release storage allocated for private key component information.

gsk_free_private_key_info()

Releases storage allocated for private key information.

Format

```
#include <gskcms.h>

void gsk_free_private_key_info (
                                pkcs_private_key_info *         info)
```

Parameters

info
: Specifies the private key information to be released. The pkcs_private_key_info structure will be initialized to zero upon completion.

Usage

The **gsk_free_private_key_info()** routine is used to release storage allocated for private key information.

gsk_free_public_key()

Releases storage allocated for public key information.

Format

```
#include <gskcms.h>

gsk_status gsk_free_public_key(
                                gsk_public_key *          public_key_factors)
```

Parameters

public_key_factors
Specifies the public key components. The **gsk_free_public_key** structure will be initialized to zero upon completion.

Usage

The **gsk_free_public_key()** routine is used to release storage allocated for public key component information.

gsk_free_public_key_info()

Releases storage allocated for public key information.

Format

```
#include <gskcms.h>

void gsk_free_public_key_info (
                                x509_public_key_info *        info)
```

Parameters

info
Specifies the public key information to be released. The x509_public_key_info structure will be initialized to zero upon completion.

Usage

The **gsk_free_public_key_info()** routine is used to release storage allocated for public key information.

gsk_free_record()

Releases storage allocated for a database record.

Format

```
#include <gskcms.h>

void gsk_free_record (
                       gskdb_record *         record)
```

Parameters

record
Specifies the database record to be released. The gskdb_record structure is released in addition to the record data.

Usage

The **gsk_free_record()** routine is used to release storage allocated for a database record.

gsk_free_records()

Releases storage allocated for an array of database records.

Format

```
#include <gskcms.h>

void gsk_free_records (
                        int                     num_records,
                        gskdb_record **         records)
```

Parameters

num_records
Specifies the number of records in the array.

records
Specifies the database record array to be released. The gskdb_record structures are released in addition to the record data.

Usage

The **gsk_free_records()** routine is used to release storage allocated for an array of database records.

gsk_free_revocation_source()

Frees the revocation source initialized and allocated by **gsk_create_revocation_source()**.

Format

```
#include <gskcms.h>

void gsk_free_revocation_source (
                               gsk_handle *              revocation_handle)
```

Parameters

revocation_handle
Specifies a revocation source handle created by the **gsk_create_revocation_source()** routine.

Usage

The **gsk_free_revocation_source()** routine is used to release storage allocated for a revocation source handle.

gsk_free_signer_identifier()

Releases storage allocated for PKCS #7 signer identifier information.

Format

```
#include <gskcms.h>

void gsk_free_signer_identifier (
                                  int *                 cms_version,
                                  pkcs_signer_id *      signer_id)
```

Parameters

cms_version
Specifies the PKCS #7 Cryptographic Message Syntax version that was used to create the signer identifier as described in RFC 3852:

- Specify 1 to use the certificate's *IssuerAndSerialNumber*.
- Specify 3 to use the certificate's *SubjectKeyIdentifier*.

signer_id
Specifies the PKCS #7 signer identifier to be released.

Usage

The **gsk_free_signer_identifier()** routine is used to release storage allocated for PKCS #7 signer identifier.

gsk_free_string()

Releases storage allocated for a string.

Format

```
#include <gskcms.h>

void gsk_free_string (
                       char *         string)
```

Parameters

string
Specifies the string to be released.

Usage

The **gsk_free_string()** routine is used to release storage allocated for a string.

gsk_free_strings()

Releases storage allocated for an array of strings.

Format

```
#include <gskcms.h>

void gsk_free_strings (
                        int             num_strings,
                        char **         strings)
```

Parameters

num_strings
Specifies the number of strings in the array.

strings
Specifies the array of strings to be released.

Usage

The **gsk_free_strings()** routine is used to release storage allocated for an array of strings.

gsk_free_x509_diagnostics()

Releases the storage allocated for an X.509 diagnostic summary.

Format

```
#include <gskcms.h>

void gsk_free_x509_diagnostics (
                                 x509_diag_summary *    diag_summary)
```

Parameters

diag_summary
Specifies the X.509 diagnostic summary to be released.

Usage

The **gsk_free_x509_diagnostics()** routine releases the storage allocated for an X.509 diagnostic summary. The *diag_summary* structure will be initialized to zero upon completion.

gsk_generate_key_agreement_pair()

Generates a Diffie-Hellman public/private key pair.

Format

```
#include <gskcms.h>

gsk_status gsk_generate_key_agreement_pair (
                                   gsk_buffer *        key_params,
                                   gsk_buffer *        public_value,
                                   gsk_buffer *        private_value)
```

Parameters

key_params
Specifies the Diffie-Hellman key parameters as an ASN.1-encoded sequence.

public_value
Returns the generated public value as a binary byte string. The application should call the **gsk_free_buffer()** routine to release the public value when it is no longer needed.

private_value
Returns the generated private value as a binary byte string. The application should call the **gsk_free_buffer()** routine to release the private value when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_DH_PARAMS]
The Diffie-Hellman group parameters are not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_RNG_OUTPUT]
In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_generate_key_agreement_pair()** routine will generate a Diffie-Hellman public/private key pair using ICSF when executing in FIPS mode, and as recommended by PKCS #3 (Diffie-Hellman Key Agreement Standard) and RFC 2631 (tools.ietf.org/html/rfc2631) when in non-FIPS mode. The required key parameters P and G and the optional key parameters Q and J are supplied as an ASN.1-encoded sequence as defined in either PKCS #3 or RFC 5280 (tools.ietf.org/html/rfc5280). The return values will be the binary values for Y and X. The key size is determined by the size of the modulus P and must be between 512 and 2048 bits if not executing in FIPS mode, and must be 2048 bits if executing in FIPS mode. The private value X will be less than Q-1 if Q is present in the key parameters, otherwise the private value X will be less than P-1.

Multiple Diffie-Hellman Key Agreement key pairs can share the same group parameters (P and G). This is useful when generating multiple keys of the same type since it is very time-consuming to compute values for P and G. In addition, the Diffie-Hellman key agreement algorithm requires both parties to use the same group parameters when computing the shared secret value.

gsk_generate_key_pair()

Generates a public/private key pair.

Format

```
#include <gskcms.h>

gsk_status gsk_generate_key_pair (
                          x509_algorithm_type         key_algorithm,
                          int                         key_size,
                          gsk_buffer *                key_params,
                          x509_public_key_info *      public_key,
                          pkcs_private_key_info *     private_key,
                          gsk_buffer *                key_identifier)
```

Parameters

key_algorithm
Specifies the key algorithm.

key_size
Specifies the key size in bits.

key_params
Specifies the key parameters as an ASN.1-encoded sequence. Specify NULL for this parameter if the key algorithm does not require any parameters.

public_key
Returns the generated public key. The application should call the **gsk_free_public_key_info()** routine to release the public key when it is no longer needed.

private_key
Returns the generated private key. The application should call the **gsk_free_private_key_info()** routine to release the private key when it is no longer needed.

key_identifier
Returns the key identifier for the generated public key. The application should call the **gsk_free_buffer()** routine to release the key identifier when it is no longer needed. Specify NULL for this parameter if the key identifier is not needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The key algorithm is not supported.

[CMSERR_BAD_DH_PARAMS]
The Diffie-Hellman group parameters are not valid.

[CMSERR_BAD_DSA_PARAMS]
The DSS parameters are not valid.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_FIPS_KEY_PAIR_CONSISTENCY]
FIPS mode key generation failed pair-wise consistency check.

[CMSERR_ICSF_CLEAR_KEY_SUPPORT_NOT_AVAILABLE]
Clear key support not available due to ICSF key policy.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_generate_key_pair()** routine will generate a public/private key pair. The format of the public and private key values returned by the **gsk_generate_key_pair()** routine is defined in RFC 5280 (tools.ietf.org/html/rfc5280).

These key algorithms are supported:

- **x509_alg_rsaEncryption - RSA Encryption - {1.2.840.113549.1.1.1}**

  The key size must be between 512 and 4096 bits if not executing in FIPS mode, and must be between 1024 and 4096 bits if executing in FIPS mode, and will be rounded up to a multiple of 16 bits if necessary. No key parameters are used. The key size determines the size of the modulus N.

- **x509_alg_idDsa - Digital Signature Standard - {1.2.840.10040.4.1}**

  The key size can be between 512 and 1024 bits, which will be rounded up to a multiple of 64 bits, or precisely 2048 bits. Key sizes less than 1024 bits can only be generated in non-FIPS mode and are generated according to FIPS 186-2: Digital Signature Standard (DSS) (csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf). Keys sizes 1024 and 2048 are generated according to FIPS 186-4: Digital Signature Standard (DSS) (nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf). The key parameters are the prime p, the prime divisor q, and the generator g. The requested key size must be the same as the size of the prime p. Note that key parameters that contain a p of 2048 bits and a q of 160 bits do not conform to FIPS 186-4 and are not supported. The **gsk_generate_key_parameters()** routine can be used to generate the key parameters.

- **x509_alg_dhPublicNumber - Diffie-Hellman Key Exchange - {1.2.840.10046.2.1}**

  The key size must be between 512 and 2048 bits if not executing in FIPS mode, and must be 2048 bits if executing in FIPS mode, and will be rounded up to a multiple of 64 bits if necessary. The key parameters are the prime P, the base G, the subprime Q, and the subgroup factor J. The requested key size must be the same as the size of the prime P. The **gsk_generate_key_parameters()** routine can be used to generate the key parameters.

  In non-FIPS mode, the subprime Q and the subgroup factor J are optional key parameters. This allows the **gsk_generate_key_pair()** routine to accept key parameters generated in accordance with PKCS #3 (Diffie-Hellman Key Agreement Standard) including key parameters generated in accordance with RFC 2631 (tools.ietf.org/html/rfc2631). The private value X will be less than Q-1 if Q is present in the key parameters, otherwise the private value X will be less than P-1.

  Multiple Digital Signature Standard keys or Diffie-Hellman Key Exchange keys can share the same group parameters (P, Q, and G). This is useful when generating multiple keys of the same type since it is very time-consuming to compute values for P, Q, and G. In addition, the Diffie-Hellman key agreement algorithm requires both parties to use the same group parameters when computing the secret value.

- **x509_alg_ecPublicKey – ECDSA and ECDH Public Key - {1.2.840.10045.2.1}**

  The EC named curve used to generate the ECC key pair can be specified using either the *key_params* buffer or the *key_size* parameter. If the *key_params* buffer is supplied, the *key_size* parameter will be ignored. The *key_params* buffer must contain ASN.1 encoded EC domain parameters, or be NULL. If the *key_params* buffer is NULL, the *key_size* parameter will be rounded up to the nearest supported key size and the default EC named curve for that key size will be used, as specified in Table 3 on page 13. In FIPS mode, only NIST recommended curves are supported.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

For more information about FIPS supported algorithms and key sizes, see “Algorithms and key sizes” on page 19 and FIPS 140-2.

gsk_generate_key_parameters()

Generates ASN.1 encoded key parameters.

Format

```
#include <gskcms.h>

gsk_status gsk_generate_key_parameters(
                                   x509_algorithm_type     key_algorithm,
                                   int                     key_size,
                                   gsk_buffer *            key_params )
```

Parameters

key_algorithm
Specifies the key algorithm.

key_size
Specifies the key size in bits.

key_params
Specifies the key parameters as an ASN.1-encoded sequence. The application should call the **gsk_free_buffer()** routine to release the key parameters when they are no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The key algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_generate_key_parameters()** routine will generate key parameters that can then be used with the **gsk_generate_key_pair()** routine to generate one or more public/private key pairs.

These key algorithms are supported:

- **x509_alg_idDsa - Digital Signature Standard - {1.2.840.10040.4.1}**

  The key size can be between 512 and 1024 bits, which will be rounded up to a multiple of 64 bits, or precisely 2048 bits. Key sizes less than 1024 bits can only be generated in non-FIPS mode and are generated according to FIPS 186-2. Keys sizes 1024 and 2048 are generated according to FIPS 186-4. The generated ASN.1 sequence will consist of the prime P, the subprime Q, and the base G. For 2048-bit key size, the size of the subprime Q will be 256. See FIPS 186-4: Digital Signature Standard (DSS) (nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf) for more information about the generation of the key parameters for 1024-bit and greater key sizes. See FIPS 186-2: Digital Signature Standard (DSS) (csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf) for smaller key sizes.

- **x509_alg_dhPublicNumber - Diffie-Hellman Key Exchange - {1.2.840.10046.2.1}**

  The key size must be between 512 and 2048 bits if not executing in FIPS mode, and must be 2048 bits if executing in FIPS mode, and will be rounded up to a multiple of 64 bits if necessary. In non-FIPS mode, the generated ASN.1 sequence will consist of the prime P, the base G, the subprime Q, and the subgroup factor J. In FIPS mode, the generated ASN.1 sequence will consist of the prime P and the base G. See RFC 2631 (tools.ietf.org/html/rfc2631) for more information about the generation of the key parameters, and RFC 5280 (tools.ietf.org/html/rfc5280) for more information about the ASN.1 encoding.

Multiple Digital Signature Standard keys or Diffie-Hellman Key Exchange keys can share the same group parameters (P, Q, and G). This is useful when generating multiple keys of the same type since it is very time-consuming to compute values for P, Q, and G. In addition, the Diffie-Hellman key agreement algorithm requires both parties to use the same group parameters when computing the secret value (an SSL client will generate temporary Diffie-Hellman values if the group parameters in the client certificate are not the same as the group parameters in the server certificate).

- **x509_alg_ecPublicKey – ECDSA and ECDH Public Key - {1.2.840.10045.2.1}**

  The key size must be between 0 and 521 bits. The key size value will be rounded up to the nearest supported key size, and the default EC named curve for that key size will be used, as specified in Table 3 on page 13. In FIPS mode, only NIST recommended curves are supported.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

For more information about FIPS supported algorithms and key sizes, see “Algorithms and key sizes” on page 19 and FIPS 140-2.

gsk_generate_random_bytes()

Generates a random byte stream.

Format

```
#include <gskcms.h>

gsk_status gsk_generate_random_bytes (
                                            gsk_buffer *          buffer)
```

Parameters

buffer
Specifies the buffer for the random byte stream. The application is responsible for providing the buffer and setting the *length* and *data* fields appropriately.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_RNG]
Generate random bytes input buffer not valid.

[CMSERR_BAD_RNG_OUTPUT]
Generate random bytes produced duplicate output.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

Usage

The **gsk_generate_random_bytes()** routine will return a random byte stream. The application provides the buffer for the byte stream. The length value determines how many bytes will be generated.

System SSL attempts to use the ICSF PKCS #11 pseudo-random callable service (CSFPPRF) to generate a random byte stream. If ICSF is unavailable or returns an error and System SSL is in non-FIPS mode, an internal RNG will be used to generate the random data. If System SSL is in FIPS mode, the API call will fail.

The contents of the generated byte stream can be modified by setting the GSK_RNG_ALLOW_ZERO_BYTES environment variable. A GSK_RNG_ALLOW_ZERO_BYTES setting of TRUE, ON, or 1 will retain bytes with a zero value in the random byte stream. A setting of FALSE, OFF, or 0 will remove bytes with a zero value from the random byte stream. The default setting is TRUE.

Note: The GSK_RNG_ALLOW_ZERO_BYTES environment variable is processed during System SSL initialization and is not checked afterward.

gsk_generate_secret()

Generates the Diffie-Hellman shared secret.

Format

```
#include <gskcms.h>

gsk_status gsk_generate_secret (
                                gsk_buffer *        key_params,
                                gsk_buffer *        public_value,
                                gsk_buffer *        private_value
                                gsk_buffer *        secret_value)
```

Parameters

key_params
Specifies the Diffie-Hellman key parameters as an ASN.1-encoded sequence.

public_value
Specifies the public value for the partner application as a binary byte string.

private_value
Specifies the private value for the local application as a binary byte string.

secret_value
Returns the secret value as a binary byte string. The application should call the **gsk_free_buffer()** routine to release the secret value when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_DH_PARAMS]
The Diffie-Hellman group parameters are not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_generate_secret()** routine will generate the Diffie-Hellman shared secret value as defined in PKCS #3 (Diffie-Hellman Key Agreement Standard) and RFC 2631 (tools.ietf.org/html/rfc2631). The required key parameters P and G, and, in non-FIPS mode, the optional key parameters Q and J are supplied as an ASN.1-encoded sequence as defined in either PKCS #3 or RFC 5280 (tools.ietf.org/html/rfc5280). The return value will be the binary value for Z. The key size is determined by the size of the modulus P, and must be between 512 and 2048 bits if not executing in FIPS mode, or it must be 2048 bits if in FIPS mode.

gsk_get_certificate_algorithms()

Get the public key and certificate signature algorithms for a database record.

Format

```
#include <gskcms.h>

gsk_status gsk_get_certificate_algorithms (
                              gsk_handle *             db_handle,
                              const char *             label,
                              x509_algorithm_type *    public_key_algorithm,
                              x509_algorithm_type *    signature_algorithm,
                              x509_algorithm_type *    signature_key_algorithm)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine. The database must be a key database, SAF key ring or z/OS PKCS #11 token.

label
Specifies the label for the database record. The label is specified in the local code page.

public_key_algorithm
Returns the key algorithm for the subject public key in the certificate.

signature_algorithm
Returns the signature algorithm used to sign the certificate.

signature_key_algorithm
Returns the signature key algorithm used to sign the certificate.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_INCORRECT_DBTYPE]
The database does not support this operation.

[CMSERR_MULTIPLE_LABEL]
Multiple certificates exist for label.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_DELETED]
The requested record is deleted.

[CMSERR_RECORD_NOT_FOUND]
The request record is not found.

Usage

The **gsk_get_certificate_algorithms()** routine returns the public key algorithm, certificate signature algorithm, and signature key algorithm for the database record specified by the label parameter.

gsk_get_certificate_info()

Returns requested certificate information for an X.509 certificate.

Format

```
#include <gskcms.h>

gsk_status gsk_get_certificate_info(
                         gsk_buffer *         cert_stream,
                         x509_cert_info_id    cert_info_id,
                         gsk_buffer *         cert_info)
```

Parameters

cert_stream
Specifies either a DER-encoded X.509 certificate or a non-decoded TBSCertificate ASN.1 sequence.

cert_info_id
The X.509 certificate information identifier specifying the certificate information to be returned.

cert_info
Returns the requested certificate information. The application should call the **gsk_free_buffer()** routine to release the certificate information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_NO_MEMORY]
Insufficient storage is available.

[ASN_ELEMENTS_MISSING]
Required data element is missing.

[ASN_UNSUPPORTED_VERSION]
Version is not supported.

[CMSERR_BAD_ISSUER_NAME]
Issuer name is not valid.

[CMSERR_BAD_SUBJECT_NAME]
Subject name is not valid.

[CMSERR_ATTRIBUTE_INVALID_ENUMERATION]
The enumeration value is not valid.

Usage

The **gsk_get_certificate_info()** routine returns information about an X.509 certificate. The certificate stream may be either:

- An X.509 certificate encoded as described in RFC 5280 (tools.ietf.org/html/rfc5280).
- The *derCertificate* field of the x509_certificate structure, which contains the non-decoded TBSCertificate ASN.1 sequence.

The application may request certificate information by using one of the following enumeration identifiers.

x509_cert_info_subject_dn_der
The subject distinguished name for the X.509 certificate in binary ASN.1 DER-encoded format.

x509_cert_info_issuer_dn_der
The issuer distinguished name for the X.509 certificate in binary ASN.1 DER-encoded format.

gsk_get_cms_vector()

Obtains the address of the Certificate Management Services function vector.

Format

```
#include <gskcms.h>

void gsk_get_cms_vector (
                         gsk_uint32 *              function_mask,
                         gsk_cms_vector **         function_vector)
```

Parameters

function_mask
Returns a bit mask indicating the Certificate Management Services level.

function_vector
Returns the address of the Certificate Management Services function vector.

Usage

Certificate Management Services (CMS) functions can be called using either static binding or runtime binding. Static binding is performed when the application is compiled while runtime binding is performed when the application is run.

In order to use static binding, the CMS sidefile is specified as input to the binder. This causes all CMS functions to be resolved at bind time and will cause the CMS DLL to be implicitly loaded when the application is run.

In order to use runtime binding, the CMS DLL must be explicitly loaded by the application and the CMS functions must be called using indirect addresses. The **gsk_get_cms_vector()** routine allows an application to obtain the address of the CMS function vector containing an entry for each CMS API routine. This eliminates the need for the application to build the function vector through repeated calls to the **dllqueryfn()** routine.

The function mask indicates the capabilities of the version of the CMS DLL. These values have been defined:

GSKCMS_API_LVL1
CMS functions provided as part of z/OS Version 1 Release 4 are available.

GSKCMS_API_LVL2
CMS functions provided as part of z/OS Version 1 Release 6 are available.

GSKCMS_API_LVL3
CMS functions provided as part of z/OS Version 1 Release 8 are available.

GSKCMS_API_LVL4
CMS functions provided as part of z/OS Version 1 Release 9 are available.

GSKCMS_API_LVL5
CMS functions provided as part of z/OS Version 1 Release 10 are available.

GSKCMS_API_LVL6
CMS functions provided as part of z/OS Version 1 Release 11 are available.

GSKCMS_API_LVL7
CMS functions provided as part of z/OS Version 1 Release 12 are available.

GSKCMS_API_LVL8
CMS functions provided as part of z/OS Version 1 Release 13 are available.

GSKCMS_API_LVL9
CMS functions provided as part of z/OS Version 2 Release 1 are available.

GSKCMS_API_LVL10
CMS functions provided as part of z/OS Version 2 Release 2 are available.

GSKCMS_API_LVL11
CMS functions provided as part of z/OS Version 2 Release 3 are available.

GSKCMS_API_LVL12
CMS functions provided as part of z/OS Version 2 Release 4 are available.

GSKCMS_API_LVL13
CMS functions provided as part of z/OS Version 2 Release 5 are available.

gsk_get_content_type_and_cms_version()

Extracts the PKCS #7 *content_info_type*, *content_info_oid*, and *cms_version* from the pkcs_content_info structure.

Format

```
#include <gskcms.h>

gsk_status gsk_get_content_type_and_cms_version (
                                    gsk_buffer *            content_info,
                                    pkcs_content_type *     content_info_type,
                                    gsk_oid *               content_info_oid,
                                    int *                   cms_version)
```

Parameters

content_info
Specifies the encoded content_info.

content_info_type
Returns the pkcs_content_type enumeration value.

content_info_oid
Returns the pkcs_content_type OID. The application should call the **gsk_free_oid()** routine to release the OID elements when they are no longer needed.

cms_version
Returns the PKCS #7 CMSVersion as defined in RFC 3852.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_get_content_type_and_cms_version()** routine extracts the type, object id data (OID) and the CMSVersion of the content data from a PKCS #7 (Cryptographic Message Syntax) ASN.1 DER-encoded content information sequence.

Returned field *content_info_oid* contains the OID extracted from the ContentInfo sequence. The returned *cms_version* is from the ContentInfo data type.

ContentInfo data type *pkcs_content_data* does not have a *cms_version* as defined by RFC 3852. For this type, -1 is returned as the *cms_version*.

gsk_get_default_key()

Gets the default key record.

Format

```
#include <gskcms.h>

gsk_status gsk_get_default_key (
                                gsk_handle              db_handle,
                                gskdb_record **         record)
```

Parameters

db_handle
: Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine.

record
: Returns the database record. The application should call the **gsk_free_record()** routine to release the record when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
: Triple DES key parts are not unique.

[CMSERR_BAD_HANDLE]
: The database handle is not valid.

[CMSERR_INCORRECT_DBTYPE]
: The database does not support this operation.

[CMSERR_MULTIPLE_DEFAULT]
: Multiple keys are marked as the default.

[CMSERR_NO_MEMORY]
: Insufficient storage is available.

[CMSERR_RECORD_DELETED]
: The requested record is deleted.

[CMSERR_RECORD_NOT_FOUND]
: There is no default key for the database.

Usage

The **gsk_get_default_key()** routine retrieves the record for the default key. An error will be returned if there is no default key.

gsk_get_default_label()

Gets the label of the default key record.

Format

```
#include <gskcms.h>

gsk_status gsk_get_default_label (
                                   gsk_handle         db_handle,
                                   char **            label)
```

Parameters

db_handle
: Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine.

label
: Returns the label of the default key record. The application should call the **gsk_free_string()** routine to release the label when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_HANDLE]
: The database handle is not valid.

[CMSERR_INCORRECT_DBTYPE]
: The database does not support this operation.

[CMSERR_MULTIPLE_DEFAULT]
: Multiple keys are marked as the default.

[CMSERR_NO_MEMORY]
: Insufficient storage is available.

[CMSERR_RECORD_DELETED]
: The requested record is deleted.

[CMSERR_RECORD_NOT_FOUND]
: There is no default key for the database.

Usage

The **gsk_get_default_label()** routine returns the label of the default key record. An error will be returned if there is no default key.

gsk_get_directory_certificates()

Gets the certificates stored in the LDAP directory for the subject.

Format

```
#include <gskcms.h>

gsk_status gsk_get_directory_certificates (
                                  gsk_handle             directory_handle,
                                  x509_name *            subject_name,
                                  gsk_boolean            ca_certificates,
                                  pkcs_certificates *    certificates)
```

Parameters

directory_handle
Specifies the directory handle returned by the **gsk_open_directory()** or the **gsk_create_revocation_source()** routine.

subject_name
Specifies the certificate subject.

ca_certificates
Specify TRUE if the subject is a certification authority or FALSE if the subject is an end entity.

certificates
Returns the certificates for the subject. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_HANDLE]
The directory handle is not valid.

[CMSERR_LDAP]
An error is detected by the LDAP runtime support.

[CMSERR_LDAP_NOT_AVAILABLE]
The LDAP server is not available.

[CMSERR_LDAP_RESPONSE_TIMEOUT]
LDAP response not received within configured time limit.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_NOT_FOUND]
The requested certificate is not found.

Usage

The **gsk_get_directory_certificates()** routine retrieves the certificates that are stored in the LDAP directory for the specified subject name. When matching UTF-8 encoded attribute values in the subject name, System SSL uses a case sensitive (exact match) comparison. The directory schema is defined by RFC 2587 (tools.ietf.org/html/rfc2587). The certificates are stored as attributes of the subject directory entry. Each certificate is encoded as defined by RFC 5280 (tools.ietf.org/html/rfc5280). The *userCertificate* attribute is used to retrieve end-entity certificates while the *caCertificate* attribute is used to retrieve certification authority certificates.

Retrieved certificates are cached so that it is not necessary to contact the LDAP server for subsequent requests for the same certificates. The cached certificates are released when the **gsk_close_directory()** or the **gsk_free_revocation_source()** routine is called to close the directory handle.

gsk_get_directory_crls()

Gets the certificate revocation lists stored in the LDAP directory for the issuer.

Format

```
#include <gskcms.h>

gsk_status gsk_get_directory_crls (
                                    gsk_handle          directory_handle,
                                    x509_name *         dist_point_name,
                                    x509_name *         issuer_name,
                                    gsk_boolean         ca_lists,
                                    x509_crls *         crls)
```

Parameters

directory_handle
Specifies the directory handle returned by the **gsk_open_directory()** or the **gsk_create_revocation_source()** routine.

dist_point_name
Specifies the CRL distribution point name.

issuer_name
Specifies the CRL issuer name.

ca_lists
Specify TRUE to retrieve the revocation lists for CA certificates or FALSE to retrieve the revocation list for end entity certificates.

crls
Returns the certificate revocation lists. The application should call the **gsk_free_crls()** routine to release the lists when they are no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_HANDLE]
The directory handle is not valid.

[CMSERR_LDAP]
An error is detected by the LDAP runtime support.

[CMSERR_LDAP_NOT_AVAILABLE]
The LDAP server is not available.

[CMSERR_LDAP_RESPONSE_TIMEOUT]
LDAP response not received within configured time limit.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_NOT_FOUND]
The requested CRL is not found.

Usage

The **gsk_get_directory_crls()** routine retrieves the certificate revocation lists (CRLs) stored in the LDAP directory for the specified issuer name. When matching UTF-8 encoded attribute values (gsk_string_utf8) in the issuer name, System SSL uses a case sensitive (exact match) comparison. The directory schema is defined by RFC 2587 (tools.ietf.org/html/rfc2587). The revocation lists are stored as attributes of the issuer directory entry. Each CRL is encoded as defined by RFC 5280 (tools.ietf.org/html/rfc5280). The *certificateRevocationList* attribute is used to retrieve revocation lists for end-entity certificates while the *authorityRevocationList* attribute is used to retrieve revocation lists for certification authority certificates.

The *dist_point_name* parameter specifies the CRL distribution point name. This name is used as the distinguished name for the LDAP directory entry. The *issuer_name* parameter specifies the CRL issuer name. This name must match the issuer name stored in the CRL.

If the directory handle was created by the **gsk_open_directory()** routine with a non-zero value for the *crl_cache_timeout* parameter, retrieved CRLs are cached so that it is not necessary to contact the LDAP server for subsequent requests for the same issuer. The cached certificate revocation lists will be discarded at the end of the cache timeout specified on the **gsk_open_directory()** routine. The cached revocation lists will also be released when the **gsk_close_directory()** routine is called to close the directory handle. By default, the size of this cache is unlimited, but the size can be changed by invoking the **gsk_set_directory_numeric_value()** after the **gsk_open_directory()** routine. If the cache size is set, the maximum number of CRLs are already stored in the cache and the cache is not due to be flushed based on the cache timeout value, the CRL that was added earliest to the cache is removed. By default, the maximum size in bytes of a CRL that is allowed to be stored in the cache is unlimited or 0, but this size can be changed by calling the **gsk_set_directory_numeric_value()**. If the LDAP server does not contain an CRL for the specified CRL distribution point name and caching is active, by default, a temporary CRL will be cached to alleviate repeated calls to the LDAP server. If this behavior is not desired, it can be changed by invoking the **gsk_set_directory_enum_value()** routine and disabling the addition of these temporary CRLs to the cache. The cached revocation lists will be released when the **gsk_close_directory()** routine is called to close the directory handle.

If the directory handle was created by the **gsk_create_revocation_source()** routine with a non-zero value for the *crlCacheSize* parameter in the gskdb_extended_directory_source structure, retrieved CRLs that contain an expiration time that is later than the current time are cached so that it is not necessary to contact the LDAP server for subsequent requests for the same issuer. The cached certificate revocation list resides in the cache as long as the expiration time is not exceeded. Once the expiration time passes, the CRL is removed from the cache when referenced or when the cache size is about to be exceeded when adding new CRLs to the cache. If the *crlCacheSize* parameter is set to a non-zero value, the cache is already full, and the cache does not contain any expired CRLs, the CRL that is closest to expiration is removed from the cache. If the *crlCacheEntryMaxSize* parameter in the gskdb_extended_directory_source structure is set to 0, there is no limit on the size of an CRL that can be stored in the cache. However, if *crlCacheEntryMaxSize* is set to a value greater than 0, then that is the largest CRL in bytes that is allowed to be stored in the cache. If the LDAP server does not contain an CRL for the specified CRL distribution point name and caching is active, a temporary CRL will only be cached if the *crlCacheEmptyCRL* parameter in the gskdb_extended_directory_source structure is set to TRUE. Caching of a temporary CRL prevents repeated calls to the LDAP server when the CRL does not exist in the directory. The *crlCacheTempCRLTimeout* value in the gskdb_extended_directory_source structure indicates the time in hours that the temporary CRL should reside in the cache. When the **gsk_free_revocation_source()** routine is called to close the extended directory handle, all cached certificate revocation lists and temporary CRLs will be released.

gsk_get_directory_enum()

Gets an enumerated value from an LDAP directory.

Format

```
#include <gskcms.h>

gsk_status gsk_get_directory_enum (
                                    gsk_handle                          directory_handle,
                                    GSKCMS_DIRECTORY_ENUM_ID            enum_id,
                                    GSKCMS_DIRECTORY_ENUM_VALUE *       enum_value)
```

Parameters

directory_handle
Specifies an LDAP directory handle returned by **gsk_open_directory()**.

enum_id
Specifies the directory enumeration identifier.

enum_value
Specifies the directory enumeration value.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ATTRIBUTE_INVALID_ID]
The enumeration identifier is not valid or cannot be used with the specified handle.

[CMSERR_ATTRIBUTE_INVALID_ENUMERATION]
The enumeration value is not valid or cannot be used with the specified enumeration ID.

[CMSERR_BAD_HANDLE]
The handle is not valid.

Usage

The **gsk_get_directory_enum()** routine returns an enumerated value for an LDAP directory.

These enumeration identifiers are supported:

GSKCMS_CRL_CACHE_TEMP_CRL
Returns whether an temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not exist on the LDAP server.

One of two possible settings for GSKCMS_CRL_CACHE_TEMP_CRL will be returned:

GSKCMS_CRL_CACHE_TEMP_CRL_ON
A temporary CRL entry is added.

GSKCMS_CRL_CACHE_TEMP_CRL_OFF
A temporary CRL entry is not added.

GSKCMS_CRL_SECURITY_LEVEL
Returns the level of security set for the LDAP directory when contact is attempted between the application and an LDAP server that may contain a Certificate Revocation List (CRL).

One of three possible settings for GSKCMS_CRL_SECURITY_LEVEL will be returned:

GSKCMS_CRL_SECURITY_LEVEL_LOW
Certificate validation will not fail if the LDAP server cannot be contacted.

GSKCMS_CRL_SECURITY_LEVEL_MEDIUM
Certificate validation requires the LDAP server to be contactable, but does not require a CRL to be defined. This is the default setting.

GSKCMS_CRL_SECURITY_LEVEL_HIGH
Certificate validation requires the LDAP server to be contactable and a CRL to be defined.

gsk_get_directory_numeric_value()

Gets an integer value from an LDAP directory.

Format

```
#include <gskcms.h>

gsk_status gsk_get_directory_numeric_value (
                                gsk_handle                        directory_handle,
                                GSKCMS_DIRECTORY_NUM_ID           num_id,
                                int *                             num_value)
```

Parameters

directory_handle
Specifies an LDAP directory handle returned by **gsk_open_directory()**.

num_id
Specifies the directory numeric identifier.

num_value
Specifies the directory integer value.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ATTRIBUTE_INVALID_ID]
The numeric identifier is not valid or cannot be used with the specified handle.

[CMSERR_BAD_HANDLE]
The handle is not valid.

[CMSERR_INVALID_NUMERIC_VALUE]
The numeric value is not valid.

Usage

The **gsk_get_directory_numeric_value()** routine returns an integer value for an LDAP directory.

These numeric identifiers are supported:

GSKCMS_CRL_CACHE_ENTRY_MAXSIZE
Returns the maximum size in bytes of a CRL that is allowed to be stored in the LDAP CRL cache.

GSKCMS_CRL_CACHE_SIZE
Returns the maximum number of CRLs that are allowed to be stored in the LDAP CRL cache.

GSKCMS_LDAP_RESPONSE_TIMEOUT
Returns the time in seconds to wait for a response from the LDAP server.

gsk_get_ec_parameters_info()

Get the named curve type and key size for EC domain parameters.

Format

```
#include <gskcms.h>

gsk_status gsk_get_ec_parameters_info (
                                       gsk_buffer *                ec_parameters,
                                       x509_ec_parameters_info *   key_info)
```

Parameters

ec_parameters
Specifies the ASN.1-encoded EC domain parameters to be analyzed.

key_info
Returns the elliptic curve information.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CSMERR_EC_PARAMETERS_NOT_SUPPLIED]
EC parameters not supplied.

[CMSERR_STRUCTURE_TOO_SMALL]
Size specified for supplied structure is too small.

Usage

The **gsk_get_ec_parameters_info()** routine returns the elliptic curve type and key size of the supplied EC domain parameters. Before calling the function, the application must initialize the size field in *key_info* to the size of the x509_ec_parameters_info structure.

gsk_get_record_by_id()

Gets a database record using the record identifier.

Format

```
#include <gskcms.h>

gsk_status gsk_get_record_by_id (
                                   gsk_handle              db_handle,
                                   gsk_int32               record_id,
                                   gskdb_record **         record)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine.

record_id
Specifies the record identifier.

record
Returns the database record. The application should call the **gsk_free_record()** routine to release the record when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_NOT_FOUND]
The requested record is not found.

Usage

The **gsk_get_record_by_id()** routine retrieves a record from a key or request database based upon the unique record identifier. The record identifier is assigned when the record is added to the database and does not change as records are added and deleted.

gsk_get_record_by_index()

Gets a database record using a sequential index.

Format

```
#include <gskcms.h>

gsk_status gsk_get_record_by_index (
                                    gsk_handle              db_handle,
                                    int                     index,
                                    gskdb_record **         record)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine.

index
Specifies the sequential index of the record. The first record in the database is record 1.

record
Returns the database record. The application should call the **gsk_free_record()** routine to release the record when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_NOT_FOUND]
The requested record is not found.

Usage

The **gsk_get_record_by_index()** routine retrieves a record from a key or request database based upon a sequential index number. The first record in the database is record 1. The index numbers will change as records are added and deleted.

gsk_get_record_by_label()

Gets a database record using the record label.

Format

```
#include <gskcms.h>

gsk_status gsk_get_record_by_label (
                                  gsk_handle             db_handle,
                                  const char *           label,
                                  gskdb_record **        record)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine.

label
Specifies the label of the database record. The label is specified in the local code page.

record
Returns the database record. The application should call the **gsk_free_record()** routine to release the record when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_LABEL]
No label specified.

[CMSERR_MULTIPLE_LABEL]
Multiple certificates exist for label.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_NOT_FOUND]
The requested record is not found.

Usage

The **gsk_get_record_by_label()** routine retrieves a record from a key or request database based upon the record label. The record label is a character string assigned when the record is added to the database. The label comparison is case-sensitive.

gsk_get_record_by_subject()

Gets one or more database records using the certificate subject.

Format

```
#include <gskcms.h>

gsk_status gsk_get_record_by_subject (
                                       gsk_handle                db_handle,
                                       x509_name *               name,
                                       int *                     num_records,
                                       gskdb_record ***          records)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine.

name
Specifies the certificate subject.

num_records
Returns the number of records in the array.

records
Returns the array of database records. The application should call the **gsk_free_records()** routine to release the array when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_INCORRECT_DBTYPE]
The database does not support this operation.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECORD_NOT_FOUND]
The requested record is not found.

Usage

The **gsk_get_record_by_subject()** routine retrieves all records from a key database with the specified subject name. When matching UTF-8 encoded attribute values (gsk_string_utf8) in the subject name, System SSL uses a case sensitive (exact match) comparison.

gsk_get_record_labels()

Gets the record labels for a key or request database.

Format

```
#include <gskcms.h>

gsk_status gsk_get_record_labels (
                                   gsk_handle          db_handle,
                                   gsk_boolean         private_key,
                                   int *               num_labels,
                                   char ***            labels)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine.

private_key
Specify TRUE if labels for records containing a private key are to be returned. Specify FALSE if labels for records without a private key are to be returned.

num_labels
Returns the number of record labels.

labels
Returns an array of string addresses. The labels are returned using the local code page. The application should call the **gsk_free_strings()** routine to release the record labels when they are no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_get_record_labels()** routine returns all of the record labels for a key or request database. The **gsk_get_record_by_label()** routine can then be used to retrieve a specific database record. The array address will be set to NULL and the number of labels will be set to 0 if there are no records in the database.

gsk_get_update_code()

Gets the database update code.

Format

```
#include <gskcms.h>

gsk_status gsk_get_update_code (
                                gsk_handle            db_handle,
                                gsk_uint32 *          update_code)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine.

update_code
Returns the current update code for the database.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_get_update_code()** routine returns the current update code for the database. For a file-based database or z/OS PKCS #11 token, this is the modification timestamp. For a SAF key ring, this is the ring sequence number. If an update has occurred, the application can close and then re-open the database to pick up the updates.

gsk_import_certificate()

Imports a certificate.

Format

```
   #include <gskcms.h>
        gsk_status gsk_import_certificate (
                                          gsk_handle          db_handle,
                                          const char *        label,
                                          gsk_buffer *        stream)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine.

label
Specifies the label for the new database record. The label is specified in the local code page.

stream
Specifies the byte stream of the encoded certificate.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The key algorithm or signature algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
The algorithm key size is not valid.

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_BASE64_ENCODING]
The Base64 encoding of the import file is not correct.

[CMSERR_BAD_ENCODING]
The import file format is not recognized.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_BAD_SIGNATURE]
The certificate signature is not correct.

[CMSERR_DUPLICATE_CERTIFICATE]
The database already contains the certificate.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_EXPIRED]
The certificate is expired.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The issuer certificate does not allow signing certificates.

[CMSERR_ISSUER_NOT_CA]
The certificate issuer is not a certification authority.

[CMSERR_ISSUER_NOT_FOUND]
The issuer certificate is not in the key database.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NOT_YET_VALID]
The certificate is not yet valid.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UNSUPPORTED_EXPIRATION]
The expiration date exceeds February 6, 2106, at 23:59:59 UTC.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_import_certificate()** routine imports an X.509 certificate and creates a new database record. An error will be returned if the certificate is already in the database. The database must be a key database and must be open for update in order to import certificates.

The supplied stream can represent either the ASN.1 DER encoding for the certificate or the Cryptographic Message Syntax (PKCS #7) encoding for the certificate. This can be either the binary value or the Base64 encoding of the binary value. A Base64 encoded stream must be in the local code page and must include the encoding header and footer lines.

The **gsk_import_certificate()** routine imports a single certificate. If the PKCS #7 message contains multiple certificates, only the first certificate and its certificate chain will be imported. The certificate subject name will be used as the label for certificates added from the certification chain. A chain certificate will not be added to the database if the label is not unique or if the certificate is already in the database.

A unique record identifier is assigned when the record is added to the database. The certificate signature will be verified using the certificate of the issuer. An error will be returned if the issuer certificate is not already in the key database and is not contained in the PKCS #7 message stream. The certificate will be marked as a trusted certificate when it is added to the database.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string.

An existing certificate can be replaced by specifying the label of the existing certificate. The issuer name, subject name, and subject public key in the new certificate must be the same as the existing certificate. If the existing certificate has a private key, the private key is not changed when the certificate is replaced.

The database file is updated as part of the **gsk_import_certificate()** processing. A temporary database file is created using the same name as the database file with ".new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

gsk_import_key()

Imports a certificate and associated private key.

Format

```
#include <gskcms.h>

gsk_status gsk_import_key (
                            gsk_handle            db_handle,
                            const char *          label,
                            const char *          password,
                            gsk_buffer *          stream)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine.

label
Specifies the label for the new database record. The label is specified in the local code page.

password
Specifies the password for the import file. The password is in the local code page and must consist of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string. The user will be prompted to enter the password if NULL is specified for this parameter.

stream
Specifies the byte stream for the encoded certificate and private key.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_ALG_NOT_SUPPORTED]
The key algorithm or signature algorithm is not supported.

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_BASE64_ENCODING]
The Base64 encoding of the import file is not correct.

[CMSERR_BAD_ENCODING]
The import file format is not recognized.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_BAD_SIGNATURE]
The certificate signature is not correct.

[CMSERR_DUPLICATE_CERTIFICATE]
The database already contains the certificate.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_EXPIRED]
The certificate is expired.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The issuer certificate does not allow signing certificates.

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_ISSUER_NOT_CA]
The certificate issuer is not a certification authority.

[CMSERR_ISSUER_NOT_FOUND]
The issuer certificate is not in the key database.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NOT_YET_VALID]
The certificate is not yet valid.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_UNSUPPORTED_EXPIRATION]
The expiration date exceeds February 6, 2106, at 23:59:59 UTC.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_import_key()** routine imports an X.509 certificate and its private key and creates a new database record. An error will be returned if the database already contains the certificate. The database must be open for update in order to import certificates.

The certificate and key must have been encoded according to the Personal Information Exchange Syntax (PKCS #12). If executing in FIPS mode, the only supported encryption is the x509_alg_pbeWithSha1And3DesCbc algorithm. The supplied stream can be the binary ASN.1 sequence or the Base64 encoding of the ASN.1 sequence. A Base64 encoded stream is assumed to be in the local code page and must include the encoding header and footer lines.

The record label is used as a friendly name for the database entry. It can be any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be an empty string. An error will be returned if the certificate already exists in the key database or the record label is not unique.

A unique record identifier is assigned when the record is added to the database. The certificate signature will be verified using the certificate of the issuer. The certificate will be marked as a trusted certificate when it is added to the database.

Each certificate in the certification chain will be imported if it is present in the import file. The certificate subject name will be used as the label for certificates added from the certification chain. A chain certificate will not be added to the database if the label is not unique or if the certificate is already in the database.

The database file is updated as part of the **gsk_import_key()** processing. A temporary database file is created using the same name as the database file with ".new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

gsk_make_content_msg()

Creates a PKCS #7 content information message.

Format

```
 #include <gskcms.h>

gsk_status gsk_make_content_msg (
                                    pkcs_content_info *         content_info,
                                    gsk_buffer *                stream)
```

Parameters

content_info
Specifies the content information for the message.

stream
Returns the ASN.1 DER-encoded stream. The application should call the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported

[CMSERR_NO_MEMORY]
Insufficient storage is available

Usage

The **gsk_make_content_msg()** routine creates a PKCS #7 (Cryptographic Message Syntax) message using the supplied content information and returns the ASN.1 DER-encoded ContentInfo sequence. The message content type can be any of the types defined by the PKCS #7 specification. The **gsk_read_content_msg()** routine can be used to extract the content information from the stream.

gsk_make_data_content()

Creates PKCS #7 Data content information from application data.

Format

```
#include <gskcms.h>

gsk_status gsk_make_data_content (
                                  gsk_buffer *          data,
                                  pkcs_content_info *   content_info)
```

Parameters

data
Specifies the application data.

content_info
Returns the Data content information. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_NO_CONTENT_DATA]
The application data length is zero

[CMSERR_NO_MEMORY]
Insufficient storage is available

Usage

The **gsk_make_data_content()** routine creates PKCS #7 (Cryptographic Message Syntax) Data content information. The **gsk_read_data_content()** routine can be used to extract the application data from the content information.

gsk_make_data_msg()

Creates a PKCS #7 Data message from application data.

Format

```
#include <gskcms.h>

gsk_status gsk_make_data_msg (
                              gsk_buffer *                  data,
                              gsk_buffer *                  stream)
```

Parameters

data
Specifies the application data.

stream
Returns the ASN.1 DER-encoded stream. The application should call the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_NO_CONTENT_DATA]
The application data length is zero

[CMSERR_NO_MEMORY]
Insufficient storage is available

Usage

The **gsk_make_data_msg()** routine creates a PKCS #7 (Cryptographic Message Syntax) Data message and returns the ASN.1 DER-encoded ContentInfo sequence. The message content type will be Data. The **gsk_read_data_msg()** routine can be used to extract the application data from the stream.

Calling the **gsk_make_data_msg()** routine is equivalent to calling the **gsk_make_data_content()** routine followed by the **gsk_make_content_msg()** routine.

gsk_make_encrypted_data_content()

Creates PKCS #7 EncryptedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_make_encrypted_data_content (
                                  int                      version,
                                  x509_algorithm_type      pbe_algorithm,
                                  const char *             password,
                                  int                      iterations,
                                  pkcs_content_info *      content_data,
                                  pkcs_content_info *      content_info)
```

Parameters

version
Specifies the PKCS #7 EncryptedData version number. This must be 0.

pbe_algorithm
Specifies the password-based encryption algorithm.

password
Specifies the encryption password as a null-terminated string in the local code page. The user will be prompted to enter the password if NULL is specified for this parameter.

iterations
Specifies the number of iterations used to derive the encryption key from the password. It is recommended that iterations be specified as 1024 or greater.

content_data
Specifies the EncryptedData content. This must be one of the content information types defined in PKCS #7.

content_info
Returns the EncryptedData content information. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_AVAILABLE]
Encryption algorithm is not available

[CMSERR_ALG_NOT_SUPPORTED]
Encryption algorithm is not supported

[CMSERR_API_NOT_SUPPORTED]
The API is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported

[CMSERR_NO_CONTENT_DATA]
The content data length is zero

[CMSERR_NO_MEMORY]
Insufficient storage is available

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid

Usage

The **gsk_make_encrypted_data_content()** routine creates PKCS #7 (Cryptographic Message Syntax) EncryptedData content information. The data content type must be one of the types defined by PKCS #7. The **gsk_read_encrypted_data_content()** routine can be used to extract the content data from the content information.

gsk_make_encrypted_data_content() is not supported when executing in FIPS mode and will return CMSERR_API_NOT_SUPPORTED.

The encryption key is derived from the password as described in RFC 2898 (tools.ietf.org/html/rfc2898) and *PKCS #12, Version 1.0: Personal Information Exchange*. The selected algorithm determines how the key is derived from the password.

These password-based encryption algorithms are supported. The strong encryption algorithms may not be available depending upon government export regulations.

- **x509_alg_pbeWithMd2AndDesCbc - 56-bit DES encryption with MD2 digest - {1.2.840.113549.1.5.1}**
- **x509_alg_pbeWithMd5AndDesCbc - 56-bit DES encryption with MD5 digest - {1.2.840.113549.1.5.3}**
- **x509_alg_pbeWithSha1AndDesCbc - 56-bit DES encryption with SHA-1 digest - {1.2.840.113549.1.5.10}**
- **x509_alg_pbeWithMd2AndRc2Cbc - 64-bit RC2 encryption with MD2 digest - {1.2.840.113549.1.5.4}**
- **x509_alg_pbeWithMd5AndRc2Cbc - 64-bit RC2 encryption with MD5 digest - {1.2.840.113549.1.5.6}**
- **x509_alg_pbeWithSha1AndRc2Cbc - 64-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.5.11}**
- **x509_alg_pbeWithSha1And40BitRc2Cbc - 40-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.6}**
- **x509_alg_pbeWithSha1And128BitRc2Cbc - 128-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.5}**
- **x509_alg_pbeWithSha1And40BitRc4 - 40-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.2}**
- **x509_alg_pbeWithSha1And128BitRc4 - 128-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.1}**
- **x509_alg_pbeWithSha1And3DesCbc - 168-bit 3DES encryption with SHA-1 digest - {1.2.840.113549.1.12.1.3}**

gsk_make_encrypted_data_msg()

Creates a PKCS #7 EncryptedData message from application data.

Format

```
#include <gskcms.h>

gsk_status gsk_make_encrypted_data_msg (
                               int                        version,
                               x509_algorithm_type        pbe_algorithm,
                               const char *               password,
                               int                        iterations,
                               gsk_buffer *               data,
                               gsk_buffer *               stream)
```

Parameters

version
Specifies the PKCS #7 EncryptedData version number. This must be 0.

pbe_algorithm
Specifies the password-based encryption algorithm.

password
Specifies the encryption password as a null-terminated string in the local code page. The user will be prompted to enter the password if NULL is specified for this parameter.

iterations
Specifies the number of iterations used to derive the encryption key from the password. It is recommended that iterations be specified as 1024 or greater.

data
Specifies the application data for the EncryptedData message.

stream
Returns the ASN.1 DER-encoded stream. The application should call the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_AVAILABLE]
Encryption algorithm is not available

[CMSERR_ALG_NOT_SUPPORTED]
Encryption algorithm is not supported

[CMSERR_API_NOT_SUPPORTED]
The API is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported

[CMSERR_NO_CONTENT_DATA]
The content data length is zero

[CMSERR_NO_MEMORY]
Insufficient storage is available

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid

Usage

The **gsk_make_encrypted_data_msg()** routine creates a PKCS #7 (Cryptographic Message Syntax) EncryptedData message and returns the ASN.1 DER-encoded ContentInfo sequence. The encrypted data content type will be Data. The **gsk_read_encrypted_data_msg()** routine can be used to extract the application data from the stream.

gsk_make_encrypted_data_msg() is not supported when executing in FIPS mode and will return CMSERR_API_NOT_SUPPORTED.

Calling the **gsk_make_encrypted_data_msg()** routine is equivalent to calling the gsk_make_data_content() routine, the **gsk_make_encrypted_data_content()** routine, and the **gsk_make_content_msg()** routine.

The encryption key is derived from the password as described in RFC 2898 (tools.ietf.org/html/rfc2898) and *PKCS #12, Version 1.0: Personal Information Exchange*. The selected algorithm determines how the key is derived from the password.

These password-based encryption algorithms are supported. The strong encryption algorithms may not be available depending upon government export regulations.

- **x509_alg_pbeWithMd2AndDesCbc - 56-bit DES encryption with MD2 digest - {1.2.840.113549.1.5.1}**
- **x509_alg_pbeWithMd5AndDesCbc - 56-bit DES encryption with MD5 digest - {1.2.840.113549.1.5.3}**
- **x509_alg_pbeWithSha1AndDesCbc - 56-bit DES encryption with SHA-1 digest - {1.2.840.113549.1.5.10}**
- **x509_alg_pbeWithMd2AndRc2Cbc - 64-bit RC2 encryption with MD2 digest - {1.2.840.113549.1.5.4}**
- **x509_alg_pbeWithMd5AndRc2Cbc - 64-bit RC2 encryption with MD5 digest - {1.2.840.113549.1.5.6}**
- **x509_alg_pbeWithSha1AndRc2Cbc - 64-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.5.11}**
- **x509_alg_pbeWithSha1And40BitRc2Cbc - 40-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.6}**
- **x509_alg_pbeWithSha1And128BitRc2Cbc - 128-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.5}**
- **x509_alg_pbeWithSha1And40BitRc4 - 40-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.2}**
- **x509_alg_pbeWithSha1And128BitRc4 - 128-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.1}**
- **x509_alg_pbeWithSha1And3DesCbc - 168-bit 3DES encryption with SHA-1 digest - {1.2.840.113549.1.12.1.3}**

gsk_make_enveloped_data_content()

Creates PKCS #7 EnvelopedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_make_enveloped_data_content (
               int                      version,
               pkcs_session_key *       session_key,
               pkcs_certificates *      recipient_certificates,
               pkcs_content_info *      content_data,
               pkcs_content_info *      content_info)
```

Parameters

version
Specifies the PKCS #7 EnvelopedData standard version:

- Specify 0 to create EnvelopedData content as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 1 to create EnvelopedData content as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 2 to create EnvelopedData content as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *RecipientIdentifier*.

session_key
Specifies the session encryption key as follows:

- The *encryptionType* field specifies the encryption algorithm.
- The *encryptionKey.length* field specifies the encryption key length in bytes.
- The *encryptionKey.data* field specifies the address of the encryption key. A new key will be generated and returned in this parameter if the key address is NULL. If a new key is generated, the application should call the **gsk_free_buffer()** routine to release the key when it is no longer needed. Note that the *encryptionType* and *encryptionKey.length* fields must be set by the application even when a new session key is to be generated.

recipient_certificates
Specifies the certificates for the message recipients. There must be at least one recipient.

content_data
Specifies the EnvelopedData content. This must be one of the content information types defined in PKCS #7.

content_info
Returns the EnvelopedData content information. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_ALG_NOT_AVAILABLE]
The encryption algorithm is not available

[CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported

[CMSERR_BAD_RNG_OUTPUT]
In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_INCORRECT_KEY_USAGE]
A recipient certificate does not allow key encipherment

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_KEY_MISMATCH]
A recipient public key does not support data encryption

[CMSERR_NO_CONTENT_DATA]
The content data length is zero

[CMSERR_NO_MEMORY]
Insufficient storage is available

[CMSERR_RECIPIENT_NOT_FOUND]
No recipient certificates provided

[CMSERR_REQUIRED_EXT_MISSING]
Required certificate extension is missing.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid

[CMSERR_WEAK_KEY]
Triple DES (3DES) key parts are not unique.

Usage

The **gsk_make_enveloped_data_content()** routine creates PKCS #7 (Cryptographic Message Syntax) EnvelopedData content information. The data content type must be one of the types defined by PKCS #7. The **gsk_read_enveloped_data_content()** routine can be used to extract the content data from the EnvelopedData content information. No validity checking is performed on the recipient certificates. It is assumed that the application has already validated the recipient certificates.

The session key is used to encrypt the message content. A new session key is generated and returned to the application if no key is provided. For each recipient, the session key is encrypted with the recipient's public key and stored in the EnvelopedData message. This means the public key algorithm must support data encryption. Currently, only RSA public keys support data encryption. In addition, the certificate key usage must allow key encipherment.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - Key lengths 5 and 16 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - Key lengths 5 and 16 - {1.2.840.113549.3.4}**
- **x509_alg_desCbcPad - 56-bit DES - Key length 8 - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - Key length 24 - {1.2.840.113549.3.7}**
- **x509_alg_aesCbc128 - 128-bit AES CBC - Key length 16 - {2.16.840.1.101.3.4.1.2}**
- **x509_alg_aesCbc256 - 256-bit AES CBC - Key length 32 - {2.16.840.1.101.3.4.1.42}**

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.

When executing in FIPS mode if a 3DES session key is supplied, the three key parts are checked for key uniqueness.

gsk_make_enveloped_data_content_extended()

Creates PKCS #7 EnvelopedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_make_enveloped_data_content_extended (
                                   gsk_process_option        option_flag,
                                   int                       version,
                                   pkcs_session_key *        session_key,
                                   pkcs_certificates *       recipient_certificates,
                                   pkcs_content_info *       content_data,
                                   pkcs_content_info *       content_info)
```

Parameters

option_flag
Specifies process options to customize process behavior:

- Enforce recipient certificate has key encipherment capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate keyEnchiperment.

version
Specifies the PKCS #7 EnvelopedData standard version:

- Specify 0 to create EnvelopedData content as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 1 to create EnvelopedData content as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 2 to create EnvelopedData content as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *RecipientIdentifier*.

session key
Specifies the session encryption key as follows:

- The *encryptionType* field specifies the encryption algorithm.
- The *encryptionKey.length* field specifies the encryption key length in bytes.
- The *encryptionKey.data* field specifies the address of the encryption key. A new key will be generated and returned in this parameter if the key address is NULL. If a new key is generated, the application should call the **gsk_free_buffer()** routine to release the key when it is no longer needed. Note that the *encryptionType* and *encryptionKey.length* fields must be set by the application even when a new session key is to be generated.

recipient_certificates
Specifies the certificates for the message recipients. There must be at least one recipient.

content_data
Specifies the EnvelopedData content. This must be one of the content information types defined in PKCS #7.

content_info
Returns the EnvelopedData content information. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_ALG_NOT_AVAILABLE]
The encryption algorithm is not available

[CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported

[CMSERR_BAD_RNG_OUTPUT]
In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_INCORRECT_KEY_USAGE]
A recipient certificate does not allow key encipherment

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_KEY_MISMATCH]
A recipient public key does not support data encryption

[CMSERR_NO_CONTENT_DATA]
The content data length is zero

[CMSERR_NO_MEMORY]
Insufficient storage is available

[CMSERR_RECIPIENT_NOT_FOUND]
No recipient certificates provided

[CMSERR_REQUIRED_EXT_MISSING]
Required certificate extension is missing.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid

[CMSERR_WEAK_KEY]
Triple DES (3DES) key parts are not unique.

Usage

The **gsk_make_enveloped_data_content_extended()** routine creates PKCS #7 (Cryptographic Message Syntax) EnvelopedData content information. Processing is equivalent to **gsk_make_enveloped_data_content()**, except that the recipient certificate key usage need not assert key encipherment. The data content type must be one of the types defined by PKCS #7. The **gsk_read_enveloped_data_content()** routine or the **gsk_read_enveloped_data_content_extended()** routine can be used to extract the content data from the EnvelopedData content information. No validity checking is performed on the recipient certificates. It is assumed that the application has already validated the recipient certificates.

The session key is used to encrypt the message content. A new session key is generated and returned to the application if no key is provided. For each recipient, the session key is encrypted with the recipient's public key and stored in the EnvelopedData message. This means the public key algorithm must support data encryption. Currently, only RSA public keys support data encryption. In addition, if *option_flag* specifies that key encipherment is to be enforced, then the certificate key usage must allow key encipherment.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - Key lengths 5 and 16 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - Key lengths 5 and 16 - {1.2.840.113549.3.4}**
- **x509_alg_desCbcPad - 56-bit DES - Key length 8 - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - Key length 24 - {1.2.840.113549.3.7}**
- **x509_alg_aesCbc128 - 128-bit AES CBC - Key length 16 - {2.16.840.1.101.3.4.1.2}**
- **x509_alg_aesCbc256 - 256-bit AES CBC - Key length 32 - {2.16.840.1.101.3.4.1.42}**

When executing in FIPS mode, encryption algorithms x509_alg_rc2CbcPad, x509_alg_rc4 and x509_alg_desCbcPad are not supported.

When executing in FIPS mode if a 3DES session key is supplied, the three key parts are checked for key uniqueness.

gsk_make_enveloped_data_msg()

Creates a PKCS #7 EnvelopedData message from application data.

Format

```
#include <gskcms.h>

gsk_status gsk_make_enveloped_data_msg (
                                  int                      version,
                                  pkcs_session_key *       session_key,
                                  pkcs_certificates *      recipient_certificates,
                                  gsk_buffer *             data,
                                  gsk_buffer *             stream)
```

Parameters

version
Specifies the PKCS #7 EnvelopedData standard version:

- Specify 0 to create an EnvelopedData message as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 1 to create an EnvelopedData message as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 2 to create an EnvelopedData message as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *RecipientIdentifier*.

session_key
Specifies the session encryption key as follows:

- The *encryptionType* field specifies the encryption algorithm.
- The *encryptionKey.length* field specifies the encryption key length in bytes.
- The *encryptionKey.data* field specifies the address of the encryption key. A new key will be generated and returned in this parameter if the key address is NULL. If a new key is generated, the application should call the **gsk_free_buffer()** routine to release the key when it is no longer needed. Note that the *encryptionType* and *encryptionKey.length* fields must be set by the application even when a new session key is to be generated.

recipient_certificates
Specifies the certificates for the message recipients. There must be at least one recipient.

data
Specifies the application data for the EnvelopedData message.

stream
Returns the ASN.1 DER-encoded stream. The application should call the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_ALG_NOT_AVAILABLE]
The encryption algorithm is not available.

[CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_INCORRECT_KEY_USAGE]
A recipient certificate does not allow key encipherment.

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_KEY_MISMATCH]
A recipient public key does not support data encryption.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECIPIENT_NOT_FOUND]
No recipient certificates provided.

[CMSERR_REQUIRED_EXT_MISSING]
Required certificate extension is missing.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid.

[CMSERR_WEAK_KEY]
Triple DES (3DES) key parts are not unique.

Usage

The **gsk_make_enveloped_data_msg()** routine creates a PKCS #7 (Cryptographic Message Syntax) EnvelopedData message and returns the ASN.1 DER-encoded ContentInfo sequence. The enveloped data content type will be Data. The **gsk_read_enveloped_data_msg()** routine can be used to extract the application data from the stream. No validity checking is performed on the recipient certificates. It is assumed that the application has already validated the recipient certificates.

Calling the **gsk_make_enveloped_data_msg()** routine is equivalent to calling the **gsk_make_data_content()** routine, the **gsk_make_enveloped_data_content()** routine, and the **gsk_make_content_msg()** routine.

The session key is used to encrypt the message content. A new session key is generated and returned to the application if no key is provided. For each recipient, the session key is encrypted with the recipient's public key and stored in the EnvelopedData message. This means the public key algorithm must support data encryption. Currently, only RSA public keys support data encryption. In addition, the certificate key usage must allow key encipherment.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - Key lengths 5 and 16 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - Key lengths 5 and 16 - {1.2.840.113549.3.4}**
- **x509_alg_desCbcPad - 56-bit DES - Key length 8 - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - Key length 24 - {1.2.840.113549.3.7}**
- **x509_alg_aesCbc128 - 128-bit AES CBC - Key length 16 - {2.16.840.1.101.3.4.1.2}**
- **x509_alg_aesCbc256 - 256-bit AES CBC - Key length 32 - {2.16.840.1.101.3.4.1.42}**

When executing in FIPS mode, encryption algorithms **x509_alg_rc2CbcPad, x509_alg_rc4** and **x509_alg_desCbcPad** are not supported.

When executing in FIPS mode if a 3DES session key is supplied, the three key parts are checked for key uniqueness.

gsk_make_enveloped_data_msg_extended()

Creates a PKCS #7 EnvelopedData message from application data.

Format

```
#include <gskcms.h>

gsk_status gsk_make_enveloped_data_msg_extended (
                                  gsk_process_option      option_flag,
                                  int                     version,
                                  pkcs_session_key *      session_key,
                                  pkcs_certificates *     recipient_certificates,
                                  gsk_buffer *            data,
                                  gsk_buffer *            stream)
```

Parameters

option_flag

Specifies process options to customize process behavior:

- Enforce recipient certificate has key encipherment capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate keyEncipherment.

version

Specifies the PKCS #7 EnvelopedData standard version:

- Specify 0 to create an EnvelopedData message as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 1 to create an EnvelopedData message as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *RecipientIdentifier*.
- Specify 2 to create an EnvelopedData message as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *RecipientIdentifier*.

session_key

Specifies the session encryption key as follows:

- The *encryptionType* field specifies the encryption algorithm.
- The *encryptionKey.length* field specifies the encryption key length in bytes.
- The *encryptionKey.data* field specifies the address of the encryption key. A new key will be generated and returned in this parameter if the key address is NULL. If a new key is generated, the application should call the **gsk_free_buffer()** routine to release the key when it is no longer needed. Note that the *encryptionType* and *encryptionKey.length* fields must be set by the application even when a new session key is to be generated.

recipient_certificates

Specifies the certificates for the message recipients. There must be at least one recipient.

data

Specifies the application data for the EnvelopedData message.

stream

Returns the ASN.1 DER-encoded stream. The application should call the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_ALG_NOT_AVAILABLE]
The encryption algorithm is not available.

[CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_INCORRECT_KEY_USAGE]
A recipient certificate does not allow key encipherment.

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_KEY_MISMATCH]
A recipient public key does not support data encryption.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RECIPIENT_NOT_FOUND]
No recipient certificates provided.

[CMSERR_REQUIRED_EXT_MISSING]
Required certificate extension is missing.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid.

[CMSERR_WEAK_KEY]
Triple DES (3DES) key parts are not unique.

Usage

The **gsk_make_enveloped_data_msg_extended()** routine creates a PKCS #7 (Cryptographic Message Syntax) EnvelopedData message and returns the ASN.1 DER-encoded ContentInfo sequence. Processing is equivalent to **gsk_make_enveloped_data_msg()**, except that the recipient certificate key usage need not assert key encipherment. The enveloped data content type will be Data. The **gsk_read_enveloped_data_msg()** routine or the **gsk_read_enveloped_data_msg_extended()** routine can be used to extract the application data from the stream. No validity checking is performed on the recipient certificates. It is assumed that the application has already validated the recipient certificates.

Calling the **gsk_make_enveloped_data_msg_extended()** routine is equivalent to calling the **gsk_make_data_content()** routine, the **gsk_make_enveloped_data_content_extended()** routine, and the **gsk_make_content_msg()** routine.

The session key is used to encrypt the message content. A new session key is generated and returned to the application if no key is provided. For each recipient, the session key is encrypted with the recipient's public key and stored in the EnvelopedData message. This means the public key algorithm must support data encryption. Currently, only RSA public keys support data encryption. In addition, if option_flag specifies that key encipherment is to be enforced, then the certificate key usage must allow key encipherment.

These encryption algorithms are supported. Strong encryption may not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - Key lengths 5 and 16 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - Key lengths 5 and 16 - {1.2.840.113549.3.4}**

- **x509_alg_desCbcPad - 56-bit DES - Key length 8 - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - Key length 24 - {1.2.840.113549.3.7}**
- **x509_alg_aesCbc128 - 128-bit AES CBC - Key length 16 - {2.16.840.1.101.3.4.1.2}**
- **x509_alg_aesCbc256 - 256-bit AES CBC - Key length 32 - {2.16.840.1.101.3.4.1.42}**

When executing in FIPS mode, encryption algorithms **x509_alg_rc2CbcPad, x509_alg_rc4** and **x509_alg_desCbcPad** are not supported.

When executing in FIPS mode if a 3DES session key is supplied, the three key parts are checked for key uniqueness.

gsk_make_enveloped_private_key_msg()

Creates a PKCS #7 EnvelopedData message from application data. The application data passed in is the PKCS #11 secure key label name.

Format

```
#include <gskcms.h>

gsk_status gsk_make_enveloped_private_key_msg (
                                gsk_uint32                  option_flag,
                                int                         version,
                                x509_algorithm_type         encryption_algorithm,
                                pkcs_certificates *         recipient_certificates,
                                gsk_buffer *                secure_key_label,
                                gsk_buffer *                stream)
```

Parameters

option_flag
Specifies process options to customize process behavior. Specify execution options using bit setting.

- GSK_PROCESS_OPTION_ENFORCE_KEYUSAGE - Enforce recipient certificate has key encipherment capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate keyEncipherment is supported.
- Any other bit values are ignored.

version
Specify PKCS #7 EnvelopedData version number. Only version 0, PKCS #7 Version 1.5, is supported.

encryption_algorithm
Specifies the algorithm to be used:

- **x509_alg_aesCbc128** for AES with Key length 16.
- **x509_alg_aesCbc256** for AES with Key length 32.
- **x509_alg_desEde3CbcPad** for 3DES with Key length 24.

recipient_certificates
Specifies the certificates for the message recipients. There must be at least one recipient.

secure_key_label
Specifies a PKCS #11 secure private key label object. No other type of object is supported.

stream
Returns the ASN.1 DER-encoded stream. The application calls the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_AVAILABLE]
The encryption algorithm is not available.

[CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
The recipient key size is not supported.

[CMSERR_CRYPTO_HARDWARE_NOT_AVAILABLE]
Cryptographic hardware does not support service or algorithm.

[CMSERR_ICSF_FIPS_BAD_ALG_OR_KEY_SIZE]
A recipient algorithm or key size is not FIPS approved for an ICSF operation.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_FIPS]
ICSF is not operating in FIPS mode.

[CMSERR_INCORRECT_KEY_ATTRIBUTE]
Key attributes do not support envelope operation.

[CMSERR_INCORRECT_KEY_USAGE]
A recipient certificate does not allow key encipherment.

[CMSERR_KEY_MISMATCH]
A recipient public key does not support data encryption.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PKCS11_OBJECT_NOT_FOUND]
A PKCS #11 key label is either missing or not valid.

[CMSERR_NO_PRIVATE_KEY]
No private key.

[CMSERR_PKCS11_LABEL_INVALID]
PKCS #11 label is not valid.

[CMSERR_RECIPIENT_NOT_FOUND]
No recipient certificates provided.

Usage

The **gsk_make_enveloped_private_key_msg()** routine creates a PKCS #7 (Cryptographic Message Syntax) EnvelopedData message using a TKDS secure private key label and returns the ASN.1 DER-encoded ContentInfo sequence. The **gsk_read_enveloped_data_content()** routine or the **gsk_read_enveloped_data_content_extended()** routine can be used to extract the content data from the EnvelopedData content information. No validity checking is performed on the recipient certificates. It is assumed that the application validated the recipient certificates.

A session key is used to encrypt the message content. A new session key is generated but is not returned to the application. For each recipient, the session key is encrypted with the public key of the recipient and stored in the EnvelopedData message. Each recipient's public key must be type RSA.

In addition, if *option_flag* specifies that key usage is to be enforced, then each recipient certificate key usage must allow key encipherment.

These encryption algorithms are supported. Strong encryption might not be available, depending upon government export regulations.

- **x509_alg_desEde3CbcPad - 168-bit 3DES - Key length 24 - {1.2.840.113549.3.7}**
- **x509_alg_aesCbc128 - 128-bit AES CBC - Key length 16 - {2.16.840.1.101.3.4.1.2}**
- **x509_alg_aesCbc256 - 256-bit AES CBC - Key length 32 - {2.16.840.1.101.3.4.1.42}**

gsk_make_signed_data_content()

Creates PKCS #7 SignedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_make_signed_data_content (
                                   int                        version,
                                   x509_algorithm_type        digest_algorithm,
                                   gsk_boolean                include_certificates,
                                   pkcs_cert_keys *           signer_certificates,
                                   pkcs_certificates *        ca_certificates,
                                   pkcs_content_info *        content_data,
                                   pkcs_content_info *        content_info)
```

Parameters

version
Specifies the PKCS #7 SignedData standard version:

- Specify 0 to create SignedData content as described in PKCS #7 Version 1.4. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 1 to create SignedData content as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 2 to create SignedData content as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 3 to create SignedData content as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *signerIdentifier*.

digest_algorithm
Specifies the digest algorithm.

include_certificates
Specify TRUE if the signer and certification authority certificates are to be included in the SignedData content information. Specify FALSE if the certificates are not to be included.

signer_certificates
Specifies the certificates and associated private keys for the message signers. There must be at least one signer.

ca_certificates
Specifies the certification authority certificates. Zero or more certification authority certificates can be included in the SignedData content information. This parameter is ignored if the include_certificates parameter is set to FALSE. NULL can be specified for this parameter if no CA certificates are to be included in the message.

content_data
Specifies the SignedData content. This must be one of the content information types defined in PKCS #7.

content_info
Returns the SignedData content information. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
Private key does not exist or is not accessible.

[CMSERR_REQUIRED_EXT_MISSING]
Required certificate extension is missing.

[CMSERR_SIGNER_NOT_FOUND]
No signer certificate provided or the certificate is not valid.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid

Usage

The **gsk_make_signed_data_content()** routine creates PKCS #7 (Cryptographic Message Syntax) SignedData content information. The data content type must be one of the types defined by PKCS #7. The **gsk_read_signed_data_content()** routine can be used to extract the content data from the SignedData content information. The key usage for the signer certificates must allow digital signature. No validity checking will be performed on the signer certificates. It is assumed that the application has already validated the signer certificates.

A signature is included for each signer provided by the *signer_certificates* parameter. The X.509 certificates used to sign the message will be included in the SignedData content information if the *include_certificates* parameter is set to TRUE. The message receiver will need to provide the signer certificates if the *include_certificates* parameter is set to FALSE.

You can optionally include certification authority certificates in the SignedData content information. These certificate can then be used by the message receiver to validate the signer certificates.

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.3}

When creating a PKCS #7 message utilizing an RSA signer certificate and an RSASSA-PSS signature, the following signature algorithms are supported. The digest defined within the algorithm is the digest used.

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation algorithm 1.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_make_signed_data_content_extended()

Creates PKCS #7 SignedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_make_signed_data_content_extended (
                                gsk_process_option        option_flag,
                                int                       version,
                                x509_algorithm_type       digest_algorithm,
                                gsk_boolean               include_certificates,
                                pkcs_cert_keys *          signer_certificates,
                                pkcs_certificates *       ca_certificates,
                                pkcs_content_info *       content_data,
                                gsk_attributes_signers *  attributes_signers,
                                pkcs_content_info *       content_info)
```

Parameters

option_flag

Specifies process options to customize process behavior.

- Enforce signing certificate has digital signing capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate digitalSignature.
- Do not allow zero-length content data.
- Create detached (external) signature content data. The passed in content data is included in the data being digitally signed, but is not included in the returned SignedData content. This flag is only supported when version 500, 501, 502, or 503 is specified. It is ignored when version 0, 1, 2, or 3 is specified.

version

Specifies the PKCS #7 SignedData standard version:

- Specify 0 to create SignedData content as described in PKCS #7 Version 1.4. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 1 to create SignedData content as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 2 to create SignedData content as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 3 to create SignedData content as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *signerIdentifier*.

The following versions have similar meanings to the above PKCS #7 SignedData standard versions, but when using these versions, the caller of this routine has ensured that all bits within the *option_flag* have been set to 0 except for those bits which need to be processed to build the appropriate signedData message.

- Specify 500 to create SignedData content as described in PKCS #7 Version 1.4 This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 501 to create SignedData content as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 502 to create SignedData content as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 503 to create Signed Data content as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *signerIdentifier*.

digest_algorithm
Specifies the digest algorithm.

include_certificates
Specify TRUE if the signer and certification authority certificates are to be included in the SignedData content information. Specify FALSE if the certificates are not to be included.

signer_certificates
Specifies the certificates and associated private keys for the message signers. There must be at least one signer.

ca_certificates
Specifies the certification authority certificates. Zero or more certification authority certificates can be included in the SignedData content information. This parameter is ignored if the include_certificates parameter is set to FALSE. NULL can be specified for this parameter if no CA certificates are to be included in the message.

content_data
Specifies the SignedData content. This must be one of the content information types defined in PKCS #7.

attributes_signers
Specifies the authenticated attributes per signer to be added to the message. Specify NULL for this parameter if there are no authenticated attributes to be included in the message. If specified, the set of authenticated attributes must NOT include content-type or message-digest authenticated attributes as these are automatically provided by **gsk_make_signed_data_content_extended()**. If the set of authenticated attributes includes signing-time, then this will override the signing-time attribute generated by **gsk_make_signed_data_content_extended()**. The *digestAlgorithm* field within each *gsk_attributes_signer* structure is ignored - the digest algorithm is specified by the *digest_algorithm* parameter. If version 3 is requested, authenticated attributes continue to utilize the signer certificate's *IssuerAndSerialNumber*.

content_info
Returns the SignedData content information. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
Encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_CONTENTTYPE_NOT_ALLOWED]
The content-type authenticated attribute is not allowed in *attributes_signers*.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_RSA_PRIVATE_KEY_BAD_TYPE]
PKDS RSA private key type not valid for RSASSA-PSS signature generation.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_MESSAGEDIGEST_NOT_ALLOWED]
The message-digest authenticated attribute is not allowed in *attributes_signers*.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
Private key does not exist or is not accessible.

[CMSERR_REQUIRED_EXT_MISSING]
Required certificate extension is missing.

[CMSERR_SIGNER_NOT_FOUND]
No signer certificate provided or the certificate is not valid.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid

[CMSERR_CONTENTTYPE_NOT_ALLOWED]
The content-type authenticated attribute is not allowed in *attributes_signers*.

[CMSERR_MESSAGEDIGEST_NOT_ALLOWED]
The message-digest authenticated attribute is not allowed in *attributes_signers*

Usage

The **gsk_make_signed_data_content_extended()** routine creates PKCS #7 (Cryptographic Message Syntax) SignedData content information. The data content type must be one of the types defined by PKCS #7. Processing is similar to **gsk_make_signed_data_content()** except for the presence of the *option_flag* and *authenticated_attributes* parameters. The **gsk_read_signed_data_content()** routine or the **gsk_read_signed_data_content_extended()** routine can be used to extract the content data from the SignedData content information, except for a detached (external) signature. Detached signature SignedData messages do not contain any content information and are not supported by these read routines. The key usage for the signer certificates can be optionally specified as to whether digital signature must be allowed. No validity checking is performed on the signer certificates. It is assumed that the application has already validated the signer certificates.

A signature is included for each signer provided by the *signer_certificates* parameter. The X.509 certificates used to sign the message will be included in the SignedData content information if the *include_certificates* parameter is set to TRUE. The message receiver will need to provide the signer certificates if the *include_certificates* parameter is set to FALSE.

You can optionally include certification authority certificates in the SignedData content information. These certificates can then be used by the message receiver to validate the signer certificates.

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.3}

When creating a PKCS #7 message utilizing an RSA signer certificate and an RSASSA-PSS signature, these algorithms are supported. The digest defined within the algorithm is the digest used.

x509_alg_mgf1Sha2564WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation algorithm 1.

If authenticated attributes are provided from the *attributes_signers* parameter, then signing certificates for all signers represented within the gsk_attributes_signers structure must be provided from the *signer_certificates* parameter.

If the RSA private key is in the PKDS, see “RSASSA-PSS signature support” on page 14 for information about required hardware support.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_make_signed_data_msg()

Creates a PKCS #7 SignedData message from application data.

Format

```
#include <gskcms.h>

gsk_status gsk_make_signed_data_msg (
                                      int                         version,
                                      x509_algorithm_type         digest_algorithm,
                                      gsk_boolean                 include_certificates,
                                      pkcs_cert_keys *            signer_certificates,
                                      pkcs_certificates *         ca_certificates,
                                      gsk_buffer *                data,
                                      gsk_buffer *                stream)
```

Parameters

version

Specifies the PKCS #7 SignedData standard version:

- Specify 0 to create SignedData content as described in PKCS #7 Version 1.4. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 1 to create SignedData content as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 2 to create SignedData content as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 3 to create SignedData content as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *signerIdentifier*.

digest_algorithm

Specifies the digest algorithm.

include_certificates

Specify TRUE if the signer and certification authority certificates are to be included in the SignedData message. Specify FALSE if the certificates are not to be included.

signer_certificates

Specifies the certificates and associated private keys for the message signers. There must be at least one signer.

ca_certificates

Specifies the certification authority certificates. Zero or more certification authority certificates can be included in the SignedData message. This parameter is ignored if the include_certificates parameter is set to FALSE. NULL can be specified for this parameter if no CA certificates are to be included in the message.

data

Specifies the application data for the SignedData message.

stream

Returns the ASN.1 DER-encoded stream. The application should call the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
Private key does not exist or is not accessible.

[CMSERR_REQUIRED_EXT_MISSING]
Required certificate extension is missing.

[CMSERR_SIGNER_NOT_FOUND]
No signer certificate provided or the certificate is not valid.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid.

Usage

The **gsk_make_signed_data_msg()** routine creates a PKCS #7 (Cryptographic Message Syntax) SignedData message and returns the ASN.1 DER-encoded ContentInfo sequence. The signed data content type will be Data. The **gsk_read_signed_data_msg()** routine can be used to extract the application data from the stream. The key usage for the signer certificates must allow digital signature. No validity checking will be performed on the signer certificates. It is assumed that the application has already validated the signer certificates.

Calling the **gsk_make_signed_data_msg(**) routine is equivalent to calling the **gsk_make_data_content()** routine, the **gsk_make_signed_data_content()** routine, and the **gsk_make_content_msg()** routine.

A signature is included for each signer provided by the *signer_certificates* parameter. The X.509 certificates used to sign the message will be included in the SignedData message if the *include_certificates* parameter is set to TRUE. The message receiver will need to provide the signer certificates if the *include_certificates* parameter is set to FALSE.

You can optionally include certification authority certificates in the SignedData message. These certificates can then be used by the message receiver to validate the signer certificates.

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.3}

When creating a PKCS #7 message utilizing an RSA signer certificate and an RSASSA-PSS signature, these algorithms are supported. The digest defined within the algorithm is the digest used.

x509_alg_mgf1Sha2564WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation algorithm 1.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_make_signed_data_msg_extended()

Creates a PKCS #7 SignedData message from application data.

Format

```
#include <gskcms.h>

gsk_status gsk_make_signed_data_msg_extended (
                                     gsk_process_option         option_flag,
                                     int                        version,
                                     x509_algorithm_type        digest_algorithm,
                                     gsk_boolean                include_certificates,
                                     pkcs_cert_keys *           signer_certificates,
                                     pkcs_certificates *        ca_certificates,
                                     gsk_buffer *               data,
                                     gsk_attributes_signers *   attributes_signers,
                                     gsk_buffer *               stream)
```

Parameters

option_flag

Specifies process options to customize process behavior.

- Enforce signing certificate has digital signing capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate digitalSignature.
- Do not allow zero-length content data.
- Create detached (external) signature content data. The passed in data is included in the data being digitally signed, but is not included in the returned SignedData content. This flag is only supported when version 500, 501, 502, or 503 is specified. It is ignored when version 0, 1, 2, or 3 is specified.

version

Specifies the PKCS #7 SignedData standard version:

- Specify 0 to create SignedData content as described in PKCS #7 Version 1.4. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 1 to create SignedData content as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 2 to create SignedData content as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 3 to create SignedData content as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *signerIdentifier*.

The following versions have similar meanings to the above PKCS #7 SignedData standard versions, but when using these versions, the caller of this routine has ensured that all bits within the *option_flag* have been set to 0 except for those bits which need to be processed to build the appropriate signedData message.

- Specify 500 to create SignedData content as described in PKCS #7 Version 1.4 This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 501 to create SignedData content as described in PKCS #7 Version 1.5. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 502 to create SignedData content as described in PKCS #7 Version 1.6. This version encodes the *IssuerAndSerialNumber* as the *signerIdentifier*.
- Specify 503 to create Signed Data content as described in PKCS #7 RFC 3852. This version encodes the *SubjectKeyIdentifier* as the *signerIdentifier*.

digest_algorithm
Specifies the digest algorithm.

include_certificates
Specify TRUE if the signer and certification authority certificates are to be included in the SignedData message. Specify FALSE if the certificates are not to be included.

signer_certificates
Specifies the certificates and associated private keys for the message signers. There must be at least one signer.

ca_certificates
Specifies the certification authority certificates. Zero or more certification authority certificates can be included in the SignedData message. This parameter is ignored if the include_certificates parameter is set to FALSE. NULL can be specified for this parameter if no CA certificates are to be included in the message.

data
Specifies the application data for the SignedData message.

attributes_signers
Specifies the authenticated attributes per signer to be added to the message. Specify NULL for this parameter if there are no authenticated attributes to be included in the message. If specified, then the set of authenticated attributes must NOT include content-type or message-digest authenticated attributes as these are automatically provided by **gsk_make_signed_data_msg_extended()**. If the set of authenticated attributes includes signing-time, then this will override the signing-time attribute generated by **gsk_make_signed_data_msg_extended()**. The *digest_algorithm* field within each *gsk_attributes_signer* structure is ignored - the digest algorithm is specified by the *digest_algorithm* parameter. If version 3 is requested, authenticated attributes continue to utilize the signer certificate's *IssuerAndSerialNumber*.

stream
Returns the ASN.1 DER-encoded stream. The application should call the **gsk_free_buffer()** routine to release the stream when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
Encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_CONTENTTYPE_NOT_ALLOWED]
The content-type authenticated attribute is not allowed in *attributes_signers*.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_RSA_PRIVATE_KEY_BAD_TYPE]
PKDS RSA private key type not valid for RSASSA-PSS signature generation.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_MESSAGEDIGEST_NOT_ALLOWED]
The message-digest authenticated attribute is not allowed in *attributes_signers*.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
Private key does not exist or is not accessible.

[CMSERR_REQUIRED_EXT_MISSING]
Required certificate extension is missing.

[CMSERR_SIGNER_NOT_FOUND]
No signer certificate provided or the certificate is not valid.

[CMSERR_VERSION_NOT_SUPPORTED]
The version is not valid.

[CMSERR_CONTENTTYPE_NOT_ALLOWED]
The content-type authenticated attribute is not allowed in *attributes_signers*.

[CMSERR_MESSAGEDIGEST_NOT_ALLOWED]
The message-digest authenticated attribute is not allowed in *attributes_signers*

Usage

The **gsk_make_signed_data_msg_extended()** routine creates a PKCS #7 (Cryptographic Message Syntax) SignedData message and returns the ASN.1 DER-encoded ContentInfo sequence. The signed data content type will be Data. The **gsk_read_signed_data_msg()** or the **gsk_read_signed_data_msg_extended()** routine can be used to extract the application data from the stream when included in the message. Detached signature SignedData messages do not contain the application data and are not supported by these read routines. The key usage for the signer certificates can be optionally specified as to whether digital signature must be allowed. No validity checking will be performed on the signer certificates. It is assumed that the application has already validated the signer certificates.

Calling the **gsk_make_signed_data_msg_extended()** routine is equivalent to calling the **gsk_make_data_content()** routine, the **gsk_make_signed_data_content_extended()** routine, and the **gsk_make_content_msg()** routine.

A signature is included for each signer provided by the *signer_certificates* parameter. The X.509 certificates used to sign the message will be included in the SignedData message if the *include_certificates* parameter is set to TRUE. The message receiver will need to provide the signer certificates if the *include_certificates* parameter is set to FALSE.

You can optionally include certification authority certificates in the SignedData message. These certificates can then be used by the message receiver to validate the signer certificates.

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.3}

When creating a PKCS #7 message utilizing an RSA signer certificate and an RSASSA-PSS signature, these algorithms are supported. The digest defined within the algorithm is the digest used.

x509_alg_mgf1Sha2564WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation algorithm 1.

If authenticated attributes are provided from the *attributes_signers* parameter, then signing certificates for all signers represented within the gsk_attributes_signers structure must be provided from the *signer_certificates* parameter.

If the RSA private key is in the PKDS, see "RSASSA-PSS signature support" on page 14 for information about required hardware support.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, "System SSL and FIPS 140-2," on page 19.

gsk_make_wrapped_content()

Format

```
#include <gskcms.h>

gsk_status gsk_make_wrapped_content (
                                     pkcs_content_info *            content_info,
                                     pkcs_content_info *            wrapped_content)
```

Parameters

content_info
Specifies the content information to be wrapped.

wrapped_content
Returns the wrapped content information. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_make_wrapped_content()** routine wraps the supplied content information in an ASN.1 sequence and returns a new content information containing the wrapped data. The type of the wrapped content information is the same as the type of the original content information. The **gsk_read_wrapped_content()** routine can be used to extract the original content information.

gsk_mktime()

Converts year/month/day time value to number of seconds since the POSIX epoch

Format

```
#include <gskcms.h>

gsk_time gsk_mktime (
                    gsk_timeval *           ts)
```

Parameters

ts

Specifies the time to be converted. The tm_year, tm_mon, tm_mday, tm_hour, tm_min, and tm_sec fields are used to generate the converted time.

Results

The return value is the number of seconds since January 1, 1970. Leap seconds are not included in the computation. If a date is specified that is before January 1, 1970, or after February 6, 2106, **gsk_mktime()** returns the maximum unsigned 32-bit value, 4294967295 (0xFFFFFFFF).

Usage

The **gsk_mktime()** routine converts the time specified in year/month/day format to the number of seconds since the POSIX epoch (January 1, 1970). The **gsk_mktime()** routine differs from the **mktime()** routine in that the time is UTC and is not adjusted for the local timezone or for daylight savings time.

The year value must be 1970 - 2106 and is the actual year minus 1900. Therefore:

- *tm_year* must be 70 - 206.
- *tm_mon* must be 0 - 11.
- *tm_day* must be 1 - 31.
- *tm_hour* must be 0 - 23.
- *tm_min* must be 0 - 59.
- *tm_sec* must be 0 - 59.

The maximum valid date is February 6, 2106, at 23:59:59 UTC.

gsk_modify_pkcs11_key_label()

Return a gsk_buffer that adds or removes the equals sign (=) from the first position of an input TKDS key token label.

Format

```
#include <gskcms.h>

gsk_status gsk_modify_pkcs11_key_label (
                                        gsk_buffer *            in_buffer,
                                        gsk_boolean             add_preface,
                                        gsk_buffer *            out_buffer)
```

Parameters

in_buffer
Specifies the gsk_buffer containing the TKDS key token label.

add_preface
Specify TRUE if you want an equal sign (=) prefaced at the beginning of the TKDS key token label. This shifts the original string to the right one position.

Specify FALSE if you want the equal sign (=) removed from the beginning of the TKDS key token label. This shifts the original string to the left one position.

out_buffer
Returns a new gsk_buffer with the TKDS key token label in its new form.

Results

The function return value is 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_PKCS11_LABEL_INVALID]
The input did not have a label.

If *add_preface* is FALSE and the calculated length of the output string minus the equal sign (=) is zero.

Usage

The **gsk_modify_pkcs11_key_label()** routine creates gsk_buffer of the TKDS key label either with or without the equal sign (=) in the first position of the string.

- A TKDS key label without the equals sign is always 44 characters long.
- A TKDS key label with an equal sign is always 45 characters long.
- The caller is responsible for freeing the storage that is allocated to create the returned *out_buffer*.
  - If the returned *out_buffer* does not become the *keyToken* field of structure pkcs_private_key_info, call **gsk_free_buffer()** to free *out_buffer*.
  - If the returned *out_buffer* is used as the *keyToken* field of structure pkcs_private_key_info, the returned *out_buffer* is freed when calling **gsk_free_private_key_info()**.
- If the supplied TKDS key token label already has the equal sign in the first position and *add_preface* is TRUE, a copy of the original string is returned.
- If the supplied TKDS key token label already does not have the equal sign in the first position and *add_preface* is FALSE, a copy of the original string is returned.

gsk_name_compare()

Compares two X.509 names.

Format

```
#include <gskcms.h>

gsk_boolean gsk_name_compare (
                              x509_name *             name1,
                              x509_name *             name2)
```

Parameters

name1
Specifies the first name to be compared.

name2
Specifies the second name to be compared.

Results

Returns TRUE if the two x.509 names are the same and FALSE if the two x.509 names are different.

Usage

The **gsk_name_compare()** routine compares two X.509 names and return TRUE if the names are the same and FALSE if they are not the same.

Two names are considered equal if they contain the same sequence of attribute types and attribute values. Attribute values are considered equal if they represent the same character string. If a relative distinguished name (RDN) contains multiple attributes, the attributes must be specified in ascending order based on their ASN.1 DER encoding. Strings are always stored using UTF-8 encoding.When matching UTF-8 encoded attribute values (x509_string_utf8) in the X.509 names, System SSL uses a case sensitive (exact match) comparison.

Printable strings (gsk_string_printable) are a special case. Multiple spaces are treated as a single space and the comparison is not case-sensitive. Case-sensitive comparisons are used for all other string types.

gsk_name_to_dn()

Converts an X.509 name to a DN string.

Format

```
#include <gskcms.h>

gsk_status gsk_name_to_dn (
                            x509_name *          name,
                            char **              dn)
```

Parameters

name
Specifies the X.509 name to be converted to a distinguished name string. The X.509 strings use UTF-8 encoding.

dn
Returns the distinguished name in the local code page. The application should call the **gsk_free_string()** routine to release the string when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_CANT_CONVERT]
The X.509 name is not a distinguished name.

[ASN_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_name_to_dn()** routine converts an X.509 name to a distinguished name (DN) string in accordance with RFC 2253 (tools.ietf.org/html/rfc2253). The DN string will consist of single-byte characters in the local code page. A double-byte character will be represented using the escaped UTF-8 encoding of the double-byte character in the UCS-2 or UCS-4 character set.

These DN attribute names are generated by the System SSL runtime. Unrecognized attribute types will be encoded using the numeric object identifier followed by the DER-encoded representation of the attribute value.

- C - Country
- CN - Common name
- DC - Domain component
- DNQUALIFIER - Distinguished name qualifier
- EMAIL - E-mail address
- GENERATIONQUALIFIER - Generation qualifier
- GIVENNAME - Given name
- INITIALS - Initials
- L - Locality
- MAIL - Mail RFC 822 style address
- NAME - Name

- O - Organization name
- OU - Organizational unit name
- PC - Postal code
- SERIALNUMBER - Serial number
- SN - Surname
- ST - State or province
- STREET - Street
- T - Title

gsk_open_database()

Opens a key or request database.

Format

```
#include <gskcms.h>

gsk_status gsk_open_database (
                              const char *              filename,
                              const char *              password,
                              gsk_boolean               update_mode,
                              gsk_handle *              db_handle,
                              gskdb_database_type *     db_type,
                              int *                     num_records)
```

Parameters

filename
Specifies the database file name in the local code page. The length of the fully-qualified filename cannot exceed 251.

password
Specifies the database password in the local code page. The user will be prompted to enter the password if NULL is specified for this parameter.

update_mode
Specifies the file access mode. Specify TRUE if the database will be updated and FALSE if the database will not be updated. The application must have write access to the file if TRUE is specified.

db_handle
Returns the database handle. The application should call the **gsk_close_database()** routine when it no longer needs access to the database.

db_type
Returns the database type.

num_records
Returns the number of records in the database.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_ACCESS_DENIED]
The file permissions do not allow access.

[CMSERR_BAD_FILENAME]
The database file name is not valid.

[CMSERR_BAD_RNG_OUTPUT]
In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_DB_CORRUPTED]
The database file is not valid.

[CMSERR_DB_FIPS_MODE_ONLY]
Key database can only be opened for update if running in FIPS mode.

[CMSERR_DB_LOCKED]
The database is open for update by another process.

[CMSERR_DB_NOT_FIPS]
Key database is not a FIPS mode database.

[CMSERR_FILE_NOT_FOUND]
The database file is not found.

[CMSERR_IO_CANCELED]
The user canceled the password prompt.

[CMSERR_IO_ERROR]
An input/output request failed.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_OPEN_FAILED]
Unable to open the database.

Usage

The **gsk_open_database()** routine will open a key or request database file for either read-only or read/write access. The database must already exist. The database integrity will be verified and the open will fail if the database has been incorrectly modified. Only one process at a time may open a database in update mode. The database may be accessed by multiple concurrent threads in the same process if the same database handle is used by all of the threads.

A FIPS database file may only be opened for update while executing in FIPS mode. A FIPS database may be opened read-only while executing in non-FIPS mode. A non-FIPS database file cannot be opened for read or update while executing in FIPS mode.

Opening a GSKIT CMS V4 key database using **gsk_open_database()** is not supported.

gsk_open_database_using_stash_file()

Opens a key or request database using a stash file for the database password.

Format

```
#include <gskcms.h>

gsk_status gsk_open_database_using_stash_file (
                                      const char *           database_filename,
                                      const char *           stash_filename,
                                      gsk_boolean            update_mode,
                                      gsk_handle *           db_handle,
                                      gskdb_database_type *  db_type,
                                      int *                  num_records)
```

Parameters

database_filename
Specifies the database file name in the local code page. The length of the fully-qualified filename cannot exceed 251.

stash_filename
Specifies the stash file name in the local code page. The length of the fully-qualified filename cannot exceed 251. The stash file name always has an extension of ".sth" and the supplied name will be changed if it does not have the correct extension.

update_mode
Specifies the file access mode. Specify TRUE if the database will be updated and FALSE if the database will not be updated. The application must have write access to the file if TRUE is specified.

db_handle
Returns the database handle. The application should call the **gsk_close_database()** routine when it no longer needs access to the database.

db_type
Returns the database type.

num_records
Returns the number of records in the database.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_ACCESS_DENIED]
The file permissions do not allow access.

[CMSERR_BAD_FILENAME]
The database file name is not valid.

[CMSERR_DB_CORRUPTED]
The database file is not valid.

[CMSERR_DB_FIPS_MODE_ONLY]
Key database can only be opened for update if running in FIPS mode.

[CMSERR_DB_LOCKED]
The database is open for update by another process.

[CMSERR_NOT_FIPS]
Key database is not a FIPS mode database.

[CMSERR_FILE_NOT_FOUND]
The database file is not found.

[CMSERR_IO_ERROR]
An input/output request failed.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_OPEN_FAILED]
Unable to open the database.

Usage

The **gsk_open_database_using_stash_file()** routine is the same as the **gsk_open_database()** routine except the database password is obtained from the password stash file instead of being specified as a call parameter. The key or request database can be opened for read-only access or for read/write access. The database must already exist. The database integrity will be verified and the open will fail if the database has been incorrectly modified. Only one process at a time may open a database in update mode. The database may be accessed by multiple concurrent threads in the same process if the same database handle is used by all of the threads.

A FIPS database file may only be opened for update while executing in FIPS mode. A FIPS database may be opened read-only while executing in non-FIPS mode. A non-FIPS database file cannot be opened for read or update while executing in FIPS mode.

Opening a GSKIT CMS V4 key database using **gsk_open_database_using_stash_file()** is not supported.

gsk_open_directory()

Opens an LDAP directory.

Format

```
#include <gskcms.h>

gsk_status gsk_open_directory (
                               const char *            server_name,
                               int                     server_port,
                               const char *            user_name,
                               const char *            user_password,
                               int                     crl_cache_timeout,
                               gsk_handle *            directory_handle)
```

Parameters

server_name
: Specifies one or more blank-separated LDAP server host names. Each host name can contain an optional port number separated from the host name by a colon.

server_port
: Specifies the port assigned to the LDAP server. The default port will be used if zero is specified.

user_name
: Specifies the distinguished name to be used when binding to the LDAP server. An unauthenticated bind will be done if NULL is specified for this parameter.

user_password
: Specifies the password to be used when binding to the LDAP server. NULL may be specified for this parameter when NULL is also specified for the *user_name* parameter.

crl_cache_timeout
: Specifies the CRL cache timeout interval in hours. Specify 0 to disable CRL caching.

directory_handle
: Returns the directory handle. The application should call the **gsk_close_directory()** routine when it no longer needs access to the LDAP directory.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_LDAP_NOT_AVAILABLE]
: LDAP server is not available.

[CMSERR_NO_MEMORY]
: Insufficient storage is available

Usage

The **gsk_open_directory()** routine will open an LDAP directory and return a directory handle.

gsk_open_keyring()

Opens a SAF digital certificate key ring or z/OS PKCS #11 token.

Format

```
#include <gskcms.h>

gsk_status gsk_open_keyring (
                             const char *          ring_name,
                             gsk_handle *          db_handle,
                             int *                 num_records)
```

Parameters

ring_name
: Specifies the SAF key ring or z/OS PKCS #11 token name in the local code page. When using a key ring owned by the current user, specify the ring name as "name". When using a key ring owned by another user, specify the ring name as "userid/name". The maximum user ID length is 8 and the maximum name length is 237. The z/OS PKCS #11 token name is specified as *TOKEN*/token-name. *TOKEN* indicates that the specified key ring is actually a token name.

db_handle
: Returns the database handle. The application should call the **gsk_close_database()** routine when it no longer needs access to the key ring.

num_records
: Returns the number of records in the key ring or token.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ACCESS_DENIED]
: The access permissions do not allow access.

[CMSERR_BAD_FILENAME]
: The key ring or token name is not valid.

[CMSERR_BAD_RNG_OUTPUT]
: In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_FILE_NOT_FOUND]
: The key ring or token does not exist

[CMSERR_ICSF_FIPS_DISABLED]
: ICSF PKCS #11 services are disabled.

[CMSERR_INTERNAL_ERROR]
: An internal processing error has occurred.

[CMSERR_IO_ERROR]
: An error occurred while listing the key ring or token.

[CMSERR_NO_MEMORY]
: Insufficient storage is available.

Usage

The **gsk_open_keyring()** routine will open a key ring maintained by the System Authorization Facility (SAF) and construct a read-only key database. Only trusted certificates connected to the specified key ring are included in the key database. The GSKDB_RECFLAG_DEFAULT flag will be set if the certificate is the default certificate for the key ring or token.

The user must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by the user. The user must have UPDATE access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by another user.

The application user ID must have READ access to resource USER.tokenname in the CRYPTOZ class in order for the certificates and their private keys, if present, to be read from a z/OS PKCS #11 token.

gsk_perform_kat()

Conducts a set of known answer tests for the System SSL algorithms validated by NIST. The caller must set FIPS mode (see "gsk_fips_state_set()" on page 323) before calling this function.

Format

```
#include <gskcms.h>

gsk_status gsk_perform_kat ()
```

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors

[CMSERR_API_NOT_SUPPORTED]
The API is not supported in non-FIPS mode.

[CMSERR_KATPW_FAILED]
A known answer test has failed. This is a severe error and the application should terminate.

[CMSERR_KATPW_ICSF_FAILED]
A known answer test failed because ICSF was not available or ICSF encountered an error.

Usage

The **gsk_perform_kat()** routine can be used whenever an application, in order to meet security requirements, needs to check the correctness of cryptographic algorithms that are part of the product. The routine performs Known Answer Tests on the following cryptographic algorithms

- AES CBC 128-bit and AES CBC 256-bit encryption and decryption
- DSA signature generation and verification
- RSA signature generation/verification and encryption/decryption
- SHA Digest Algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, HMAC-SHA-1, HMAC-SHA-256, and HMAC-SHA-384
- TLS V1.0, V1.1 and V1.2 key derivation function
- TripleDES encryption and decryption

If an error is encountered during testing, the **gsk_perform_kat()** routine will terminate and return the appropriate error code.

The **gsk_perform_kat()** routine will test software or hardware cryptographic algorithms depending on the value of the GSK_HW_CRYPTO environment variable.

gsk_query_crypto_level()

Returns the available cryptographic levels.

Format

```
#include <gskcms.h>

void gsk_query_crypto_level (
                             int *                cms_version,
                             int *                cms_release,
                             gsk_uint32 *         crypto_level)
```

Parameters

cms_version
Returns the runtime version number.

cms_release
Returns the runtime release number.

crypto_level
Returns the available cryptographic levels.

Results

The **gsk_query_crypto_level()** routine returns the System SSL run time version, release, and available cryptographic levels. The current System SSL run time is Version 4 Release 5. The cryptographic level is a bit mask as follows:

[GSK_CRYPTO_64]
Set if 64-bit encryption keys are supported.

[GSK_CRYPTO_128]
Set if 128-bit encryption keys are supported.

[GSK_CRYPTO_168]
Set if 168-bit encryption keys are supported.

gsk_query_database_label()

Determines if a database label exists

Format

```
#include <gskcms.h>

gsk_status gsk_query_database_label (
                                    gsk_handle             db_handle,
                                    const char *           label)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine.

label
Specifies the database label. The label is specified in the local code page.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_LABEL]
No label specified.

[CMSERR_MULTIPLE_LABEL]
Multiple certificates exist for label.

[CMSERR_RECORD_NOT_FOUND]
The label does not exist in the database.

Usage

The **gsk_query_database_label()** routine will check the database for the requested label.

gsk_query_database_record_length()

Queries the database record length.

Format

```
#include <gskcms.h>

gsk_status gsk_query_database_record_length (
                                             gsk_handle          db_handle,
                                             gsk_size *          record_length)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine.

record_length
Returns the current database record length. All records in the database have this length.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_HANDLE]
The database handle is not valid.

Usage

The **gsk_query_database_record_length()** routine will return the record length for the database. All records in the database have the same length and a database entry cannot span records. The **gsk_change_database_record_length()** routine can be called to change the database record length.

gsk_rdtime()

Converts the number of seconds since the POSIX epoch to year/month/day.

Format

```
#include <gskcms.h>

gsk_timeval * gsk_rdtime (
                          gsk_time            secs,
                          gsk_timeval *       ts)
```

Parameters

secs
: Specifies the time value to be converted.

ts
: Returns the converted time in the tm_year, tm_mon, tm_mday, tm_hour, tm_min, and tm_sec fields.

Usage

The **gsk_rdtime()** routine converts the number of seconds since the POSIX epoch (January 1, 1970) to year/month/day format. The year value is the actual year minus 1900 and the month value is the actual month minus 1 (that is, January is 0 and December is 11). The return value is the same as the second parameter (the address of the struct tm).

gsk_read_content_msg()

Processes a PKCS #7 message.

Format

```
#include <gskcms.h>

gsk_status gsk_read_content_msg (
                                 gsk_buffer *           stream,
                                 pkcs_content_info *    content_info)
```

Parameters

stream
Specifies the ASN.1 DER-encoded stream to be processed.

content_info
Returns the content information for the message. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_NO_MEMORY]
Insufficient storage is available

Usage

The **gsk_read_content_msg()** routine processes a PKCS #7 (Cryptographic Message Syntax) content information message and returns the content information. The message content type can be any of the types defined by the PKCS #7 specification.

gsk_read_data_content()

Processes PKCS #7 Data content information.

Format

```
#include <gskcms.h>

gsk_status gsk_read_data_content (
                                  pkcs_content_info *         content_info,
                                  gsk_buffer *                data)
```

Parameters

content_info
Specifies the content information to be processed.

data
Returns the application data. The application should call the **gsk_free_buffer()** routine to release the data when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not Data.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_read_data_content()** routine processes PKCS #7 (Cryptographic Message Syntax) Data content information created by the **gsk_make_data_content()** routine and returns the application data.

gsk_read_data_msg()

Processes a PKCS #7 Data message.

Format

```
#include <gskcms.h>

gsk_status gsk_read_data_msg (
                              gsk_buffer *        stream,
                              gsk_buffer *        data)
```

Parameters

stream
Specifies the ASN.1 DER-encoded stream to be processed.

data
Returns the application data. The application should call the **gsk_free_buffer()** routine to release the data when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not Data.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_read_data_msg()** routine processes a PKCS #7 (Cryptographic Message Syntax) Data message created by the **gsk_make_data_msg()** routine and returns the application data. The message content type must be Data.

Calling the **gsk_read_data_msg()** routine is equivalent to calling the **gsk_read_content_msg()** routine followed by the **gsk_read_data_content()** routine.

gsk_read_encrypted_data_content()

Processes PKCS #7 EncryptedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_read_encrypted_data_content (
                                            const char *            password,
                                            pkcs_content_info *     content_info,
                                            pkcs_content_info *     content_data)
```

Parameters

password
: Specifies the encryption password as a null-terminated string in the local code page. The user will be prompted to enter the password if NULL is specified for this parameter.

content_info
: Specifies the content information to be processed

content_data
: Returns the decrypted content data. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_AVAILABLE]
: Encryption algorithm is not available.

[CMSERR_ALG_NOT_SUPPORTED]
: Encryption algorithm is not supported.

[CMSERR_API_NOT_SUPPORTED]
: The API is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
: The message content type is not EncryptedData or the content of the EncryptedData message is not supported.

[CMSERR_NO_CONTENT_DATA]
: The encrypted data length is zero.

[CMSERR_NO_MEMORY]
: Insufficient storage is available.

Usage

The **gsk_read_encrypted_data_content()** routine processes PKCS #7 (Cryptographic Message Syntax) EncryptedData content information created by the **gsk_make_encrypted_data_content()** routine and returns the decrypted content data.

gsk_read_encrypted_data_content() is not supported when executing in FIPS mode and will return CMSERR_API_NOT_SUPPORTED.

The decryption key is derived from the password as described in RFC 2898 (tools.ietf.org/html/rfc2898) and *PKCS #12, Version 1.0: Personal Information Exchange*. The selected algorithm determines how the key is derived from the password.

These password-based encryption algorithms are supported. The strong encryption algorithms might not be available depending upon government export regulations.

- **x509_alg_pbeWithMd2AndDesCbc - 56-bit DES encryption with MD2 digest - {1.2.840.113549.1.5.1}**
- **x509_alg_pbeWithMd5AndDesCbc - 56-bit DES encryption with MD5 digest - {1.2.840.113549.1.5.3}**
- **x509_alg_pbeWithSha1AndDesCbc - 56-bit DES encryption with SHA-1 digest - {1.2.840.113549.1.5.10}**
- **x509_alg_pbeWithMd2AndRc2Cbc - 64-bit RC2 encryption with MD2 digest - {1.2.840.113549.1.5.4}**
- **x509_alg_pbeWithMd5AndRc2Cbc - 64-bit RC2 encryption with MD5 digest - {1.2.840.113549.1.5.6}**
- **x509_alg_pbeWithSha1AndRc2Cbc - 64-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.5.11}**
- **x509_alg_pbeWithSha1And40BitRc2Cbc - 40-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.6}**
- **x509_alg_pbeWithSha1And128BitRc2Cbc - 128-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.5}**
- **x509_alg_pbeWithSha1And40BitRc4 - 40-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.2}**
- **x509_alg_pbeWithSha1And128BitRc4 - 128-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.1}**
- **x509_alg_pbeWithSha1And3DesCbc - 168-bit 3DES encryption with SHA-1 digest - {1.2.840.113549.1.12.1.3}**

gsk_read_encrypted_data_msg()

Processes a PKCS #7 EncryptedData message.

Format

```
#include <gskcms.h>

gsk_status gsk_read_encrypted_data_msg (
                                        const char *             password,
                                        gsk_buffer *             stream,
                                        gsk_buffer *             data)
```

Parameters

password
: Specifies the encryption password as a null-terminated string in the local code page. The user will be prompted to enter the password if NULL is specified for this parameter.

stream
: Specifies the ASN.1 DER-encoded stream to be processed.

data
: Returns the decrypted content of the EncryptedData message. The application should call the **gsk_free_buffer()** routine to release the data when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_AVAILABLE]
: Encryption algorithm is not available.

[CMSERR_ALG_NOT_SUPPORTED]
: Encryption algorithm is not supported.

[CMSERR_API_NOT_SUPPORTED]
: The API is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
: The message content type is not EncryptedData or the content of the EncryptedData message is not Data.

[CMSERR_NO_CONTENT_DATA]
: The encrypted data length is zero.

[CMSERR_NO_MEMORY]
: Insufficient storage is available.

Usage

The **gsk_read_encrypted_data_msg()** routine processes a PKCS #7 (Cryptographic Message Syntax) EncryptedData message created by the **gsk_make_encrypted_data_msg()** routine and returns the decrypted message content. The encrypted data content type must be Data.

gsk_read_encrypted_data_msg() is not supported when executing in FIPS mode and will return CMSERR_API_NOT_SUPPORTED.

Calling the **gsk_read_encrypted_data_msg()** routine is equivalent to calling the **gsk_read_content_msg()** routine, the **gsk_read_encrypted_data_content()** routine, and the **gsk_read_data_content()** routine.

The decryption key is derived from the password as described in RFC 2898 (tools.ietf.org/html/rfc2898) and *PKCS #12, Version 1.0: Personal Information Exchange*. The selected algorithm determines how the key is derived from the password.

These password-based encryption algorithms are supported. The strong encryption algorithms might not be available depending upon government export regulations.

- **x509_alg_pbeWithMd2AndDesCbc - 56-bit DES encryption with MD2 digest - {1.2.840.113549.1.5.1}**
- **x509_alg_pbeWithMd5AndDesCbc - 56-bit DES encryption with MD5 digest - {1.2.840.113549.1.5.3}**
- **x509_alg_pbeWithSha1AndDesCbc - 56-bit DES encryption with SHA-1 digest - {1.2.840.113549.1.5.10}**
- **x509_alg_pbeWithMd2AndRc2Cbc - 64-bit RC2 encryption with MD2 digest - {1.2.840.113549.1.5.4}**
- **x509_alg_pbeWithMd5AndRc2Cbc - 64-bit RC2 encryption with MD5 digest - {1.2.840.113549.1.5.6}**
- **x509_alg_pbeWithSha1AndRc2Cbc - 64-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.5.11}**
- **x509_alg_pbeWithSha1And40BitRc2Cbc - 40-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.6}**
- **x509_alg_pbeWithSha1And128BitRc2Cbc - 128-bit RC2 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.5}**
- **x509_alg_pbeWithSha1And40BitRc4 - 40-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.2}**
- **x509_alg_pbeWithSha1And128BitRc4 - 128-bit RC4 encryption with SHA-1 digest - {1.2.840.113549.1.12.1.1}**
- **x509_alg_pbeWithSha1And3DesCbc - 168-bit 3DES encryption with SHA-1 digest - {1.2.840.113549.1.12.1.3}**

gsk_read_enveloped_data_content()

Processes PKCS #7 EnvelopedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_read_enveloped_data_content (
                                  pkcs_cert_keys *          recipient_keys,
                                  pkcs_content_info *       content_info,
                                  x509_algorithm_type *     encryption_algorithm,
                                  gsk_size *                key_size,
                                  pkcs_content_info *       content_data)
```

Parameters

recipient_keys
Specifies one or more certificates and associated private keys.

content_info
Specifies the content information to be processed.

encryption_algorithm
Returns the encryption algorithm used to encrypt the message content.

key_size
Returns the encryption key size in bytes.

content_data
Returns the EnvelopedData content data. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_ALG_NOT_AVAILABLE]
The encryption algorithm is not available.

[CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not EnvelopedData or the content of the EnvelopedData message is not supported.

[CMSERR_CRYPTO_HARDWARE_NOT_AVAILABLE]
Cryptographic hardware does not support service or algorithm.

[CMSERR_INCORRECT_KEY_USAGE]
The recipient certificate does not allow key encipherment.

[CMSERR_KEY_MISMATCH]
A recipient private key does not support data decryption.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
Private key does not exist or is not accessible.

[CMSERR_RECIPIENT_NOT_FOUND]
No matching recipient certificate provided.

Usage

The **gsk_read_enveloped_data_content()** routine processes PKCS #7 (Cryptographic Message Syntax) EnvelopedData content information created by the **gsk_make_enveloped_data_content()** routine.

The *recipient_keys* parameter supplies one or more recipient certificates and associated private keys. The **gsk_read_enveloped_data_content()** routine will search for a certificate matching one of the message recipients. The private key will be used to decrypt the session key and the session key will then be used to decrypt the enveloped data. The certificate key usage must allow key encipherment.

No certificate validation is performed by the **gsk_read_enveloped_data_content()** routine. It is assumed that the application has already validated the recipient certificates.

These encryption algorithms are supported. Strong encryption might not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - {1.2.840.113549.3.4}**
- **x509_alg_desCbcPad - 56-bit DES - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - {1.2.840.113549.3.7}**
- **x509_alg_aesCbc128 - 128-bit AES CBC - {2.16.840.1.101.3.4.1.2}**
- **x509_alg_aesCbc256 - 256-bit AES CBC - {2.16.840.1.101.3.4.1.42}**

When executing in FIPS mode, encryption algorithms **x509_alg_rc2CbcPad, x509_alg_rc4** and **x509_alg_desCbcPad** are not supported.

gsk_read_enveloped_data_content_extended()

Processes PKCS #7 EnvelopedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_read_enveloped_data_content_extended (
                                  gsk_process_option          option_flag
                                  pkcs_cert_keys *            recipient_keys,
                                  pkcs_content_info *         content_info,
                                  x509_algorithm_type *       encryption_algorithm,
                                  gsk_size *                  key_size,
                                  pkcs_content_info *         content_data)
```

Parameters

option_flag
Specifies process options to customize process behavior.

- Enforce recipient certificate has key encipherment capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate keyEncipherment.
- Enforce key parity when using DES or 3DES session keys.

recipient_keys
Specifies one or more certificates and associated private keys.

content_info
Specifies the content information to be processed.

encryption_algorithm
Returns the encryption algorithm used to encrypt the message content.

key_size
Returns the encryption key size in bytes.

content_data
Returns the EnvelopedData content data. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it is one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_ALG_NOT_AVAILABLE]
The encryption algorithm is not available.

[CMSERR_ALG_NOT_SUPPORTED]
The encryption algorithm is not supported.

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not EnvelopedData or the content of the EnvelopedData message is not supported.

[CMSERR_CRYPTO_HARDWARE_NOT_AVAILABLE]
Cryptographic hardware does not support service or algorithm.

[CMSERR_INCORRECT_KEY_USAGE]
The recipient certificate does not allow key encipherment.

[CMSERR_KEY_MISMATCH]
A recipient private key does not support data decryption.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
Private key does not exist or is not accessible.

[CMSERR_RECIPIENT_NOT_FOUND]
No matching recipient certificate provided.

Usage

The **gsk_read_enveloped_data_content_extended()** routine processes PKCS #7 (Cryptographic Message Syntax) EnvelopedData content information that is created by the **gsk_make_enveloped_data_content()** routine, the **gsk_make_enveloped_data_content_extended()**, or the **gsk_make_enveloped_private_key_msg()** routine. Processing is equivalent to **gsk_read_enveloped_data_content()**, except that the recipient certificate key usage need not assert key encipherment.

The *recipient_keys* parameter supplies one or more recipient certificates and associated private keys. The **gsk_read_enveloped_data_content_extended()** routine searches for a certificate matching one of the message recipients. The private key will be used to decrypt the session key and the session key will then be used to decrypt the enveloped data. In addition, if *option_flag* specifies that key encipherment is to be enforced, then the certificate key usage must allow key encipherment and session keys need not be odd parity.

No certificate validation is performed by the **gsk_read_enveloped_data_content_extended()** routine. It is assumed that the application has already validated the recipient certificates.

These encryption algorithms are supported. Strong encryption might not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - {1.2.840.113549.3.4}**
- **x509_alg_desCbcPad - 56-bit DES - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - {1.2.840.113549.3.7}**
- **x509_alg_aesCbc128 - 128-bit AES CBC - {2.16.840.1.101.3.4.1.2}**
- **x509_alg_aesCbc256 - 256-bit AES CBC - {2.16.840.1.101.3.4.1.42}**

When executing in FIPS mode, encryption algorithms **x509_alg_rc2CbcPad, x509_alg_rc4** and **x509_alg_desCbcPad** are not supported.

gsk_read_enveloped_data_msg()

Processes a PKCS #7 EnvelopedData message.

Format

```
#include <gskcms.h>

gsk_status gsk_read_enveloped_data_msg (
                                  pkcs_cert_keys *         recipient_keys,
                                  gsk_buffer *             stream,
                                  x509_algorithm_type *    encryption_algorithm,
                                  gsk_size *               key_size,
                                  gsk_buffer *             data)
```

Parameters

recipient_keys
Specifies one or more certificates and associated private keys.

stream
Specifies the ASN.1 DER-encoded stream to be processed.

encryption_algorithm
Returns the encryption algorithm used to encrypt the message content.

key_size
Returns the encryption key size in bytes.

data
Returns the content of the EnvelopedData message. The application should call the **gsk_free_buffer()** routine to release the data when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_ALG_NOT_SUPPORTED]
Encryption algorithm is not supported.

[CMSERR_BAD_ENCODING]
The message content type is not EnvelopedData or the message content is not Data.

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not EnvelopedData or the content of the EnvelopedData message is not Data.

[CMSERR_CRYPTO_HARDWARE_NOT_AVAILABLE]
Cryptographic hardware does not support service or algorithm.

[CMSERR_INCORRECT_KEY_USAGE]
The recipient certificate does not allow key encipherment.

[CMSERR_KEY_MISMATCH]
A recipient private key does not support data decryption.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
Private key does not exist or is not accessible.

[CMSERR_RECIPIENT_NOT_FOUND]
No matching recipient certificate provided.

Usage

The **gsk_read_enveloped_data_msg()** routine processes a PKCS #7 (Cryptographic Message Syntax) EnvelopedData message created by the **gsk_make_enveloped_data_msg()** routine and returns the message content. The enveloped data content type must be Data.

Calling the **gsk_read_enveloped_data_msg()** routine is equivalent to calling the **gsk_read_content_msg()** routine, the **gsk_read_enveloped_data_content()** routine, and the **gsk_read_data_content()** routine.

The recipient_keys parameter supplies one or more recipient certificates and associated private keys. The **gsk_read_enveloped_data_msg()** routine will search for a certificate matching one of the message recipients. The private key will be used to decrypt the session key and the session key will then be used to decrypt the enveloped data. The certificate key usage must allow key encipherment.

No certificate validation is performed by the **gsk_read_enveloped_data_msg()** routine. It is assumed that the application has already validated the recipient certificates.

These encryption algorithms are supported. Strong encryption might not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - {1.2.840.113549.3.4}**
- **x509_alg_desCbcPad - 56-bit DES - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - {1.2.840.113549.3.7}**
- **x509_alg_aesCbc128 - 128-bit AES CBC - {2.16.840.1.101.3.4.1.2}**
- **x509_alg_aesCbc256 - 256-bit AES CBC - {2.16.840.1.101.3.4.1.42}**

When executing in FIPS mode, encryption algorithms **x509_alg_rc2CbcPad, x509_alg_rc4** and **x509_alg_desCbcPad** are not supported.

gsk_read_enveloped_data_msg_extended()

Processes a PKCS #7 EnvelopedData message.

Format

```
#include <gskcms.h>

gsk_status gsk_read_enveloped_data_msg_extended (
                                   gsk_process_option      option_flag,
                                   pkcs_cert_keys *        recipient_keys,
                                   gsk_buffer *            stream,
                                   x509_algorithm_type *   encryption_algorithm,
                                   gsk_size *              key_size,
                                   gsk_buffer *            data)
```

Parameters

option_flag
Specifies process options to customize process behavior.

- Enforce recipient certificate has key encipherment capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate keyEncipherment.
- Enforce key parity when using DES or 3DES session keys.

recipient_keys
Specifies one or more certificates and associated private keys.

stream
Specifies the ASN.1 DER-encoded stream to be processed.

encryption_algorithm
Returns the encryption algorithm used to encrypt the message content.

key_size
Returns the encryption key size in bytes.

data
Returns the content of the EnvelopedData message. The application should call the **gsk_free_buffer()** routine to release the data when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_3DES_KEY_PARTS_NOT_UNIQUE]
Triple DES key parts are not unique.

[CMSERR_ALG_NOT_SUPPORTED]
Encryption algorithm is not supported.

[CMSERR_BAD_ENCODING]
The message content type is not EnvelopedData or the message content is not Data.

[CMSERR_BAD_KEY_SIZE]
The encryption key size is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not EnvelopedData or the content of the EnvelopedData message is not Data.

[CMSERR_CRYPTO_HARDWARE_NOT_AVAILABLE]
Cryptographic hardware does not support service or algorithm.

[CMSERR_INCORRECT_KEY_USAGE]
The recipient certificate does not allow key encipherment.

[CMSERR_KEY_MISMATCH]
A recipient private key does not support data decryption.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
Private key does not exist or is not accessible.

[CMSERR_RECIPIENT_NOT_FOUND]
No matching recipient certificate provided.

Usage

The **gsk_read_enveloped_data_msg_extended()** routine processes a PKCS #7 (Cryptographic Message Syntax) EnvelopedData message created by the **gsk_make_enveloped_data_content()** routine or the **gsk_make_enveloped_data_msg_extended()** routine and returns the message content. Processing is equivalent to **gsk_read_enveloped_data_content()**, except that the recipient certificate key usage need not assert key encipherment and session keys need not be odd parity. The enveloped data content type must be Data.

Calling the **gsk_read_enveloped_data_msg_extended()** routine is equivalent to calling the **gsk_read_content_msg()** routine, the **gsk_read_enveloped_data_content_extended()** routine, and the **gsk_read_data_content()** routine.

The recipient_keys parameter supplies one or more recipient certificates and associated private keys. The **gsk_read_enveloped_data_msg_extended()** routine will search for a certificate matching one of the message recipients. The private key will be used to decrypt the session key and the session key will then be used to decrypt the enveloped data. If *option_flag* specifies that key encipherment is to be enforced, then the certificate key usage must allow key encipherment.

No certificate validation is performed by the **gsk_read_enveloped_data_msg_extended()** routine. It is assumed that the application has already validated the recipient certificates.

These encryption algorithms are supported. Strong encryption might not be available depending upon government export regulations.

- **x509_alg_rc2CbcPad - 40-bit and 128-bit RC2 - {1.2.840.113549.3.2}**
- **x509_alg_rc4 - 40-bit and 128-bit RC4 - {1.2.840.113549.3.4}**
- **x509_alg_desCbcPad - 56-bit DES - {1.3.14.3.2.7}**
- **x509_alg_desEde3CbcPad - 168-bit 3DES - {1.2.840.113549.3.7}**
- **x509_alg_aesCbc128 - 128-bit AES CBC - {2.16.840.1.101.3.4.1.2}**
- **x509_alg_aesCbc256 - 256-bit AES CBC - {2.16.840.1.101.3.4.1.42}**

When executing in FIPS mode, encryption algorithms **x509_alg_rc2CbcPad, x509_alg_rc4** and **x509_alg_desCbcPad** are not supported.

gsk_read_signed_data_content()

Processes PKCS #7 SignedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_read_signed_data_content (
                                pkcs_certificates *      local_certificates,
                                pkcs_content_info *      content_info,
                                gsk_boolean *            used_local,
                                pkcs_certificates *      msg_certificates,
                                pkcs_certificates *      signer_certificates,
                                pkcs_content_info *      content_data)
```

Parameters

local_certificates
Specifies zero or more X.509 certificates to use when verifying the message signatures. NULL can be specified for this parameter if no local certificates are provided.

content_info
Specifies the content information to be processed.

used_local
This parameter will be set to TRUE if the signatures were verified using just the certificates supplied by the local_certificates parameter. This parameter will be set to FALSE if any of the signatures were verified using certificates contained within the message.

msg_certificates
Returns the X.509 certificates contained within the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the message certificates are not needed.

signer_certificates
Returns the certificates used to sign the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the signer certificates are not needed.

content_data
Returns the SignedData content data. The application should call the **gsk_free_content_info()** routine to release the data when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_BAD_SIGNATURE]
Signature is not correct.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not SignedData.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_MASK_ALG_NOT_SUPPORTED]
RSASSA-PSS mask generation algorithm is not supported.

[CMSERR_SIGNER_NOT_FOUND]
Signer certificate not found.

Usage

The **gsk_read_signed_data_content()** routine processes PKCS #7 (Cryptographic Message Syntax) SignedData message created by the **gsk_make_signed_data_content()** routine and returns the content data.

The *local_certificates* parameter can supply the signer certificates used to verify the message signatures. If a certificate is not found for a message signer, the **gsk_read_signed_data_content()** routine attempts to locate the signer certificate in the SignedData message. An error will be returned if the signer certificate cannot be found or if the certificate key usage does not allow digital signature.

No certificate validation is performed by the **gsk_read_signed_data_content()** routine. It is assumed that the application has already validated the local certificates. The certificates contained in the SignedData message will be returned in the *msg_certificates* parameter and the *used_local* parameter will be set to FALSE if any of these certificates were used to verify the message signatures. It is the responsibility of the application to validate the message certificates (for example, by calling the **gsk_validate_certificate()** routine for each of the signer certificates).

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.3}

These RSASSA-PSS [1.2.840.113549.1.1.10] signatures are supported:

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation algorithm 1.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_read_signed_data_content_extended()

Processes PKCS #7 SignedData content information.

Format

```
#include <gskcms.h>

gsk_status gsk_read_signed_data_content_extended (
                                gsk_process_option          option_flag
                                pkcs_certificates *         local_certificates,
                                pkcs_content_info *         content_info,
                                gsk_boolean *               used_local,
                                pkcs_certificates *         msg_certificates,
                                pkcs_certificates *         signer_certificates,
                                gsk_attributes_signers *    attributes_signers,
                                pkcs_content_info *         content_data)
```

Parameters

option_flag
Specifies process options to customize process behavior.

- Enforce signing certificate has digital signing capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate digitalSignature.
- Do not allow zero-length content data.

local_certificates
Specifies zero or more X.509 certificates to use when verifying the message signatures. NULL can be specified for this parameter if no local certificates are provided.

content_info
Specifies the content information to be processed.

used_local
This parameter will be set to TRUE if the signatures were verified using just the certificates supplied by the local_certificates parameter. This parameter will be set to FALSE if any of the signatures were verified using certificates contained within the message.

msg_certificates
Returns the X.509 certificates contained within the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the message certificates are not needed.

signer_certificates
Returns the certificates used to sign the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the signer certificates are not needed.

attributes_signers
Returns the authenticated attributes per signer contained within the message. The application should call the **gsk_free_attributes_signers()** routine to release the gsk_attributes_signers structure when it is no longer needed. Specify NULL for this parameter if the authenticated attributes per signer are not needed. The set of authenticated attributes returned, omits the content-type and message-digest authenticated attributes as these authenticated attributes must always be present, if any authenticated attributes are present, and are automatically verified by **gsk_read_signed_data_content_extended()**. The *digestAlgorithm* field within each *gsk_attributes_signer* structure returns the digest algorithm originally used for the signer.

content_data
Returns the SignedData content data. The application should call the **gsk_free_content_info()** routine to release the data when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_BAD_SIGNATURE]
Signature is not correct.

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not SignedData.

[CMSERR_CRYPTO_FAILED]
Unexpected cryptographic request failure.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_MASK_ALG_NOT_SUPPORTED]
RSASSA-PSS mask generation algorithm is not supported.

[CMSERR_SIGNER_NOT_FOUND]
Signer certificate not found.

Usage

The **gsk_read_signed_data_content_extended()** routine processes PKCS #7 (Cryptographic Message Syntax) SignedData message created by the **gsk_make_signed_data_content_extended()** routine and returns the content data and authenticated attributes per signed (if present).

Processing is equivalent to **gsk_read_signed_data_content()**, with these differences:

- The signing certificate key usage need not assert digital signing capabilities depending on *option_flag*.
- Zero length content is acceptable depending on *option_flag*.
- Authenticated attributes and the digest algorithm used to create the signed data per signer, if present, are returned.

The *local_certificates* parameter can supply the signer certificates used to verify the message signatures. If a certificate is not found for a message signer, the **gsk_read_signed_data_content_extended()** routine attempts to locate the signer certificate in the SignedData message. An error will be returned if the signer certificate cannot be found. An error may optionally be returned if the certificate key usage does not allow digital signature.

No certificate validation is performed by the **gsk_read_signed_data_content_extended()** routine. It is assumed that the application has already validated the local certificates. The certificates contained in the SignedData message will be returned in the *msg_certificates* parameter and the *used_local* parameter will be set to FALSE if any of these certificates were used to verify the message signatures. It is the responsibility of the application to validate the message certificates (for example, by calling the **gsk_validate_certificate()** routine for each of the signer certificates).

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.3}

These RSASSA-PSS [1.2.840.113549.1.1.10] signatures are supported:

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation algorithm 1.

If authenticated attributes are returned from the *attributes_signers* parameter, then it is recommended that signing certificates for all signers represented within the gsk_attributes_signers structure should be requested from the *signer_certificates* parameter.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_read_signed_data_msg()

Processes a PKCS #7 SignedData message.

Format

```
#include <gskcms.h>

gsk_status gsk_read_signed_data_msg (
                      pkcs_certificates *      local_certificates,
                      gsk_buffer *             stream,
                      gsk_boolean *            used_local,
                      pkcs_certificates *      msg_certificates,
                      pkcs_certificates *      signer_certificates,
                      gsk_buffer *             data)
```

Parameters

local_certificates
Specifies zero or more X.509 certificates to use when verifying the message signatures. NULL can be specified for this parameter if no local certificates are provided.

stream
Specifies the ASN.1 DER-encoded stream to be processed.

used_local
This parameter will be set to TRUE if the signatures were verified using just the certificates supplied by the local_certificates parameter. This parameter will be set to FALSE if any of the signatures were verified using certificates contained within the message.

msg_certificates
Returns the X.509 certificates contained within the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the message certificates are not needed.

signer_certificates
Returns the certificates used to sign the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the signer certificates are not needed.

data
Returns the content of the SignedData message. The application should call the **gsk_free_buffer()** routine to release the data when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_NO_MEMORY]
Insufficient storage is available.

[ASN_SELECTION_OUT_OF_RANGE]
Certificate type or version number is not valid.

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_BAD_SIGNATURE]
Signature is not correct.

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not SignedData or the content of the SignedData message is not Data.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_MASK_ALG_NOT_SUPPORTED]
RSASSA-PSS mask generation algorithm is not supported.

[CMSERR_SIGNER_NOT_FOUND]
Signer certificate not found.

Usage

The **gsk_read_signed_data_msg()** routine processes a PKCS #7 (Cryptographic Message Syntax) SignedData message created by the **gsk_make_signed_data_msg()** routine and returns the message content. The signed data content type must be Data.

Calling the **gsk_read_signed_data_msg()** routine is equivalent to calling the **gsk_read_content_msg()** routine, the **gsk_read_signed_data_content()** routine, and the **gsk_read_data_content()** routine.

The *local_certificates* parameter can supply the signer certificates used to verify the message signatures. If a certificate is not found for a message signer, the **gsk_read_signed_data_msg()** routine attempts to locate the signer certificate in the SignedData message. An error will be returned if the signer certificate cannot be found or if the certificate key usage does not allow digital signature.

No certificate validation is performed by the **gsk_read_signed_data_msg()** routine. It is assumed that the application has already validated the local certificates. The certificates contained in the SignedData message will be returned in the *msg_certificates* parameter and the *used_local* parameter will be set to FALSE if any of these certificates were used to verify the message signatures. It is the responsibility of the application to validate the message certificates (for example, by calling the **gsk_validate_certificate()** routine for each of the signer certificates).

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.3}

These RSASSA-PSS [1.2.840.113549.1.1.10] signatures are supported:

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation function algorithm 1.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_read_signed_data_msg_extended()

Processes a PKCS #7 SignedData message.

Format

```
#include <gskcms.h>

gsk_status gsk_read_signed_data_msg_extended (
                             gsk_process_option          option_flag
                             pkcs_certificates *         local_certificates,
                             gsk_buffer *                stream,
                             gsk_boolean *               used_local,
                             pkcs_certificates *         msg_certificates,
                             pkcs_certificates *         signer_certificates,
                             gsk_attributes_signers *    attributes_signers,
                             gsk_buffer *                data)
```

Parameters

option_flag
Specifies process options to customize process behavior.

- Enforce signing certificate has digital signing capabilities. That is, the purpose of the certificate key as reflected by the key usage extension must indicate digitalSignature.
- Do not allow zero-length content data.

local_certificates
Specifies zero or more X.509 certificates to use when verifying the message signatures. NULL can be specified for this parameter if no local certificates are provided.

stream
Specifies the ASN.1 DER-encoded stream to be processed.

used_local
This parameter is set to TRUE if the signatures were verified by using just the certificates that are supplied by the local_certificates parameter. This parameter is set to FALSE if any of the signatures were verified by using certificates that are contained within the message.

msg_certificates
Returns the X.509 certificates that are contained within the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the message certificates are not needed.

signer_certificates
Returns the certificates that are used to sign the message. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed. Specify NULL for this parameter if the signer certificates are not needed.

attributes_signers
Returns the authenticated attributes per signer that is contained within the message. The application should call the **gsk_free_attributes_signers()** routine to release the gsk_attributes_signers structure when it is no longer needed. Specify NULL for this parameter if the authenticated attributes per signer are not needed. The set of authenticated attributes returned, omits the content-type and message-digest authenticated attributes as these authenticated attributes must always be present, if any authenticated attributes are present, and are automatically verified by **gsk_read_signed_data_msg_extended()**. The *digestAlgorithm* field within each *gsk_attributes_signer* structure returns the digest algorithm that is originally used for the signer.

data
Returns the content of the SignedData message. The application should call the **gsk_free_buffer()** routine to release the data when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it is one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[ASN_NO_MEMORY]
Insufficient storage is available.

[ASN_SELECTION_OUT_OF_RANGE]
Certificate type or version number is not valid.

[CMSERR_ALG_NOT_SUPPORTED]
The digest algorithm is not supported.

[CMSERR_CONTENT_NOT_SUPPORTED]
The message content type is not SignedData or the content of the SignedData message is not Data.

[CMSERR_BAD_SIGNATURE]
Signature is not correct.

[CMSERR_DIGEST_KEY_MISMATCH]
The digest algorithm is not supported for the private key type.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_KEY_USAGE]
A signer certificate does not allow digital signature.

[CMSERR_NO_CONTENT_DATA]
The content data length is zero.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_MASK_ALG_NOT_SUPPORTED]
RSASSA-PSS mask generation algorithm is not supported.

[CMSERR_SIGNER_NOT_FOUND]
Signer certificate not found.

Usage

The **gsk_read_signed_data_msg_extended()** routine processes a PKCS #7 (Cryptographic Message Syntax) SignedData message that is created by the **gsk_make_signed_data_msg_extended()** routine and returns the message content and all authenticated attributes (if present). The signed data content type must be Data.

Processing is equivalent to **gsk_read_signed_data_msg()**, with these differences:

- The signing certificate key usage need not assert digital signing capabilities depending on *option_flag*.
- Zero length content is acceptable depending on *option_flag*.
- Authenticated attributes and the digest algorithm that is used to create the signed data per signer, if present, are returned.

Calling the **gsk_read_signed_data_msg_extended()** routine is equivalent to calling the **gsk_read_content_msg()** routine, the **gsk_read_signed_data_content_extended()** routine, and the **gsk_read_data_content()** routine.

The *local_certificates* parameter can supply the signer certificates that are used to verify the message signatures. If a certificate is not found for a message signer, the **gsk_read_signed_data_msg_extended()** routine attempts to locate the signer certificate in the SignedData message. An error is returned if the signer certificate cannot be found. An error may optionally be returned if the certificate key usage does not allow digital signature.

No certificate validation is performed by the **gsk_read_signed_data_msg_extended()** routine. It is assumed that the application validated the local certificates. The certificates that are contained in the SignedData message are returned in the *msg_certificates* parameter and the *used_local* parameter is set to FALSE if any of these certificates were used to verify the message signatures. It is the responsibility of the application to validate the message certificates (for example, by calling the **gsk_validate_certificate_mode()** routine for each of the signer certificates).

These digest algorithms are supported:

x509_alg_md2Digest
MD2 digest (RSA keys only) - {1.2.840.113549.2.2}

x509_alg_md5Digest
MD5 digest (RSA keys only) - {1.2.840.113549.2.5}

x509_alg_sha1Digest
SHA-1 digest (RSA, DSA, and ECDSA keys only) - {1.3.14.3.2.26}

x509_alg_sha224Digest
SHA-224 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.4}

x509_alg_sha256Digest
SHA-256 digest (RSA, DSA, and ECDSA keys only) - {2.16.840.1.101.3.4.2.1}

x509_alg_sha384Digest
SHA-384 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.2}

x509_alg_sha512Digest
SHA-512 digest (RSA and ECDSA keys only) - {2.16.840.1.101.3.4.2.3}

These RSASSA-PSS [1.2.840.113549.1.1.10] signatures are supported:

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation function algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation function algorithm 1.

If authenticated attributes are returned from the *attributes_signers* parameter, then it is recommended that signing certificates for all signers represented within the gsk_attributes_signers structure should be requested from the *signer_certificates* parameter.

When executing in FIPS mode, digest algorithms x509_alg_md2Digest and x509_alg_md5Digest are not supported.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_read_wrapped_content()

Processes wrapped content information.

Format

```
#include <gskcms.h>

gsk_status gsk_read_wrapped_content (
                                     pkcs_content_info *          wrapped_content,
                                     pkcs_content_info *          content_info)
```

Parameters

wrapped_content
Specifies the wrapped content information.

content_info
Returns the content information. The application should call the **gsk_free_content_info()** routine to release the content information when it is no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_CONTENT_NOT_SUPPORTED]
The content type is not supported.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_read_wrapped_content()** routine processes an ASN.1 sequence containing encoded content information and returns the unwrapped content information.

gsk_receive_certificate()

Receives one or more certificates.

Format

```
#include <gskcms.h>

gsk_status gsk_receive_certificate (
                                    gsk_buffer *              stream,
                                    pkcs_certificates *       certificates)
```

Parameters

stream
Specifies the byte stream of the encoded certificate.

certificate
Returns the decoded certificates. The application should call the **gsk_free_certificates()** routine to release the certificates when they are no longer needed.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_BASE64_ENCODING]
The Base64 encoding of the import file is not correct.

[CMSERR_BAD_ENCODING]
The import file format is not recognized.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

Usage

The **gsk_receive_certificate()** routine receives one or more X.509 certificates and returns the decoded certificates to the caller.

The supplied stream can represent either the ASN.1 DER encoding for the certificate or the Cryptographic Message Syntax (PKCS #7) encoding for the certificate. This can be either the binary value or the Base64 encoding of the binary value. A Base64 encoded stream must be in the local code page and must include the encoding header and footer lines.

A Base64 DER-encoded sequence must start with the encoding header '-----BEGIN CERTIFICATE-----' and end with the encoding footer '----END CERTIFICATE-----'. A Base 64 PKCS #7 signed data message must start with the encoding header '-----BEGIN CERTIFICATE-----' and end with the encoding footer '----END CERTIFICATE-----' or must start with the encoding header '----BEGIN PKCS #7 SIGNED DATA-----' and end with the encoding footer '-----END PKCS #7 SIGNED DATA-----'.

A DER-encoded certificate stream contains a single X.509 certificate while a PKCS #7 message stream contains one or more certificates. All of the certificates in a PKCS #7 message will be returned to the application for processing.

gsk_replace_record()

Replaces a record in a key or request database.

Format

```
#include <gskcms.h>

gsk_status gsk_replace_record (
                               gsk_handle                 db_handle,
                               gskdb_record *             record)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine.

record
Specifies the database record.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_LABEL]
The record label is not valid.

[CMSERR_BAD_RNG_OUTPUT]
In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_DEFAULT_KEY_CHANGED]
The default key cannot be changed.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The record type is not supported for the database type.

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_LABEL_NOT_UNIQUE]
The record label is not unique.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
No private key is provided for a record type that requires a private key.

[CMSERR_PUBLIC_KEY_CHANGED]
The subject public key cannot be changed.

[CMSERR_RECORD_NOT_FOUND]
Record is not found.

[CMSERR_RECORD_TOO_BIG]
The record is larger than the database record length.

[CMSERR_RECTYPE_NOT_VALID]
The record type is not valid.

[CMSERR_SUBJECT_CHANGED]
The subject name cannot be changed.

[CMSERR_UNSUPPORTED_EXPIRATION]
The expiration date exceeds February 6, 2106, at 23:59:59 UTC.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_replace_record()** routine replaces a record in a key or request database. The database must be open for update in order to replace records. The unique record identifier identifies the record to be replaced. Unused and reserved fields in the gskdb_record structure must be initialized to zero. If the record has a private key, the encrypted private key will be generated from the private key supplied in the database record.

The recordType field identifies the database record type as follows:

gskdb_rectype_certificate
The record contains an X.509 certificate.

gskdb_rectype_certKey
The record contains an X.509 certificate and private key.

gskdb_rectype_keyPair
The record contains a PKCS #10 certification request and private key.

The recordFlags field is a bit field with these values:

GSKDB_RECFLAG_TRUSTED
The certificate is trusted.

GSKDB_RECFLAG_DEFAULT
This is the default key

The record label is used as a friendly name for the database entry and is in the local code page. It can be set to any value and consists of characters which can be represented using 7-bit ASCII (letters, numbers, and punctuation). It may not be set to an empty string.

If the record contains a certificate, the certificate will be validated and the record will not be replaced in the database if the validation check fails. If executing in FIPS mode, only FIPS-approved algorithms and key sizes are supported.

With the exception of the record label, all character strings are specified using UTF-8.

The record type, subject name, and subject public key cannot be changed when replacing a record. In addition, the GSKDB_RECFLAG_DEFAULT flag cannot be changed when replacing a record (call the **gsk_set_default_key()** routine to change the default record for the database).

The database file is updated as part of the **gsk_replace_record()** processing. A temporary database file is created using the same name as the database file with ".new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_set_default_key()

Sets the default key.

Format

```
#include <gskcms.h>

gsk_status gsk_set_default_key (
                                gsk_handle          db_handle,
                                gsk_int32           record_id)
```

Parameters

db_handle
Specifies the database handle returned by the **gsk_create_database()** routine or the **gsk_open_database()** routine.

record_id
Specifies the unique record identifier of the new default key.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BACKUP_EXISTS]
The backup file already exists.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support a default key.

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_IO_ERROR]
Unable to write record.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
The database record does not contain a private key.

[CMSERR_RECORD_NOT_FOUND]
Record is not found.

[CMSERR_UPDATE_NOT_ALLOWED]
Database is not open for update or update attempted on a FIPS mode database while in non-FIPS mode.

Usage

The **gsk_set_default_key()** routine sets the default key for a key database. If the key database already has a default key, the record for the old default key is updated to remove the GSKDB_RECFLAG_DEFAULT flag. The record for the new default key is then updated to add the GSKDB_RECFLAG_DEFAULT flag. The database must be open for update in order to set the default key. An error will be returned if the specified database record does not contain a private key.

The database file is updated as part of the **gsk_set_default_key()** processing. A temporary database file is created using the same name as the database file with ".new" appended to the name. The database file is then overwritten and the temporary database file is deleted. The temporary database file will not be deleted if an error occurs while rewriting the database file.

gsk_set_directory_enum()

Sets an enumerated value for an LDAP directory.

Format

```
#include <gskcms.h>

gsk_status gsk_set_directory_enum (
                                gsk_handle                        directory_handle,
                                GSKCMS_DIRECTORY_ENUM_ID          enum_id,
                                GSKCMS_DIRECTORY_ENUM_VALUE       enum_value)
```

Parameters

directory_handle
Specifies an LDAP directory handle returned by **gsk_open_directory()**.

enum_id
Specifies the directory enumeration identifier.

enum_value
Specifies the directory enumeration value.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ATTRIBUTE_INVALID_ID]
The enumeration identifier is not valid or cannot be used with the specified handle.

[CMSERR_ATTRIBUTE_INVALID_ENUMERATION]
The enumeration value is not valid or cannot be used with the specified enumeration ID.

[CMSERR_BAD_HANDLE]
The handle is not valid.

Usage

The **gsk_set_directory_enum()** routine sets the enumerated value for an LDAP directory vector. The LDAP directory must have a valid LDAP handle as initialized using **gsk_open_directory()**

These enumeration identifiers are supported:

GSKCMS_CRL_CACHE_TEMP_CRL
Specifies if a temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not exist on the LDAP server:

GSKCMS_CRL_CACHE_TEMP_CRL_ON
A temporary CRL entry is added. This is the default setting.

GSKCMS_CRL_CACHE_TEMP_CRL_OFF
A temporary CRL entry is not added.

GSKCMS_CRL_SECURITY_LEVEL
Specifies the level of security to be used when contacting an LDAP server in order to check for revoked certificates in a Certificate Revocation List (CRL). CRLs located will be cached according to the GSK_CRL_CACHE_TIMEOUT setting of the SSL environment. To enforce contact with the LDAP server for each CRL check, CRL caching must be disabled. See "gsk_attribute_set_numeric_value()" on page 124 and Appendix A, "Environment variables," on page 763 for additional information about the GSK_CRL_CACHE_TIMEOUT setting.

Three levels of security are available:

GSKCMS_CRL_SECURITY_LEVEL_LOW
Certificate validation will not fail if the LDAP server cannot be contacted.

GSKCMS_CRL_SECURITY_LEVEL_MEDIUM
Certificate validation requires the LDAP server to be contactable, but does not require a CRL to be defined. This is the default setting.

GSKCMS_CRL_SECURITY_LEVEL_HIGH
Certificate validation requires the LDAP server to be contactable and a CRL to be defined.

gsk_set_directory_numeric_value()

Sets an integer value for an LDAP directory.

Format

```
#include <gskcms.h>

gsk_status gsk_set_directory_numeric_value (
                                gsk_handle                  directory_handle,
                                GSKCMS_DIRECTORY_NUM_ID     num_id,
                                int                         num_value)
```

Parameters

directory_handle
Specifies an LDAP directory handle returned by **gsk_open_directory()**.

num_id
Specifies the directory numeric identifier.

num_value
Specifies the directory integer value.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ATTRIBUTE_INVALID_ID]
The numeric identifier is not valid or cannot be used with the specified handle.

[CMSERR_BAD_HANDLE]
The handle is not valid.

[CMSERR_INVALID_NUMERIC_VALUE]
The numeric value is not valid.

Usage

The **gsk_set_directory_numeric_value()** routine sets an integer value for an LDAP directory. The LDAP directory must have a valid LDAP handle as initialized using **gsk_open_directory()**.

These numeric identifiers are supported:

GSKCMS_CRL_CACHE_ENTRY_MAXSIZE
Specifies the maximum size in bytes of a CRL that is allowed to be stored in the LDAP CRL cache. Any CRLs larger than this size are not cached. The default is 0, which means there is no limit on the size of the CRL stored in the LDAP CRL cache.

GSKCMS_CRL_CACHE_SIZE
Specifies the maximum number of CRLs that are allowed to be stored in the LDAP CRL cache. The valid cache sizes are -1 through 32000. Specify -1 if the LDAP CRL cache size is to be unlimited. Caching only occurs if the *crl_cache_timeout* specified on the **gsk_open_directory()** call is not 0. The default is unlimited.

GSKCMS_LDAP_RESPONSE_TIMEOUT
Specifies the time in seconds to wait for a response from the LDAP server. The default is 30 seconds. A value of 0 indicates that there is no time limit.

gsk_sign_certificate()

Signs an X.509 certificate.

Format

```
#include <gskcms.h>

gsk_status gsk_sign_certificate (
                                 x509_certificate *              certificate,
                                 pkcs_private_key_info *         private_key)
```

Parameters

certificate
Specifies the X.509 certificate.

private_key
Specifies the private key.

Results

The return status will be zero if the signature is successfully generated. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
Private key does not exist or is not accessible.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_MASK_ALG_NOT_SUPPORTED]
RSASSA-PSS mask generation algorithm is not supported.

Usage

The **gsk_sign_certificate()** routine will sign an X.509 certificate using the supplied private key. The private key can be an RSA key, DSA key, or an ECDSA key. If executing in FIPS mode, the minimum key size for RSA and DSA keys is 1024 bits, and the minimum key size for ECDSA keys is 192 bits. Ensure that the keys and the algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, "System SSL and FIPS 140-2," on page 19. The private key can be an ASN.1-encoded value contained in the privateKey field or an ICSF key label contained in the keyToken field. In either case, the key type must be specified by the privateKeyAlgorithm field.

The signature algorithm is obtained from the signature field of the x509_tbs_certificate structure contained within the x509_certificate structure. The generated signature will be placed in the signatureAlgorithm and signatureValue fields of the x509_certificate structure.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest – {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest – {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest – {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest – {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest – {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest – {1.2.840.10045.4.3.4}

These RSASSA-PSS [1.2.840.113549.1.1.10] signatures are supported:

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation algorithm 1.

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

gsk_sign_crl()

Signs an X.509 certificate revocation list.

Format

```
#include <gskcms.h>

gsk_status gsk_sign_crl (
                             x509_crl *                        crl,
                             pkcs_private_key_info *           private_key)
```

Parameters

crl
Specifies the X.509 certificate revocation list.

private_key
Specifies the private key.

Results

The return status will be zero if the signature is successfully generated. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]
Encryption key size is not supported.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_RSA_PRIVATE_KEY_BAD_TYPE]
PKDS RSA private key type not valid for RSASSA-PSS signature generation.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PRIVATE_KEY]
Private key does not exist or is not accessible.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_MASK_ALG_NOT_SUPPORTED]
RSASSA-PSS mask generation algorithm is not supported.

Usage

The **gsk_sign_crl()** routine will sign an X.509 certificate revocation list using the supplied private key. The private key can be an RSA key, a DSA key, or an ECDSA key. If executing in FIPS mode, the minimum key size for RSA and DSA keys is 1024 bits, and the minimum key size for ECDSA keys is 192 bits. Ensure that the keys and the algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19. The private key can be an ASN.1-encoded value contained in the privateKey field or an ICSF key label contained in the keyToken field. In either case, the key type must be specified by the privateKeyAlgorithm field.

The signature algorithm is obtained from the signature field of the x509_tbs_crl structure contained within the x509_crl structure. The generated signature will be placed in the signatureAlgorithm and signatureValue fields of the x509_crl structure.

If the signature from the signature field of the x509_tbs_crl structure is type x509_alg_rsaSsaPss, the signature algorithms supported are:

- x509_alg_mgf1Sha256WithRsaSsaPss
- x509_alg_mgf1Sha384WithRsaSsaPss
- x509_alg_mgf1Sha512WithRsaSsaPss

Only RSA encryption private_key sizes 2048 through 4096 inclusive are supported.

If the RSA private key is in the PKDS, see “RSASSA-PSS signature support” on page 14 for information concerning required hardware support.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation algorithm 1.

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest – {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest – {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest – {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest – {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest – {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest – {1.2.840.10045.4.3.4}

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

gsk_sign_data()

Signs a data stream.

Format

```
#include <gskcms.h>

gsk_status gsk_sign_data (
                         x509_algorithm_type              sign_algorithm,
                         pkcs_private_key_info *          private_key,
                         gsk_boolean                      is_digest,
                         gsk_buffer *                     data,
                         gsk_buffer *                     signature)
```

Parameters

sign_algorithm
: Specifies the signature algorithm.

private_key
: Specifies the private key.

is_digest
: Specify TRUE if the data stream digest has been computed or FALSE if the data stream digest needs to be computed.

data
: Specifies either the data stream digest (*is_digest* is TRUE) or the data stream (*is_digest* is FALSE).

signature
: Returns the generated signature. The caller should release the signature buffer when it is no longer needed by calling the **gsk_free_buffer()** routine.

Results

The return status will be zero if the signature is successfully generated. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
: The signature algorithm is not supported.

[CMSERR_BAD_DIGEST_SIZE]
: The digest size is not correct.

[CMSERR_BAD_EC_PARAMS]
: Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]
: Encryption key size is not supported.

[CMSERR_BAD_RNG_OUTPUT]
: In FIPS mode, random bytes generation produced duplicate output.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
: Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
: Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
: ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
: ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_RSA_PRIVATE_KEY_BAD_TYPE]
PKDS RSA private key type not valid for RSASSA-PSS signature generation.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INVALID_KEY_ATTRIBUTE]
Key does not have required PKCS #11 attributes to perform signing.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NO_PKCS11_KEY_LABEL]
A TKDS secure key label is either invalid or missing.

[CMSERR_NO_PRIVATE_KEY]
Private key does not exist, is not accessible, or the PKCS #11 TKDS secure key is not a supported algorithm type.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_MASK_ALG_NOT_SUPPORTED]
RSASSA-PSS mask generation algorithm is not supported.

Usage

The **gsk_sign_data()** routine will generate the signature for a data stream using the supplied private key. The private key can be an RSA key, a DSA key, or an ECDSA key. If executing in FIPS mode, the minimum key size for RSA and DSA keys is 1024 bits, and the minimum size for ECDSA keys is 192 bits. Ensure that the keys and the algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19. The private key can be an ASN.1-encoded value contained in the privateKey field or an ICSF key label contained in the keyToken field. In either case, the key type must be specified by the privateKeyAlgorithm field.

The application can either provide the message digest or have the **gsk_sign_data()** routine compute the message digest.

When the application provides the message digest, the digest length must be correct for the specified signature algorithm. Digest lengths: MD2 and MD5 are 16 bytes; SHA-1 is 20 bytes; SHA-224 is 28 bytes; SHA-256 is 32 bytes; SHA-384 is 48 bytes and SHA-512 is 64 bytes. The supplied digest will be used as-is without any further processing (specifically, for an RSA encryption key, the digest will not be encoded as an ASN.1 DigestInfo sequence before generating the signature).

When utilizing an RSASSA-PSS signature algorithm:

- Only RSA encryption *private_key* sizes 2048 through 4096 inclusive are supported. If *if_digest* is TRUE, the digest lengths are: SHA-256 is 32 bytes, SHA-384 is 48 bytes, and SHA-512 is 64 bytes. The salt lengths for the corresponding *sign_algorithms* are: SHA-256 is 32 bytes, SHA-384 is 48 bytes, and SHA-512 is 64 bytes.
- If the RSA private key is in the PKDS, see “RSASSA-PSS signature support” on page 14 for information about required hardware support.

If authenticated attributes are provided from the *attributes_signers* parameter, then signing certificates for all signers represented within the gsk_attributes_signers structure must be provided from the *signer_certificates* parameter.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation algorithm 1.

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest – {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest – {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest – {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest – {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest – {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest – {1.2.840.10045.4.3.4}

x509_alg_md5Sha1WithRsaEncryption
RSA encryption with combined MD5 and SHA-1 digests

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

gsk_validate_certificate()

Validates an X.509 certificate.

This function is deprecated. Use **gsk_validate_certificate_mode()** instead.

Format

```
#include <gskcms.h>

gsk_status gsk_validate_certificate (
                       gskdb_data_sources *        data_sources,
                       x509_certificate *          subject_certificate,
                       gsk_boolean                 accept_root,
                       gsk_int32 *                 issuer_record_id)
```

Parameters

data_sources
Specifies the data sources for CA certificates and revocation lists. The data sources are searched in the order they occur in the data source array, so trusted sources should be included before untrusted sources and local sources should be included before remote sources.

subject_certificate
Specifies the certificate to be validated.

accept_root
Specify TRUE if a self-signed root certificate is to be accepted without checking the data sources. Specify FALSE if a self-signed root certificate must be found in one of the trusted data sources in order to be accepted.

issuer_record_id
Returns the record identifier for the issuer certificate used to validate the certificate. The record identifier will be 0 if the issuer certificate is found in a non-database source. Specify NULL for this parameter if the issuer record identifier is not needed.

Results

The return status will be zero if the validation is successful. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_HTTP_RESPONSE]
HTTP response is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_ISSUER_NAME]
The certificate issuer name is not valid.

[CMSERR_BAD_OCSP_RESPONSE]
OCSP response is not valid.

[CMSERR_BAD_SIGNATURE]
The signature is not correct.

[CMSERR_CERT_CHAIN_NOT_TRUST]
The certification chain is not trusted

[CMSERR_CERTIFICATE_REVOKED]
The certificate is revoked.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_EXPIRED]
The certificate is expired.

[CMSERR_HOSTNAME_NOT_VALID]
HTTP server hostname is not valid.

[CMSERR_HTTP_IO_ERROR]
HTTP server communication error.

[CMSERR_HTTP_RESPONSE_TIMEOUT]
HTTP response not received within the configured time limit.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The issuer certificate does not allow signing certificates

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_ISSUER_NOT_CA]
The certificate issuer is not a certification authority.

[CMSERR_ISSUER_NOT_FOUND]
The issuer certificate is not found in one of the data sources.

[CMSERR_LDAP_RESPONSE_TIMEOUT]
LDAP response not received within configured time limit.

[CMSERR_MAX_RESPONSE_SIZE_EXCEEDED]
The maximum response size has been exceeded.

[CMSERR_NAME_CONSTRAINTS_VIOLATED]
The certificate name is not consistent with the name constraints.

[CMSERR_NAME_NOT_SUPPORTED]
The AuthorityKeyIdentifier extension name is not a directory name.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NOT_YET_VALID]
The certificate is not yet valid.

[CMSERR_OCSP_NONCE_CHECK_FAILED]
Nonce in OCSP response does not match value in OCSP request.

[CMSERR_OCSP_RESPONDER_ERROR]
OCSP request failed with internal responder error.

[CMSERR_OCSP_RESPONSE_TIMEOUT]
OCSP response was not received within the configured time limit.

[CMSERR_OCSP_REQ_ERROR]
Error creating the OCSP request.

[CMSERR_OCSP_SIG_REQUIRED]
OCSP responder requires a signed request.

[CMSERR_OCSP_EXPIRED]
The OCSP response has expired.

[CMSERR_PATH_TOO_LONG]
The certification chain exceeds the maximum allowed by the CA.

[CMSERR_REQUIRED_BC_EXT_MISSING]
The required basic constraints certificate extension is missing.

[CMSERR_REVINFO_NOT_YET_VALID]
Revocation information is not yet valid.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_MASK_ALG_NOT_SUPPORTED]
RSASSA-PSS mask generation algorithm is not supported.

[CMSERR_SELF_SIGNED_NOT_FOUND]
A self-signed certificate is not found in a trusted data source

[CMSERR_UNKNOWN_ERROR]
An unknown error has occurred.

Usage

The **gsk_validate_certificate()** routine validates an X.509 certificate by performing these checks on the subject certificate:

- The certificate subject name must be either a non-empty distinguished name or an empty distinguished name with a SubjectAltName certificate extension
- An empty subject name is not allowed for a CA certificate
- The certificate issuer name must not be an empty distinguished name
- The CertificatePolicy extension, if present, must not be a critical extension
- The current time must not be earlier than the start of the certificate validity period
- The current time must not be later than the end of the certificate validity period
- The issuer certificate must be a valid CA certificate
- The certificate signature must be correct
- The certificate must not be revoked
- The certification chain must lead to a certificate obtained from a trusted data source
- No certificate in the certification chain can be revoked or expired.

If executing in FIPS mode, only FIPS-approved algorithms and key sizes are supported (see Chapter 4, “System SSL and FIPS 140-2,” on page 19 for more details).

The **gsk_validate_certificate()** routine will obtain any necessary CA certificates from the supplied data sources. The CA certificate will be validated as described if it is obtained from an untrusted data source. In addition, these checks will be performed on CA certificates when validating the certification chain:

- The BasicConstraints extension, if present, must have the CA indicator set and the path length constraint must not be violated by subordinate certificates in the certification chain
- The NameConstraints extension, if present, must not be violated by the subject certificate

A root certificate is a self-signed certificate and its signature is verified using the public key in the certificate. If *accept_root* is FALSE, the root certificate must be found in a trusted data source in order to be accepted. If *accept_root* is TRUE, the self-signed certificate is accepted if the signature is correct.

An intermediate certificate or an end-entity certificate is a certificate signed by another entity. Its signature is verified using the public key in the issuer's certificate. The issuer certificate must be found in one of the supplied data sources. When intermediate CA certificates are used, the certificate chain is validated until the root certificate for the chain is found in one of the trusted data sources. If a sole intermediate certificate is found in a SAF key ring and the next issuer is not found in the same SAF key ring, the intermediate certificate will be allowed to act as a trust anchor and the chain will be considered complete. It is strongly recommended that a SAF key ring containing an intermediate certificate also has the rest of the certificate chain connected to the key ring, including the root certificate.

The data sources must contain at least one of the following sources to check for revoked certificates: LDAP directory, LDAP extended directory, HTTP CDP, OCSP, or a CRL source. The LDAP, HTTP CDP, and OCSP data sources are untrusted. A CRL data source can be either trusted or untrusted.

If using an LDAP directory or LDAP extended directory source, the CRL distribution point name (or the certificate issuer name if the certificate does not have a CrlDistributionPoints extension) is used as the distinguished name of the LDAP directory entry containing the certificate revocation list (CRL). The CRL distribution point name and CRL issuer name must be X.500 directory names.

The BasicConstraints certificate extension determines whether the CA revocation list or the user revocation list is used. An error is returned if a CRL obtained from an untrusted source cannot be validated.

If using an HTTP CDP data source, the CRL distribution point name indicates a Universal Resource Indicator (URI), which indicates the HTTP server to contact for the CRL.

If using an OCSP data source, the AIA extension in the certificate or the *ocsp_url* parameter specified in the *gskdb_ocsp_source* structure specifies the URI of the OCSP responder to contact.

An LDAP extended directory, HTTP CDP, and OCSP data source is created by calling the **gsk_create_revocation_source()** routine and filling in the *gskdb_source* structure and the embedded specific data source structure. See **gsk_create_revocation_source()** for more information about the creating the data sources.

Security levels for connecting to LDAP directories are based on the GSKCMS_CRL_SECURITY_LEVEL setting. When using the **gsk_open_directory ()** routine, the GSKCMS_CRL_SECURITY_LEVEL setting can be specified using the **gsk_set_directory_enum()** routine. When using the **gsk_create_revocation_source()** routine to create an extended LDAP directory source, the security setting can be set in the *crlSecurityLevel* parameter in the *gskdb_extended_directory_source* structure within the gskdb_source structure. Security levels can be set to LOW, MEDIUM or HIGH. See "gsk_attribute_set_enum()" on page 112 for more information about CRL security level settings. See "gsk_create_revocation_source()" on page 249 for more information about the creation of the extended LDAP directory data source.

By default, the revocation security level for OCSP and HTTP CDP revocation sources is set to GSKCMS_REVOCATION_SECURITY_LEVEL_MEDIUM. See "gsk_validate_certificate_mode()" on page 491 for information on the *security_level*.

By default, the maximum number of locations that will be contacted per data source when attempting validation of a certificate containing AIA or CDP extensions is 10. See "gsk_validate_certificate_mode()" on page 491 for information on the *max_source_rev_ext_loc_values*.

By default, the maximum number of HTTP URI values that will be contacted when performing validation of a certificate chain containing AIA or CDP extensions is 10. See "gsk_validate_certificate_mode()" on page 491 for information on the *max_validation_rev_ext_loc_values*.

These data sources are supported:

- gskdb_source_key_database - The source is a key database. The handle must be a database handle returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine. This is a trusted data source.
- gskdb_source_directory - The source is an LDAP directory. The handle must be the directory handle returned by the **gsk_open_directory()** routine. This is an untrusted data source. Any certificate or revocation list obtained from this source will be validated before being accepted.

See the **gsk_get_directory_certificates()** and **gsk_get_directory_crls()** routines for more information concerning the use of LDAP directory entries.

- gskdb_source_directory_extended - The source is an LDAP directory. The handle must be the directory handle returned by the **gsk_create_revocation_source()** routine. This is an untrusted data source. Any certificate or revocation list obtained from this source is validated before being accepted.
  See the **gsk_get_directory_certificates()** and **gsk_get_directory_crls()** routines for more information concerning the use of LDAP directory entries.
- gskdb_source_ocsp - The source is an OCSP responder. The handle must be the handle returned by the **gsk_create_revocation_source()** routine. This is an untrusted data source. The revocation information obtained from this source is validated before being accepted.
- gskdb_source_cdp - The source is an CDP. The handle must be the handle returned by the **gsk_create_revocation_source()** routine. This is an untrusted data source. The CRL obtained from this source is validated before being accepted.
- gskdb_source_trusted_certs - The source is an array of certificates. This is a trusted data source.
- gskdb_source_untrusted_certs - The source is an array of certificates. This is an untrusted data source. Any certificate used from this list will be validated before being accepted.
- gskdb_source_trusted_crls - The source is an array of certificate revocation lists. This is a trusted data source.
- gskdb_source_untrusted_crls - The source is an array of certificate revocation lists. This is an untrusted data source. Any CRL used from this list will be validated before being accepted.
- gskdb_source_cert_callback - The source is the address of a callback routine which will receive control when an issuer certificate is needed. This is a trusted data source. The subject name is passed as an input parameter and the certCallback routine returns an array of one or more certificates with that subject name. The **gsk_validate_certificate()** routine will call the freeCallback routine to release the certificates. The return status should be 0 if no errors are detected. Otherwise it should be one of the error code listed in the **gskcms.h** include file. The return status should be 0 and the certificate count should be 0 if there are no certificates matching the supplied subject name.
- gskdb_source_crl_callback - The source is the address of a callback routine which will receive control when a certificate needs to be checked to see if it has been revoked. This is a trusted source. The return value should be 0 if the certificate is not revoked. If the callback routine is unable to check the certificate for revocation and processing should continue to the next data source, the return value should be -1. Otherwise it should be one of the error codes defined in the **gskcms.h** include file.

gsk_validate_certificate_mode()

Validates an X.509 certificate.

Format

```
#include <gskcms.h>

gsk_status gsk_validate_certificate_mode (
            gskdb_data_sources *                data_sources,
            x509_certificate *                  subject_certificate,
            gsk_boolean                         accept_root,
            gsk_int32 *                         issuer_record_id,
            GSKCMS_CERT_VALIDATION_MODE         validation_mode,
            gsk_uint32                          arg_count
            [,GSKCMS_CERT_VALIDATE_KEYRING_ROOT validate_root]
            [,GSKCMS_REVOCATION_SECURITY_LEVEL  security_level]
            [,gsk_int32                         max_source_rev_ext_loc_values]
            [,gsk_int32                         max_validation_rev_ext_loc_values]
            [,x509_diag_summary *               diag_summary]
            ...)
```

Parameters

data_sources
Specifies the data sources for CA certificates and revocation lists. The data sources are searched in the order they occur in the data source array, so trusted sources should be included before untrusted sources and local sources should be included before remote sources.

subject_certificate
Specifies the certificate to be validated.

accept_root
Specify TRUE if a self-signed root certificate is to be accepted without checking the data sources. Specify FALSE if a self-signed root certificate must be found in one of the trusted data sources to be accepted.

issuer_record_id
Returns the record identifier for the issuer certificate that is used to validate the certificate. The record identifier is 0 if the issuer certificate is found in a non-database source. Specify NULL for this parameter if the issuer record identifier is not needed.

validation_mode
Specifies certificate validation mode to customize the policy that is used for certificate validation.

arg_count
Specifies the number of optional parameters following the *arg_count* parameter. The *arg_count* parameter can be set to either 0, 1, 2, 3, or 4. If set to 1, the *validate_root* parameter must be specified. If set to 2, the *validate_root* and *security_level* parameters must be specified. If set to 3, the *validate_root*, *security_level*, and *max_source_rev_ext_loc_values* parameters must be specified. If set to 4, the *validate_root*, *security_level*, *max_source_rev_ext_loc_values*, and *max_validation_rev_ext_loc_values* parameters must be specified.

validate_root
Specifies how certificates in a SAF key ring are validated. Specify GSKCMS_CERT_VALIDATE_KEYRING_ROOT_ON if SAF key ring certificates are validated to the root CA certificate. Specify GSKCMS_CERT_VALIDATE_KEYRING_ROOT_OFF if SAF key ring certificates are validated only to the trust anchor certificate when a sole intermediate certificate exists in the SAF key ring. By default, SAF key ring certificates are only validated to the trust anchor certificate. This setting does not affect the validation of SSL key database file and PKCS #11 token certificates as these certificates are always validated to the root CA certificate.

security_level

Specifies the behavior when an OCSP responder or an HTTP server can not be contacted. Specify GSKCMS_REVOCATION_SECURITY_LEVEL_LOW if certificate validation does not fail if the OCSP responder or HTTP server cannot be contacted. Specify GSKCMS_REVOCATION_SECURITY_LEVEL_MEDIUM if certificate validation requires the OCSP responder or the HTTP server in a URI value in the CDP extension to be contactable. For an OCSP responder, it must be able to provide a good certificate revocation status. If the certificate status is revoked or unknown, certificate validation fails. For an HTTP server in a CDP extension, it must be contactable and able to provide an CRL that is encoded in a valid manner. If both an OCSP and a CDP data source are specified, both are attempted to be contacted. If they both fail, certificate validation fails. This parameter is only used when the *data_sources* parameter includes at least one CDP or OCSP data source created by **gsk_create_revocation_source()**. Specify GSKCMS_REVOCATION_SECURITY_LEVEL_HIGH if certificate validation requires revocation information to be provided by the OCSP responder or HTTP server.

If the *security_level* parameter is not specified, GSKCMS_REVOCATION_SECURITY_LEVEL_MEDIUM is used.

Note: The security level for LDAP data sources is set within the data source itself.

max_source_rev_ext_loc_values

Specifies the maximum number of locations that will be contacted per data source when attempting validation of a certificate. The locations for revocation information are specified by the accessLocation in the AIA certificate extension for OCSP and the distributionPoint in the CDP extension for HTTP CRLs. When an HTTP URI is present in an AIA or CDP extension, an attempt is made to contact the remote HTTP server to obtain revocation information. Both of these extensions can contain multiple location values and therefore have the potential to negatively impact performance when there are a very large number of locations present.

The valid values are 0 through 256. A value of 0 indicates there is no limit.

If the *max_source_rev_ext_loc_values* parameter is not specified, a value of 10 is used.

max_validation_rev_ext_loc_values

Specifies the maximum number of HTTP URI values that will be contacted when performing validation of a certificate chain. The locations for revocation information are specified by the accessLocation in the AIA certificate extension for OCSP and the distributionPoint in the CDP extension for HTTP CRLs. When an HTTP URI is present in an AIA or CDP extension, an attempt is made to contact the remote HTTP server to obtain revocation information. Both of these extensions can contain multiple location values and therefore have the potential to negatively impact performance when there are a very large number of locations present.

The valid values are 0 through 1024. A value of 0 indicates there is no limit.

If the *max_validation_rev_ext_loc_values* parameter is not specified, a value of 100 is used.

diag_summary

Specifies that diagnostic information of the certificate chain should be collected and stored in the provided x509_diagnostic_summary structure during the validation process.

If the *diag_summary* parameter is NULL or not specified, the diagnostic information of the certificate validation process will not be collected.

Results

The return status is zero if the validation is successful. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]

The signature algorithm is not supported.

[CMSERR_BAD_ARG_COUNT]

Variable argument count is not valid.

[CMSERR_BAD_CRL]
Certificate revocation list cannot be found.

[CMSERR_BAD_EXT_DATA]
Certificate extension data is incorrect.

[CMSERR_BAD_HANDLE]
The database handle is not valid.

[CMSERR_BAD_HTTP_RESPONSE]
HTTP response is not valid.

[CMSERR_BAD_KEY_SIZE]
The key size is not valid.

[CMSERR_BAD_ISSUER_NAME]
The certificate issuer name is not valid.

[CMSERR_BAD_OCSP_RESPONSE]
OCSP response is not valid.

[CMSERR_BAD_SECURITY_LEVEL_ARG]
Variable argument security level is not valid.

[CMSERR_BAD_SIGNATURE]
The signature is not correct.

[CMSERR_BAD_SUBJECT_NAME]
Subject name is not valid.

[CMSERR_BAD_VALIDATE_ROOT_ARG]
Variable argument validate root is not valid.

[CMSERR_BAD_VALIDATION_OPTION]
Validation option is not valid.

[CMSERR_CERT_CHAIN_NOT_TRUSTED]
The certification chain is not trusted.

[CMSERR_CERTIFICATE_REVOKED]
The certificate is revoked.

[CMSERR_CRITICAL_EXT_INCORRECT]
Certificate extension has an incorrect critical indicator.

[CMSERR_DISTRIBUTION_POINTS]
Cannot match CRL distribution points.

[CMSERR_DUPLICATE_EXTENSION]
Supplied extensions contain a duplicate extension.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_EXPIRED]
The certificate is expired.

[CMSERR_EXT_NOT_SUPPORTED]
Certificate extension is not supported.

[CMSERR_HOSTNAME_NOT_VALID]
HTTP server hostname is not valid.

[CMSERR_HTTP_IO_ERROR]
HTTP server communication error.

[CMSERR_HTTP_RESPONSE_TIMEOUT]
HTTP response not received within the configured time limit.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_INCORRECT_DBTYPE]
The database type does not support certificates.

[CMSERR_INCORRECT_KEY_USAGE]
The issuer certificate does not allow signing certificates

[CMSERR_INTERNAL_ERROR]
An internal processing error has occurred.

[CMSERR_INVALID_NUMERIC_VALUE]
The variable argument for *max_source_rev_ext_loc_values* or *max_validation_rev_ext_loc_values* is not valid.

[CMSERR_ISSUER_NOT_CA]
The certificate issuer is not a certification authority.

[CMSERR_ISSUER_NOT_FOUND]
The issuer certificate is not found in one of the data sources.

[CMSERR_LDAP_NOT_AVAILABLE]
LDAP is not available.

[CMSERR_LDAP_RESPONSE_TIMEOUT]
LDAP response not received within configured time limit.

[CMSERR_MAX_RESPONSE_SIZE_EXCEEDED]
The maximum response size has been exceeded.

[CMSERR_MAX_REV_EXT_LOC_VALUES_EXCEEDED]
The maximum number of locations allowed to be contacted has been reached.

[CMSERR_NAME_CONSTRAINTS_VIOLATED]
The certificate name is not consistent with the name constraints.

[CMSERR_NAME_NOT_SUPPORTED]
The AuthorityKeyIdentifier extension name is not a directory name.

[CMSERR_NO_ACCEPTABLE_POLICIES]
Acceptable policy intersection cannot be found.

[CMSERR_NO_MEMORY]
Insufficient storage is available.

[CMSERR_NOT_YET_VALID]
The certificate is not yet valid.

[CMSERR_OCSP_NONCE_CHECK_FAILED]
Nonce in OCSP response does not match value in OCSP request.

[CMSERR_OCSP_RESPONDER_ERROR]
OCSP request failed with internal responder error.

[CMSERR_OCSP_RESPONSE_TIMEOUT]
OCSP response was not received within the configured time limit.

[CMSERR_OCSP_REQ_ERROR]
Error creating the OCSP request.

[CMSERR_OCSP_SIG_REQUIRED]
OCSP responder requires a signed request.

[CMSERR_OCSP_EXPIRED]
The OCSP response has expired.

[CMSERR_PATH_TOO_LONG]
The certification chain exceeds the maximum that is allowed by the CA.

[CMSERR_RECORD_NOT_FOUND]
Record not found.

[CMSERR_REQUIRED_BC_EXT_MISSING]
The required basic constraints certificate extension is missing.

[CMSERR_REQUIRED_EXT_MISSING]
Required certificate extension is missing.

[CMSERR_REVINFO_NOT_YET_VALID]
Revocation information is not yet valid.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_MASK_ALG_NOT_SUPPORTED]
RSASSA-PSS mask generation algorithm is not supported.

[CMSERR_UNKNOWN_ERROR]
An unknown error has occurred.

Usage

The **gsk_validate_certificate_mode()** routine validates an X.509 certificate according to the standards defined in RFC 2459 (tools.ietf.org/html/rfc2459), RFC 3280 (tools.ietf.org/html/rfc3280), or RFC 5280 (tools.ietf.org/html/rfc5280). Any necessary CA or issuer certificates are obtained from the supplied data sources. The CA certificate is also validated according to the previously mentioned Internet standards.

The *validation_mode* parameter determines the Internet standard that the certificate and certificate chain are validated against. The following validation modes are supported:

- GSKCMS_CERT_VALIDATION_MODE_2459 – validate the certificate against RFC 2459 only.
- GSKCMS_CERT_VALIDATION_MODE_3280 – validate the certificate against RFC 3280 only.
- GSKCMS_CERT_VALIDATION_MODE_5280 – validate the certificate against RFC 5280 only.
- GSKCMS_CERT_VALIDATION_MODE_ANY – attempt to validate the certificate against RFC 2459 initially. If that fails, validate against RFC 3280. If that fails, validate against RFC 5280.

Note: The z/OS specific HostIDMapping certificate extension is supported by System SSL and can be validated as a critical extension in any validation mode.

A root certificate is a self-signed certificate and its signature is verified by using the public key in the certificate. If *accept_root* is FALSE, the root certificate must be found in a trusted data source to be accepted. If *accept_root* is TRUE, the self-signed certificate is accepted if the signature is correct.

An intermediate certificate or an end-entity certificate is a certificate that is signed by another entity. Its signature is verified by using the public key in the issuer's certificate. The issuer certificate must be found in one of the supplied data sources. When intermediate CA certificates are used, the certificate chain is validated until the root certificate for the chain is found in one of the trusted data sources. If a sole intermediate certificate is found in a SAF key ring and the next issuer is not found in the same SAF key ring, and *validate_root* is not specified or is set to GSKCMS_CERT_VALIDATE_KEYRING_ROOT_OFF, the intermediate certificate is allowed to act as a trust anchor, and the chain is considered complete. By default, SAF key ring certificates are only validated to the trust anchor certificate. If *validate_root* is set to GSKCMS_CERT_VALIDATE_KEYRING_ROOT_ON, an intermediate certificate in a SAF key ring is not allowed to be established as a trust anchor and full certificate validation to the root CA must occur. Make sure that a SAF key ring containing an intermediate certificate also has the rest of the certificate chain that is connected to the key ring, including the root certificate. The *validate_root* setting does not affect the validation of SSL key database file and PKCS #11 token certificates because these certificates are always validated to the root CA certificate.

The data sources must contain at least one of the following sources to check for revoked certificates: LDAP directory, LDAP extended directory, HTTP CDP, OCSP, or a CRL source. The LDAP, HTTP CDP, and OCSP data sources are untrusted. A CRL data source can be either trusted or untrusted.

If using an LDAP directory or LDAP extended directory source, the CRL distribution point name (or the certificate issuer name if the certificate does not have a CrlDistributionPoints extension) is used as the distinguished name of the LDAP directory entry containing the certificate revocation list (CRL). The CRL distribution point name and CRL issuer name must be X.500 directory names.

The BasicConstraints certificate extension determines whether the CA revocation list or the user revocation list is used. An error is returned if a CRL obtained from an untrusted source cannot be validated.

If using an HTTP CDP data source, the CRL distribution point name indicates a Universal Resource Indicator (URI), which indicates the HTTP server to contact for the CRL.

If using an OCSP data source, the AIA extension in the certificate or the *ocsp_url* parameter specified in the *gskdb_ocsp_source* structure specifies the URI of the OCSP responder to contact.

An LDAP extended directory, HTTP CDP, and OCSP data source is created by calling the **gsk_create_revocation_source()** routine and filling in the *gskdb_source* structure and the embedded specific data source structure. See **gsk_create_revocation_source()** for more information about the creating the data sources.

Security levels for connecting to LDAP directories are based on the GSKCMS_CRL_SECURITY_LEVEL setting. When using the **gsk_open_directory()** routine, the GSKCMS_CRL_SECURITY_LEVEL setting can be specified using the **gsk_set_directory_enum()** routine. When using the **gsk_create_revocation_source()** routine to create an extended LDAP directory source, the security setting can be set in the *crlSecurityLevel* parameter in the *gskdb_extended_directory_source* structure within the gskdb_source structure. Security levels can be set to LOW, MEDIUM or HIGH. See "gsk_attribute_set_enum()" on page 112 for more information about CRL security level settings. See "gsk_create_revocation_source()" on page 249 for more information about the creation of the extended LDAP directory data source.

These data sources are supported:

- gskdb_source_key_database - The source is a key database. The handle must be a database handle that is returned by the **gsk_create_database()** routine, the **gsk_open_database()** routine, or the **gsk_open_keyring()** routine. This is a trusted data source.
- gskdb_source_directory - The source is an LDAP directory. The handle must be the directory handle that is returned by the **gsk_open_directory()** routine. This is an untrusted data source. Any certificate or revocation list that is obtained from this source is validated before it is accepted. See the **gsk_get_directory_certificates()** and **gsk_get_directory_crls()** routines for more information about the use of LDAP directory entries.
- gskdb_source_directory_extended - The source is an LDAP directory. The handle must be the directory handle returned by the **gsk_create_revocation_source()** routine. This is an untrusted data source. Any certificate or revocation list obtained from this source is validated before being accepted. See the **gsk_get_directory_certificates()** and **gsk_get_directory_crls()** routines for more information concerning the use of LDAP directory entries.
- gskdb_source_ocsp - The source is an OCSP responder. The handle must be the handle returned by the **gsk_create_revocation_source()** routine. This is an untrusted data source. The revocation information obtained from this source is validated before being accepted.
- gskdb_source_cdp - The source is an CDP. The handle must be the handle returned by the **gsk_create_revocation_source()** routine. This is an untrusted data source. The CRL obtained from this source is validated before being accepted.
- gskdb_source_trusted_certs - The source is an array of certificates. This is a trusted data source.
- gskdb_source_untrusted_certs - The source is an array of certificates. This is an untrusted data source. Any certificate that is used from this list is validated before it is accepted.

- gskdb_source_trusted_crls - The source is an array of certificate revocation lists. This is a trusted data source.
- gskdb_source_untrusted_crls - The source is an array of certificate revocation lists. This is an untrusted data source. Any CRL used from this list is validated before it is accepted.
- gskdb_source_cert_callback - The source is the address of a callback routine that receives control when an issuer certificate is needed. This is a trusted data source. The subject name is passed as an input parameter and the certCallback routine returns an array of one or more certificates with that subject name. The **gsk_validate_certificate_mode()** routine calls the freeCallback routine to release the certificates. The return status should be 0 if no errors are detected. Otherwise, it should be one of the error codes that are listed in the **gskcms.h** include file. The return status should be 0 and the certificate count should be 0 if there are no certificates matching the supplied subject name.
- gskdb_source_crl_callback - The source is the address of a callback routine that receives control when a certificate must be checked to see if it has been revoked. This is a trusted source. The return value should be 0 if the certificate is not revoked. If the callback routine is unable to check the certificate for revocation and processing should continue to the next data source, the return value should be -1. Otherwise, it should be one of the error codes that are defined in the **gskcms.h** include file.

The *validate_root* optional parameter must be specified when *arg_count* is set to 1 or greater. If *validate_root* is not specified and *arg_count* is set to 1 or greater, an error of CMSERR_BAD_VALIDATE_ROOT_ARG is returned.

The *security_level* parameter must be specified when *arg_count* is set to 2 or greater. If *security_level* is not specified and *arg_count* is set to 2 or greater, an error of CMSERR_BAD_SECURITY_LEVEL_ARG is returned.

The *max_source_rev_ext_loc_values* parameter must be specified when *arg_count* is set to 4 or greater. If *max_source_rev_ext_loc_values* is not specified and *arg_count* is set to 4 or greater, an error of CMSERR_INVALID_NUMERIC_VALUE is returned.

A valid x509_diag_summary address or NULL must be specified for the *diag_summary* parameter when *arg_count* is set to 5 or greater. The caller of this routine is responsible for freeing any diagnostic storage obtained from successful or failing validations when the *diag_summary* parameter is provided. This storage is freed through the **gsk_free_x509_diagnostics()** routine.

If the *arg_count* parameter is 0, any additional parameters that are specified are ignored.

Ensure that the keys and the algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. If executing in FIPS mode, only FIPS-approved algorithms and key sizes are supported. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_validate_extended_key_usage()

Validate a certificate's extended key usage extension against the supplied extended key usage list.

Format

```
#include <gskcms.h>

gsk_status gsk_validate_extended_key_usage (
                           x509_certificate *                 certificate,
                           x509_key_usages *                  key_usages,
                           GSKCMS_VALIDATE_EXTENDED_KEY_USAGE   validate_option)
```

Parameters

certificate
Specifies the x.509 certificate to be validated.

key_usages
Specifies a list of x.509 extended key usage purpose types. The count must be at least one. The key purpose type OID field is ignored unless the usage purpose type is set to **GSKCMS_VALIDATE_EXTENDED_KEY_USAGE_CHECK_OID**.

validate_option
Specifies the validation option to customize the validation process:

GSKCMS_VALIDATE_EXTENDED_KEY_USAGE_ALL
All the specified extended key purpose types must be present in the certificate's extended key usage extension.

GSKCMS_VALIDATE_EXTENDED_KEY_USAGE_SOME
At lease one of the specified extended key purpose types must be present within the certificate's extended key usage extension.

GSKCMS_VALIDATE_EXTENDED_KEY_USAGE_NONE
None of the specified extended key purposes may be present within the certificate's extended key usage extension.

Results

The function return value will be 0 if no error is detected. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_BAD_VALIDATION_OPTION]
Validation option is not valid.

[CMSERR_CERTIFICATE_NOT_SUPPLIED]
Input certificate is missing.

[CMSERR_CERT_HAS_NO_EXT_KEY_USAGE_EXTENSION]
Certificate does not have an extended key usage extension.

[CMSERR_EXT_KEY_USAGE_COMPARE_FAILED]
Extended key usage comparison failed.

[CMSERR_EXT_KEY_USAGE_COUNT_IS_INVALID]
Extended key usage count is zero.

[CMSERR_EXT_KEY_USAGE_NOT_SUPPLIED]
Extended key usage or key usage purpose types not provided.

[CMSERR_EXT_KEY_USAGE_TYPE_IS_INVALID]
Extended key usage type is not supported for this operation.

Usage

The **gsk_validate_extended_key_usage()** routine compares the provided certificate's extended key usage extension values against the provided list of extended key purposes. The extended key usage comparison scope is defined by the **GSKCMS_VALIDATE_EXTENDED_KEY_USAGE** value.

If the input certificate does not contain an extended key usage extension, the operation returns **CMSERR_CERT_HAS_NO_EXT_KEY_USAGE_EXTENSION**.

If the *x509_purpose_type* is set to **GSKCMS_VALIDATE_EXTENDED_KEY_USAGE_CHECK_OID**, the *gsk_oid* must be supplied. In this case, the user-supplied OID will be compared to the certificate extended key usage extension value OID. Use this method in instances where a certificate's extended key usage extension type and OID are not defined by System SSL *x509_purpose_type*.

The *x509_purpose_type* types are only checked for presence or absence when being compared to a certificate's extended key usage.

gsk_validate_hostname()

Validates a host certificate against the supplied hostname.

Format

```
#include <gskcms.h>

gsk_status gsk_validate_hostname (
                                x509_certificate *         host_certificate,
                                const char *               host_name,
                                GSKCMS_VALIDATE_HOSTNAME   val_option)
```

Parameters

host_certificate
Specifies the host certificate to be validated.

host_name
Specifies the fully-qualified host name in the local code page.

val_option
Specifies validation option to customize the order of the validation process.

Results

The function return value will be 0 (**GSK_OK**) if the validation is successful. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_HOST_NOT_VALID]
The certificate is not valid for the specified host name.

[CMSERR_BAD_VALIDATION_OPTION]
Validation option is not valid.

Usage

The **gsk_validate_hostname()** routine validates the certificate against the specified host name. For successful validation the certificate must contain the specified host name as either the common name (CN) element of the subject name or as a DNS entry for the subject alternate name as indicated by the validation option. A case-sensitive (exact match) comparison is used for comparison with the common name (CN) element of the subject name when the common name attribute value is encoded as UTF-8 data (x509_string_utf8).

The *val_option* parameter determines the composition and order of the validation process. A value of:

- GSKCMS_VALIDATE_HOSTNAME_CN validates the host name against the common name (CN) of the certificate first and then against the DNS entry for the subject alternate name extension if no match is found in the CN.
- GSKCMS_VALIDATE_HOSTNAME_CN_ONLY validates the host name against the common name (CN) of the certificate only.
- GSKCMS_VALIDATE_HOSTNAME_DNS validates the host name against the DNS entry in the subject alternate name extension first and, only if that is not present, validate the host name against the common name.
- GSKCMS_VALIDATE_HOSTNAME_DNS_ONLY validates the host name against the DNS entry in the subject alternate name extension only.

The host name in the certificate can be a fully-qualified name (for example, 'server1.endicott.mycompany.com'), a domain suffix (for example, '.endicott.mycompany.com') or a wildcard name beginning with an asterisk (for example, '*.endicott.mycompany.com'). A case-sensitive comparison is performed between the supplied host name and the host name in the certificate. A fully-qualified name must be the same as the supplied host name. A domain suffix matches any host name with the same suffix, but does not match the suffix itself. For example, '*.endicott.mycompany.com' matches 'ldap.server1.endicott.mycompany.com' and 'server1.endicott.mycompany.com', but does not match 'endicott.mycompany.com'. A wildcard name matches any name ending with the characters that follow the asterisk. A trailing period in a host name is ignored (for example, 'server1.endicott.mycompany.com.' is the same as 'server1.endicott.mycompany.com').

No other certificate validation is performed. The **gsk_validate_certificate_mode()** routine should be called if the certificate itself must be validated.

gsk_validate_server()

Validate a server certificate.

Format

```
#include <gskcms.h>

gsk_status gsk_validate_server (
                                x509_certificate *       server_certificate,
                                const char *             host_name)
```

Parameters

server_certificate
Specifies the server certificate to be validated.

host_name
Specifies the fully-qualified server host name in the local code page.

Results

The return status is zero if the validation is successful. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_HOST_NOT_VALID]
The server certificate is not valid for the specified host name.

Usage

The **gsk_validate_server()** routine validates a server certificate by verifying the host name that is associated with the server. The server certificate must contain the specified host name as either the common name (CN) element of the subject name or as a DNS entry for the subject alternate name. A case-sensitive (exact match) comparison is used for comparison with the common name (CN) element of the subject name when the common name attribute value is encoded as UTF-8 data (x509_string_utf8). For other combinations of host name verification options use **gsk_validate_hostname()**.

The host name in the server certificate can be a fully-qualified name (for example, 'server1.endicott.mycompany.com'), a domain suffix (for example, '.endicott.mycompany.com') or a wildcard name beginning with an asterisk (for example, '*.endicott.mycompany.com'). A not case-sensitive comparison is performed between the supplied host name and the host name in the server certificate. A fully-qualified name must be the same as the supplied host name. A domain suffix matches any host name with the same suffix, but does not match the suffix itself. For example, '*.endicott.mycompany.com' matches 'ldap.server1.endicott.mycompany.com' and 'server1.endicott.mycompany.com', but does not match 'endicott.mycompany.com'. A wildcard name matches any name ending with the characters that follow the asterisk. A trailing period in a host name is ignored (for example, 'server1.endicott.mycompany.com.' and is the same as 'server1.endicott.mycompany.com').

No other certificate validation is performed. The **gsk_validate_certificate_mode()** routine should be called if the certificate itself must be validated.

gsk_verify_certificate_signature()

Verifies the signature for an X.509 certificate.

Format

```
#include <gskcms.h>

gsk_status gsk_verify_certificate_signature (
                                        x509_certificate *         certificate,
                                        x509_public_key_info *     key)
```

Parameters

certificate
Specifies the decoded certificate returned by the **gsk_decode_certificate()** routine.

key
Specifies the public key for the Certification Authority that signed the certificate.

Results

The return status will be zero if the signature is correct. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]
Encryption key size is not supported.

[CMSERR_BAD_SIGNATURE]
The signature is not correct.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

[CMSERR_PRIVATE_KEY_INFO_NOT_SUPPLIED]
Private key information not supplied.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_MASK_ALG_NOT_SUPPORTED]
RSASSA-PSS mask generation algorithm is not supported.

[CMSERR_SIGNATURE_NOT_SUPPLIED]
Signature not supplied.

Usage

The **gsk_verify_certificate_signature()** routine validates an X.509 certificate by computing its signature and then comparing the result to the signature contained in the certificate.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation algorithm 1.

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest – {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_md5Sha1WithRsaEncryption
RSA encryption with combined MD5 and SHA-1 digests

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest – {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest – {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest – {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest – {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest – {1.2.840.10045.4.3.4}

If the signature being verified is of type RSASSA-PSS, the RSA encryption public key size must be between 2048 through 4096 inclusive.

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_verify_crl_signature()

Verifies the signature for an X.509 certificate revocation list.

Format

```
#include <gskcms.h>

gsk_status gsk_verify_crl_signature (
                                     x509_crl *                      crl,
                                     x509_public_key_info *          key)
```

Parameters

crl
Specifies the decoded certificate revocation list returned by the **gsk_decode_crl()** routine.

key
Specifies the public key for the Certification Authority that signed the certificate revocation list.

Results

The return status will be zero if the signature is correct. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]
Encryption key size is not supported.

[CMSERR_BAD_SIGNATURE]
The signature is not correct.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

[CMSERR_PRIVATE_KEY_INFO_NOT_SUPPLIED]
Private key information not supplied.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_MASK_ALG_NOT_SUPPORTED]
RSASSA-PSS mask generation algorithm is not supported.

[CMSERR_SIGNATURE_NOT_SUPPLIED]
Signature not supplied.

Usage

The **gsk_verify_crl_signature()** routine validates an X.509 certificate revocation list (CRL) by computing its signature and then comparing the result to the signature contained in the CRL.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation algorithm 1.

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest – {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_md5Sha1WithRsaEncryption
RSA encryption with combined MD5 and SHA-1 digests

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest – {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest – {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest – {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest – {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest – {1.2.840.10045.4.3.4}

If the signature being verified is of type RSASSA-PSS, the RSA encryption public key size must be between 2048 through 4096 inclusive.

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

gsk_verify_data_signature()

Verifies the signature for a data stream.

Format

```
#include <gskcms.h>

gsk_status gsk_verify_data_signature (
                                      x509_algorithm_type          sign_algorithm,
                                      x509_public_key_info *       key,
                                      gsk_boolean                  is_digest,
                                      gsk_buffer *                 data,
                                      gsk_buffer *                 signature)
```

Parameters

sign_algorithm
Specifies the signature algorithm.

key
Specifies the public key.

is_digest
Specify TRUE if the data stream digest has been computed or FALSE if the data stream digest needs to be computed.

data
Specifies either the data stream digest (is_digest is TRUE) or the data stream (is_digest is FALSE).

signature
Specifies the data stream signature.

Results

The return status will be zero if the signature is correct. Otherwise, it will be one of the return codes listed in the **gskcms.h** include file. These are some possible errors:

[CMSERR_ALG_NOT_SUPPORTED]
The signature algorithm is not supported.

[CMSERR_BAD_DIGEST_SIZE]
The digest size is not correct.

[CMSERR_BAD_EC_PARAMS]
Elliptic Curve parameters are not valid.

[CMSERR_BAD_KEY_SIZE]
Encryption key size is not supported.

[CMSERR_BAD_SIGNATURE]
The signature is not correct.

[CMSERR_ECURVE_NOT_FIPS_APPROVED]
Elliptic Curve not supported in FIPS mode.

[CMSERR_ECURVE_NOT_SUPPORTED]
Elliptic Curve is not supported.

[CMSERR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[CMSERR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[CMSERR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[CMSERR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[CMSERR_KEY_MISMATCH]
The supplied key does not match the signature algorithm.

[CMSERR_PRIVATE_KEY_INFO_NOT_SUPPLIED]
Private key information not supplied.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_DIGEST_ALG_NOT_SUPPORTED]
RSASSA-PSS digest algorithm is not supported.

[CMSERR_RSASSA_PSS_MASK_ALG_NOT_SUPPORTED]
RSASSA-PSS mask generation algorithm is not supported.

[CMSERR_SIGNATURE_NOT_SUPPLIED]
Signature not supplied.

Usage

The **gsk_verify_data_signature()** routine validates the signature for a data stream. The public key can be an RSA key, a DSA key, or an ECDSA key.

The application can either provide the message digest or have the **gsk_verify_signed_data()** routine compute the message digest.

When the application provides the message digest, the digest length must be correct for the specified signature algorithm. Digest lengths: MD2 and MD5 are 16 bytes; SHA-1 is 20 bytes; SHA-224 is 28 bytes; SHA-256 is 32 bytes; SHA-384 is 48 bytes and SHA-512 is 64 bytes. The supplied digest will be used as-is without any further processing (specifically, for an RSA encryption key, the digest will not be encoded as an ASN.1 DigestInfo sequence before comparing it with the digest in the signature)

If the signature being verified is of type RSASSA-PSS, the RSA encryption public key size must be between 2048 through 4096 inclusive. If *if_digest* is TRUE, the digest lengths are: SHA-256 is 32 bytes, SHA-384 is 48 bytes, and SHA-512 is 64 bytes. The salt lengths for the corresponding *sign_algorithms* are: SHA-256 is 32, SHA-384 is 48, and SHA-512 is 64.

The following signature algorithms are supported:

x509_alg_md2WithRsaEncryption
RSA encryption with MD2 digest - {1.2.840.113549.1.1.2}

x509_alg_md5WithRsaEncryption
RSA encryption with MD5 digest - {1.2.840.113549.1.1.4}

x509_alg_mgf1Sha256WithRsaSsaPss
RSASSA-PSS using SHA-256 digest with mask generation algorithm 1.

x509_alg_mgf1Sha384WithRsaSsaPss
RSASSA-PSS using SHA-384 digest with mask generation algorithm 1.

x509_alg_mgf1Sha512WithRsaSsaPss
RSASSA-PSS using SHA-512 digest with mask generation algorithm 1.

x509_alg_sha1WithRsaEncryption
RSA encryption with SHA-1 digest - {1.2.840.113549.1.1.5}

x509_alg_sha224WithRsaEncryption
RSA encryption with SHA-224 digest - {1.2.840.113549.1.1.14}

x509_alg_sha256WithRsaEncryption
RSA encryption with SHA-256 digest - {1.2.840.113549.1.1.11}

x509_alg_sha384WithRsaEncryption
RSA encryption with SHA-384 digest - {1.2.840.113549.1.1.12}

x509_alg_sha512WithRsaEncryption
RSA encryption with SHA-512 digest - {1.2.840.113549.1.1.13}

x509_alg_dsaWithSha1
Digital Signature Standard with SHA-1 digest - {1.2.840.10040.4.3}

x509_alg_dsaWithSha224
Digital Signature Standard with SHA-224 digest – {2.16.840.1.101.3.4.3.1}

x509_alg_dsaWithSha256
Digital Signature Standard with SHA-256 digest – {2.16.840.1.101.3.4.3.2}

x509_alg_md5Sha1WithRsaEncryption
RSA encryption with combined MD5 and SHA-1 digests

x509_alg_ecdsaWithSha1
Elliptic Curve Digital Signature Algorithm with SHA-1 digest – {1.2.840.10045.4.1}

x509_alg_ecdsaWithSha224
Elliptic Curve Digital Signature Algorithm with SHA-224 digest – {1.2.840.10045.4.3.1}

x509_alg_ecdsaWithSha256
Elliptic Curve Digital Signature Algorithm with SHA-256 digest – {1.2.840.10045.4.3.2}

x509_alg_ecdsaWithSha384
Elliptic Curve Digital Signature Algorithm with SHA-384 digest – {1.2.840.10045.4.3.3}

x509_alg_ecdsaWithSha512
Elliptic Curve Digital Signature Algorithm with SHA-512 digest – {1.2.840.10045.4.3.4}

The x509_alg_md5Sha1WithRsaEncryption algorithm is a special algorithm used by the SSL protocol. The data signature consists of the MD5 digest over the data followed by the SHA-1 digest over the data for a total digest length of 36 bytes. The digest is encrypted as-is without any further processing.

When executing in FIPS mode, signature algorithms x509_alg_md2WithRSAEncryption and x509_alg_md5WithRsaEncryption are not supported.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

Chapter 9. Deprecated Secure Socket Layer (SSL) APIs

These application programming interfaces, or APIs, are superseded by the APIs defined in Chapter 7, “API reference,” on page 73.

- **gsk_free_memory()** (see “gsk_free_memory()” on page 514)
- **gsk_get_cipher_info()** (see “gsk_get_cipher_info()” on page 515)
- **gsk_get_dn_by_label()** (see “gsk_get_dn_by_label()” on page 516)
- **gsk_initialize()** (see “gsk_initialize()” on page 517)
- **gsk_secure_soc_close()** (see “gsk_secure_soc_close()” on page 522)
- **gsk_secure_soc_init()** (see “gsk_secure_soc_init()” on page 523)
- **gsk_secure_soc_read()** (see “gsk_secure_soc_read()” on page 530)
- **gsk_secure_soc_reset()** (see “gsk_secure_soc_reset()” on page 532)
- **gsk_secure_soc_write()** (see “gsk_secure_soc_write()” on page 533)
- **gsk_srb_initialize()** (see “gsk_srb_initialize()” on page 535)
- **GSKSRBRD()** (see “GSKSRBRD” on page 536)
- **GSKSRBWT()** (see “GSKSRBWT” on page 537)
- **gsk_uninitialize()** (see “gsk_uninitialize()” on page 538)
- **gsk_user_set()** (see “gsk_user_set()” on page 539)

Although use of the deprecated set of APIs in this topic is still supported, make sure that any new applications are developed using the set of APIs defined in Chapter 7, “API reference,” on page 73.

The deprecated APIs are not being explicitly updated to allow utilization of new functionality to be added to System SSL. If an application wants to use new functionality to be added, for example TLS V1.2 protocol, the application must be coded to the SSL APIs in Chapter 7, “API reference,” on page 73.

In addition, make sure that existing applications are modified to use the set of APIs defined in Chapter 7, “API reference,” on page 73. Those modified applications should only use the new APIs, and **not** a mix of the new APIs and these deprecated APIs. Information about migrating your existing application programs to use the new API set can be found in Appendix E, “Migrating from deprecated SSL interfaces,” on page 809.

gsk_free_memory()

Releases storage allocated by the SSL run time.

Format

```
#include  <gskssl.h>

void gsk_free_memory(
                      void *     address,
                      void *     reserved)
```

Parameters

address
Specifies the address of the storage to be released.

reserved
Reserved for future use. Specify NULL for this parameter.

Usage

The **gsk_free_memory()** routine releases storage allocated by the SSL run time.

Related topics

- "gsk_get_dn_by_label()" on page 516

gsk_get_cipher_info()

Returns the supported cipher specifications.

Format

```
#include <gskssl.h>

gsk_status gsk_get_cipher_info(
                                int                 level
                                gsk_sec_level *     sec_level,
                                void *              rsvd)
```

Parameters

level
Specifies GSK_LOW_SECURITY to return just the export cipher specifications or GSK_HIGH_SECURITY to return the United States only cipher specifications including the export cipher specifications.

sec_level
Returns the cipher specifications.

rsvd
Reserved for future use. Specify NULL for this parameter.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. This is a possible error:

[GSK_BAD_PARAMETER]
The level value is not valid or a NULL address is specified for sec_level.

Usage

The **gsk_get_cipher_info()** routine returns the available cipher specifications. Both United States only and export ciphers will be included if GSK_HIGH_SECURITY is specified while only export ciphers will be included if GSK_LOW_SECURITY is specified. The **gsk_get_cipher_info()** routine can be called at any time and does not require the **gsk_initialize()** routine to be called first.

The SSL V2 cipher specifications returned for GSK_HIGH_SECURITY are "713642" while the SSL V3 cipher specifications are "050435363738392F303132330A1613100D0915120F0C0306020100" if not in FIPS mode, and "35363738392F303132330A1613100D" in FIPS mode. If the Security SSL Level 3 FMID and CPACF Feature 3863 are not installed, the SSL V2 cipher specifications are "642" and the SSL V3 cipher specifications are "0915120F0C0306020100". If the System SSL Security Level 3 FMID is not installed, FIPS mode is not supported.

The SSL V2 cipher specifications returned for GSK_LOW SECURITY are "642" while the SSL V3 cipher specifications are "0915120F0C0306020100" in non-FIPS mode and "" in FIPS mode.

Related topics

- "gsk_initialize()" on page 517
- "gsk_secure_soc_init()" on page 523

gsk_get_dn_by_label()

Gets the distinguished name for a certificate.

Format

```
#include <gskssl.h>

char * gsk_get_dn_by_label(
                            const char *     label)
```

Parameters

label
Specifies the key label.

Usage

The **gsk_get_dn_by_label()** routine returns the distinguished name for the certificate associated with the key label. The **gsk_initialize()** routine must be called before the **gsk_get_dn_by_label()** routine can be called. The application should release the returned name when it is no longer needed by calling the **gsk_free_memory()** routine. The return value will be NULL if an error occurred while accessing the key database or when using z/OS PKCS #11 token and multiple certificates exist for the specified label.

Related topics

- "gsk_free_memory()" on page 514
- "gsk_initialize()" on page 517
- "gsk_secure_soc_init()" on page 523

gsk_initialize()

Initializes the System SSL runtime environment.

Format

```
#include <gskssl.h>

gsk_status gsk_initialize(
                           gsk_init_data *     init_data)
```

Parameters

init_data
Specifies the data used to initialize the SSL runtime environment.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ERR_INIT_PARM_NOT_VALID]
An initialization parameter is not valid.

[GSK_ERROR_BAD_MALLOC]
Insufficient storage is available.

[GSK_ERROR_CRYPTO]
Cryptographic error detected.

[GSK_ERROR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[GSK_ERROR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[GSK_ERROR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[GSK_ERROR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[GSK_ERROR_LDAP]
Unable to initialize the LDAP client.

[GSK_ERROR_MULTIPLE_LABEL]
Multiple certificates exist for label.

[GSK_ERROR_MULTIPLE_DEFAULT]
Multiple keys are marked as the default.

[GSK_ERROR_PERMISSION_DENIED]
Not authorized to access the key database, PKCS #12 file, key ring or token.

[GSK_INIT_SEC_TYPE_NOT_VALID]
The security type is not valid.

[GSK_INIT_V2_TIMEOUT_NOT_VALID]
The SSL V2 timeout is not valid.

[GSK_INIT_V3_TIMEOUT_NOT_VALID]
The SSL V3 timeout is not valid.

[GSK_ERROR_INTERNAL]
An internal processing error has occurred.

[GSK_KEYFILE_BAD_FORMAT]
Key database, PKCS #12 file, or key ring format is not valid.

[GSK_KEYFILE_BAD_PASSWORD]
Key database or PKCS #12 file password is not correct.

[GSK_KEYFILE_IO_ERROR]
Unable to read the key database, PKCS #12 file, key ring or token.

[GSK_KEYFILE_NO_CERTIFICATES]
The key database, PKCS #12 file, key ring or token does not contain any certificates.

[GSK_KEYFILE_OPEN_FAILED]
Unable to open the key database, PKCS #12 file, key ring or token.

[GSK_KEYFILE_PW_EXPIRED]
Key database password is expired.

Usage

The **gsk_initialize()** routine initializes the System SSL runtime environment for the current process. The **gsk_uninitialize()** routine should be called to release the SSL environment when it is no longer needed. Multiple calls to **gsk_initialize()** causes the existing environment to be released before creating the new environment.

Environment variables are processed along with the **gsk_initialize** data structures. Information that is passed in the key database, key ring or token is read as part of the environment initialization. Upon successful completion of **gsk_initialize()**, the application is ready to begin creating and using secure socket connections.

The gsk_init_data structure contains these fields:

sec_types
Specifies one of these null-terminated character strings:

- "SSLV2" or "SSL20" to use the SSL V2 protocol
- "SSLV3" or "SSL30" to use the SSL V3 protocol
- "TLSV1" or "TLS10" to use the TLS V1.0 protocol
- "SSLV2_OFF" to allow either TLS V1.0 or SSL V3 to be used
- "ALL" to use any supported protocol (SSL V2, SSL V3, and TLS V1.0).

When "SSLV2_OFF" is specified the SSL client/server attempts first to use the TLS V1.0 protocol, before falling back to the most secure protocol supported by its SSL partner, excluding the SSL V2 protocol.

When "ALL" is specified for an SSL client, the client attempts first to use the TLS V1.0 protocol and falls back to the most secure protocol that the server supports, excluding the SSL V2 protocol (the client must explicitly request the SSL V2 protocol if it wants to use this protocol).

When "ALL" is specified for an SSL server, the server accepts any of the supported protocols.

When running in FIPS mode, the minimum requirement is TLS V1.0 protocol. If only the SSL V2 or the SSL V3 protocol is enabled, then a FIPS mode SSL connection is not possible.

keyring
Specifies the name of the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token as a null-terminated character string. When both the password and stash file name are NULL, a SAF key ring or PKCS #11 token is used.

The SAF key ring name is specified as "userid/keyring". The current user ID is used if the user ID is omitted. The user must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by the user. The user must have UPDATE access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by another user.

Thez/OS PKCS #11 token name is specified as *TOKEN*/token-name. *TOKEN* indicates that the specified key ring is actually a token name. The application user ID must have READ access to resource USER.token-name in the CRYPTOZ class in order for the certificate and their private keys, if present, to be read.

keyring_pw
Specifies the password for the key database or PKCS #12 file as a null-terminated character string. Specify NULL to indicate that no password is provided.

keyring_stash
Specifies the name of the password stash file as a null-terminated character string. Specify NULL to indicate no stash file is provided. The password stash file is used if the keyring_pw value is NULL.

V2_session_timeout
Specifies the SSL V2 session cache timeout value in seconds. The valid range is 0 to 100. A short SSL handshake is performed when a cached session exists since the session parameters have already been negotiated between the client and the server.

V3_session_timeout
Specifies the SSL V3 session cache timeout value in seconds. The valid range is 0 to 86400. A short SSL handshake is performed when a cached session exists since the session parameters have already been negotiated between the client and the server.

LDAP_server
Specifies one or more blank-separated LDAP server host names as a null-terminated character string. Each host name can contain an optional port number that is separated from the host name by a colon. The LDAP server is used for certificate validation. The LDAP server is used only when LDAP_CA_roots is set to GSK_CA_ROOTS_LOCAL_AND_X500 and *auth_type* is not set to GSK_CLIENT_AUTH_LOCAL or GSK_CLIENT_AUTH_PASSTHRU.

LDAP_port
Specifies the LDAP server port. The default LDAP port is used if 0 is specified.

LDAP_user
Specifies the distinguished name to use when connecting to the LDAP server and is a null-terminated character string. An anonymous bind is done if NULL is specified for this field.

LDAP_password
Specifies the password to use when connecting to the LDAP server and is a null-terminated character string. This field is ignored if NULL is specified for LDAP_user.

LDAP_CA_roots
Specifies the location of CA certificates and certificate revocation lists used to validate certificates. When GSK_CA_ROOTS_LOCAL_ONLY is specified, the CA certificates and certificate revocation lists are obtained from the local database. When GSK_CA_ROOTS_LOCAL_AND_X500 is specified, the CA certificates and certificate revocation lists are obtained from the LDAP server if they are not found in the local database. Even when an LDAP server is used, root CA certificates must be found in the local database since the LDAP server is not a trusted data source.

auth_type
Specifies the client authentication type. This field is ignored unless LDAP_CA_roots is set to GSK_CA_ROOTS_LOCAL_AND_X500. The client certificate is not validated when GSK_CLIENT_AUTH_PASSTHRU is specified. The client certificate is validated by using just the local database when GSK_CLIENT_AUTH_LOCAL is specified. CA certificates and certificate revocation lists not found in the local database are obtained from the LDAP server when GSK_CLIENT_AUTH_STRONG or GSK_CLIENT_AUTH_STRONG_OVER_SSL is specified (the local database must still contain the root CA certificates). There is no difference between GSK_CLIENT_AUTH_STRONG and GSK_CLIENT_AUTH_STRONG_OVER_SSL.

gsk_initialize() supported environment variables

Environment variables are processed along with the information that is passed in the gsk_init_data structure during environment initialization. Also, during environment initialization, the key database, PKCS #12 file, key ring, or token is read.

The **gsk_initialize()** routine supports the following environment variables (See Appendix A, “Environment variables,” on page 763 for information about using the environment variables):

GSK_CERT_VALIDATE_KEYRING_ROOT
Specifies the setting of how certificates in a SAF key ring are validated.

GSK_CERT_VALIDATION_MODE
Specifies which internet standard is used for certificate validation.

GSK_CLIENT_AUTH_NOCERT_ALERT
Specifies whether the SSL server application accepts a connection from a client where client authentication is requested and the client fails to supply an X.509 certificate.

GSK_CLIENT_EXTENDED_MASTER_SECRET
Specifies if the TLS client sends the extended master secret extension to the server for TLS V1.0 handshakes.

GSK_CRL_CACHE_TIMEOUT
Specifies the number of hours that a cached LDAP CRL remains valid.

GSK_CRL_SECURITY_LEVEL
Specifies the level of security to be used when contacting LDAP servers to check CRLs for revoked certificates.

GSK_EXTENDED_RENEGOTIATION_INDICATOR
Specifies the level of enforcement of renegotiation indication as specified by RFC 5746 during the initial handshake.

GSK_KEY_LABEL
Specifies the label of the key that is used to authenticate the application.

GSK_PEER_DH_MIN_KEY_SIZE
Specifies the minimum allowed X.509 certificate Diffie-Hellman key size for a peer end-entity certificate.

GSK_PEER_DSA_MIN_KEY_SIZE
Specifies the minimum allowed X.509 certificate DSA key size for a peer end-entity certificate.

GSK_PEER_ECC_MIN_KEY_SIZE
Specifies the minimum allowed X.509 certificate ECC key size for a peer end-entity certificate.

GSK_PEER_RSA_MIN_KEY_SIZE
Specifies the minimum allowed X.509 certificate RSA key size for a peer end-entity certificate.

GSK_RENEGOTIATION
Specifies the type of session renegotiation that is allowed for an SSL environment.

GSK_RENEGOTIATION_PEER_CERT_CHECK
Specifies whether the peer certificate is allowed to change during renegotiation.

GSK_SERVER_EXTENDED_MASTER_SECRET
Specifies if the TLS server supports negotiating the extended master secret extension from clients for TLS V1.0 handshakes.

GSK_SERVER_FALLBACK_SCSV
Specifies whether the server accepts the TLS fallback Signaling Cipher Suite Value (SCSV) when the client's cipher list includes it during an SSL or TLS handshake.

GSK_SYSPLEX_SIDCACHE
Specifies whether sysplex session caching for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 server sessions is supported.

GSK_TLS_CBC_PROTECTION_METHOD
Specifies an optional SSL V3.0 or TLS V1.0 CBC IV protection method when writing application data.

GSKV2CACHESIZE
Specifies the number of entries in the SSL V2 session cache. The value that is specified by the GSK_V2_SIDCACHE_SIZE environment variable is used if the GSKV2CACHESIZE variable is not defined.

GSK_V2_CIPHER_SPECS
Specifies the SSL V2 cipher specifications in order of preference.

GSKV3CACHESIZE
Specifies the number of entries in the SSL V3 session cache. The value that is specified by the GSK_V3_SIDCACHE_SIZE environment variable is used if the GSKV3CACHESIZE variable is not defined.

GSK_V3_CIPHER_SPECS
Specifies the SSL V3 cipher specifications in order of preference (2-character values).

Related topics

- "gsk_secure_soc_close()" on page 522
- "gsk_secure_soc_init()" on page 523
- "gsk_secure_soc_read()" on page 530
- "gsk_secure_soc_write()" on page 533
- "gsk_uninitialize()" on page 538

gsk_secure_soc_close()

Closes a secure socket connection.

Format

```
#include <gskssl.h>

void gsk_secure_soc_close(
                           gsk_soc_data *      handle)
```

Parameters

handle
Specifies the connection handle returned by the **gsk_secure_soc_init()** routine.

Usage

The **gsk_secure_soc_close()** routine closes a secure connection created by the **gsk_secure_soc_init()** routine. The socket itself is not closed (the application is responsible for closing the socket). The connection can no longer be used for secure communications after calling the **gsk_secure_soc_close()** routine.

Related topics

- "gsk_initialize()" on page 517
- "gsk_secure_soc_init()" on page 523
- "gsk_secure_soc_read()" on page 530
- "gsk_secure_soc_write()" on page 533

gsk_secure_soc_init()

Initializes a secure socket connection.

Format

```
#include <gskssl.h>

gsk_soc_data * gsk_secure_soc_init(
                                   gsk_soc_init_data *     init_data)
```

Parameters

init_data
Specifies the socket connection initialization data.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it is one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ERR_INIT_PARM_NOT_VALID]
A connection initialization parameter is not valid.

[GSK_ERROR_3DES_KEYS_NOT_UNIQUE]
Triple DES key parts are not unique.

[GSK_ERROR_BAD_CERT]
A certificate is not valid.

[GSK_ERROR_BAD_DATE]
A certificate is not valid yet or is expired.

[GSK_ERROR_BAD_MAC]
Message verification failed.

[GSK_ERROR_BAD_MALLOC]
Insufficient storage is available.

[GSK_ERROR_BAD_MESSAGE]
Incorrectly-formatted message received from peer application.

[GSK_ERROR_BAD_PEER]
Peer application has violated the SSL protocol.

[GSK_ERROR_BAD_STATE]
The SSL environment has not been initialized.

[GSK_ERROR_CRYPTO]
Cryptographic error detected.

[GSK_ERROR_ICSF_CLEAR_KEY_SUPPORT_NOT_AVAILABLE]
ICSF clear key support not available.

[GSK_ERROR_ICSF_FIPS_DISABLED]
ICSF PKCS #11 services are disabled.

[GSK_ERROR_ICSF_NOT_AVAILABLE]
ICSF services are not available.

[GSK_ERROR_ICSF_NOT_FIPS]
ICSF PKCS #11 not operating in FIPS mode.

[GSK_ERROR_ICSF_SERVICE_FAILURE]
ICSF callable service returned an error.

[GSK_ERROR_INAPPROPRIATE_PROTOCOL_FALLBACK]
An inappropriate protocol fallback has been detected.

[GSK_ERROR_INCOMPATIBLE_KEY]
The certificate key is not compatible with the negotiated cipher suite.

[GSK_ERROR_INCORRECT_EXT_KEY_USAGE]
Extended key usage certificate extension does not permit the requested purpose.

[GSK_ERROR_IO]
I/O error communicating with peer application.

[GSK_ERROR_KEY_IS_SMALLER_THAN_MINIMUM]
Peer's end-entity certificate key size is smaller than the minimum size allowed.

[GSK_ERROR_LDAP]
An LDAP error is detected.

[GSK_ERROR_LDAP_NOT_AVAILABLE]
The LDAP server is not available.

[GSK_ERROR_MISMATCH_TLS_EXT_MASTER_SECRET]
Extended master secret extension mismatch detected on cached TLS handshake attempt.

[GSK_ERROR_NO_CIPHERS]
No cipher specifications.

[GSK_ERROR_NO_PRIVATE_KEY]
Certificate does not contain a private key or the private key is unusable.

[GSK_ERROR_REQ_CERT_BC_EXT_MISSING]
The required basic constraints certificate extension is missing.

[GSK_ERROR_REQUIRED_TLS_EXT_MASTER_SECRET]
Missing required TLS extended master secret extension from the remote partner.

[GSK_ERROR_RNG]
Error encountered when generating random bytes.

[GSK_ERROR_SELF_SIGNED]
A self-signed certificate cannot be validated.

[GSK_ERROR_SOCKET_CLOSED]
Socket connection closed by peer application.

[GSK_ERROR_UNKNOWN_CA]
A certification authority certificate is missing.

[GSK_ERROR_UNSUPPORTED_CERTIFICATE_TYPE]
The certificate type is not supported by System SSL.

[GSK_ERROR_VALIDATION]
Certificate validation error.

[GSK_KEYFILE_BAD_DNAME]
The specified key is not found in the key database or the key is not trusted.

[GSK_KEYFILE_BAD_LABEL]
The DName field of the gsk_soc_init_data structure is an empty string. If the default key is to be used, the DName field must be NULL.

[GSK_KEYFILE_DUPLICATE_NAME]
The key database contains multiple certificates with the same subject name as the distinguished name specified in the connection initialization data.

[GSK_SOC_BAD_V2_CIPHER]
SSL V2 cipher is not valid.

[GSK_SOC_BAD_V3_CIPHER]
SSL/TLS V3 cipher is not valid.

[GSK_SOC_NO_READ_FUNCTION]
No read function is specified in the connection initialization data.

[GSK_SOC_NO_WRITE_FUNCTION]
No write function is specified in the connection initialization data.

Usage

The **gsk_secure_soc_init()** routine initializes a secure socket connection. The **gsk_initialize()** routine must be called before any secure socket connections can be initialized. After the connection has been initialized, it can be used for secure data transmission using the **gsk_secure_soc_read()** and **gsk_secure_soc_write()** routines. The **gsk_secure_soc_close()** routine should be called to close the connection when it is no longer needed. The **gsk_secure_soc_close()** routine should not be called if an error is returned by the **gsk_secure_soc_init()** routine.

Before calling the **gsk_secure_soc_init()** routine, the application must create a connected socket. For a client, this means calling the **socket()** and **connect()** routines. For a server, this means calling the **socket()**, **listen()**, and **accept()** routines. However, SSL does not require the use of TCP/IP for the communications layer. The socket descriptor can be any integer value that is meaningful to the application. The application must provide its own socket routines if it is not using TCP/IP.

An SSL handshake is performed as part of the processing of the **gsk_secure_soc_init()** routine. This establishes the server identity and optionally the client identity. It also negotiates the cryptographic parameters to be used for the connection.

The server certificate can use either RSA, DSA, or Diffie-Hellman as the public/private key algorithm. In FIPS mode, the RSA or DSA key size must be at least 1024 bits and the Diffie-Hellman key size must be 2048 bits. An RSA certificate can be used with an RSA or ephemeral Diffie-Hellman key exchange. A DSA certificate can be used with an ephemeral Diffie-Hellman key exchange. A Diffie-Hellman certificate can be used in a fixed Diffie-Hellman key exchange. If the server's certificate contains a key usage extension during the SSL handshake, it must allow key usage as follows:

- RSA certificates using export restricted ciphers (40-bit RC4 encryption and 40-bit RC2 encryption) with a public key size greater than 512 bits must allow digital signature. If operating in FIPS mode, export restricted ciphers cannot be selected.
- Diffie-Hellman certificates using fixed Diffie-Hellman key exchange must allow key agreement.
- RSA or DSA certificates using ephemeral Diffie-Hellman key exchange must allow digital signature.
- Other RSA certificates must allow key encipherment.

System SSL does not accept Verisign Global Server ID certificates. When specified, System SSL uses these certificates as any other certificate when determining the encryption cipher to be used for the SSL session.

The client certificate must support digital signatures. This means the certificate key usage extension (if any) must allow digital signature. The key algorithm can be either the RSA encryption algorithm or the Digital Signature Standard algorithm (DSA).

The SSL server always provides its certificate to the SSL client as part of the handshake. Depending upon the server handshake type, the server may ask the client to provide its certificate. The key label that is stored in the connection is used to retrieve the certificate from the key database, PKCS #12 file, key ring, or token. The default key will be used if no label is set. The key record must contain both an X.509 certificate and a private key.

These SSL V2 cipher specifications are supported in non-FIPS mode only:

- "1" = 128-bit RC4 encryption with MD5 message authentication (128-bit secret key)
- "2" = 128-bit RC4 export encryption with MD5 message authentication (40-bit secret key)
- "3" = 128-bit RC2 encryption with MD5 message authentication (128-bit secret key)
- "4" = 128-bit RC2 export encryption with MD5 message authentication (40-bit secret key)
- "6" = 56-bit DES encryption with MD5 message authentication (56-bit secret key)
- "7" = 168-bit Triple DES encryption with MD5 message authentication (168-bit secret key)

These SSL V3 cipher specifications are supported in non-FIPS mode only:

- "00" = No encryption or message authentication and RSA key exchange
- "01" = No encryption with MD5 message authentication and RSA key exchange
- "02" = No encryption with SHA-1 message authentication and RSA key exchange
- "03" = 40-bit RC4 encryption with MD5 message authentication and RSA key exchange
- "04" = 128-bit RC4 encryption with MD5 message authentication and RSA key exchange
- "05" = 128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange
- "06" = 40-bit RC2 encryption with MD5 message authentication and RSA key exchange
- "09" = 56-bit DES encryption with SHA-1 message authentication and RSA key exchange
- "0C" = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- "0F" = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "12" = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- "15" = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate

These SSL V3 cipher specifications are supported in FIPS mode and non-FIPS mode:

- "0A" = 168-bit Triple DES encryption with SHA-1 message authentication and RSA key exchange
- "0D" = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- "10" = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "13" = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- "16" = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "2F" = 128-bit AES encryption with SHA-1 message authentication and RSA key exchange
- "30" = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- "31" = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "32" = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- "33" = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "35" = 256-bit AES encryption with SHA-1 message authentication and RSA key exchange
- "36" = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate
- "37" = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "38" = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate
- "39" = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate

The client sends a list of ciphers it supports during the SSL handshake. The server application uses this list, and the defined ciphers supported by the server, to determine the cipher to be used during the SSL handshake. This selection is done by looking through the servers cipher list for a match in the clients list. The first matching cipher is used.

Environment variables are processed along with the information passed in the gsk_init_data structure during environment initialization. Also during environment initialization, the key database, PKCS #12 file, key ring or token is read.

The environment variables that are overridden by non-NULL values in the gsk_soc_init_data structure are:

- GSK_KEY_LABEL
- GSK_V2_CIPHER_SPECS
- GSK_V3_CIPHER_SPECS

The gsk_soc_init_data structure contains these fields:

fd
: Specifies the socket descriptor for the secure connection. The socket must remain open until after the **gsk_secure_soc_close()** routine has been called to close the secure connection.

hs_type
: Specifies the intended handshake type as follows:

  GSK_AS_CLIENT
  : Performs a client SSL handshake

  GSK_AS_CLIENT_NO_AUTH
  : Performs a client SSL handshake but do not provide a client certificate to the SSL server

  GSK_AS_SERVER
  : Performs a server SSL handshake

  GSK_AS_SERVER_WITH_CLIENT_AUTH
  : Performs a server SSL handshake with client authentication

DName
: Specifies either the distinguished name or the key label of the local certificate. Specify NULL to use the default key for the key database, key ring or token.

sec_type
: Returns the selected security protocol as "SSLV2", "SSLV3", or "TLSV1". This is a static string and must not be modified or freed by the application.

cipher_specs
: Specifies the SSL V2 cipher specifications as a null-terminated string consisting of 1 or more 1-character values. Specify NULL to use the default cipher specifications ("34" if United States only encryption is enabled (System SSL Security Level 3 FMID or CPACF Feature 3863 is installed) and "4" otherwise). Valid cipher specifications that are not supported because of the installed cryptographic level will be skipped when the connection is initialized. The SSL V2 protocol can only be used when executing in non-FIPS mode.

v3cipher_specs
: Specifies the SSL V3 cipher specifications as a null-terminated string consisting of 1 or more 2-character values. Specify NULL to use the default cipher specifications:

  - "3538392F3233" if in non-FIPS mode with United States encryption enabled (System SSL Security Level 3 FMID or CPACF Feature 3863 is installed).
  - "3538392F3233" if in FIPS mode with United States only encryption enabled (System SSL Security Level 3 FMID is installed).
  - "" (empty string) if in non-FIPS mode without United States encryption enabled (System SSL Security Level 3 FMID and CPACF Feature 3863 are not installed).

  The SSL V3 cipher specifications are used for both the SSL V3 and TLS V1.0 protocols. Valid cipher specifications that are not supported because of the installed cryptographic level are skipped when the connection is initialized. The SSL V3 protocol can only be used when executing in non-FIPS mode.

skread
: Specifies the address of the read routine used during the SSL handshake. See "gsk_attribute_set_callback()" on page 107 for additional information about the I/O callback routines.

skwrite
Specifies the address of the write routine used during the SSL handshake. See "gsk_attribute_set_callback()" on page 107 for additional information about the I/O callback routines.

cipherSelected
Returns the selected cipher for the SSL V2 protocol as a 3-byte binary value:

- 0x010080 - 128-bit RC4 encryption with MD5 message authentication
- 0x020080 = 128-bit RC4 export encryption with MD5 message authentication
- 0x030080 = 128-bit RC2 encryption with MD5 message authentication
- 0x040080 = 128-bit RC2 export encryption with MD5 message authentication
- 0x060040 = 56-bit DES encryption with MD5 message authentication
- 0x0700c0 = 168-bit Triple DES encryption with MD5 message authentication

v3cipherSelected
Returns the selected cipher for the SSL V3 or TLS V1.0 protocol as a 2-byte character value with no string delimiter:

- "00" = No encryption or message authentication
- "01" = No encryption with MD5 message authentication and RSA key exchange
- "02" = No encryption with SHA-1 message authentication and RSA key exchange
- "03" = 40-bit RC4 encryption with MD5 message authentication and RSA key exchange
- "04" = 128-bit RC4 encryption with MD5 message authentication and RSA key exchange
- "05" = 128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange
- "06" = 40-bit RC2 encryption with MD5 message authentication and RSA key exchange
- "09" = 56-bit DES encryption with SHA-1 message authentication and RSA key exchange
- "0A" = 168-bit Triple DES encryption with SHA-1 message authentication and RSA key exchange
- "0C" = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- "0D" = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- "0F" = 56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "10" = 168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "12" = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- "13" = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- "15" = 56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "16" = 168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "2F" = 128-bit AES encryption with SHA-1 message authentication and RSA key exchange
- "30" = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- "31" = 128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "32" = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate

- "33" = 128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate
- "35" = 256-bit AES encryption with SHA-1 message authentication and RSA key exchange
- "36" = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSS certificate
- "37" = 256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate
- "38" = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSS certificate
- "39" = 256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate

failureReasonCode
Returns the **gsk_secure_soc_init()** error code.

cert_info
Returns peer certificate information. The application must not modify or free this information.

gsk_data
This field is ignored. The key database information is set when **gsk_initialize()** is called.

Ensure that keys and algorithms are compliant for the chosen security strength when functioning at a FIPS mode level. For more information about FIPS mode level support, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

Related topics

- “gsk_get_cipher_info()” on page 515
- “gsk_get_dn_by_label()” on page 516
- “gsk_initialize()” on page 517
- “gsk_secure_soc_close()” on page 522
- “gsk_secure_soc_read()” on page 530
- “gsk_secure_soc_reset()” on page 532
- “gsk_secure_soc_write()” on page 533

gsk_secure_soc_read()

Reads data using a secure socket connection.

Format

```
#include <gskssl.h>

int gsk_secure_soc_read(
                         gsk_soc_data *       soc_handle,
                         void *               buffer,
                         int                  size)
```

Parameters

soc_handle
Specifies the connection handle returned by the **gsk_secure_soc_init()** routine.

buffer
Specifies the buffer to receive the data read from the secure socket connection. The maximum amount of data returned by **gsk_secure_soc_read()** is 16384 (16K) bytes. If the SSL V2 protocol is used, then the maximum length is 16384 minus the length of the SSL protocol headers.

size
Specifies the size of the supplied buffer.

Results

The function return value will be the number of bytes read if no error is detected. Otherwise, it will be a negative value representing one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ERROR_BAD_BUFFER_SIZE]
The buffer address or buffer size is not valid.

[GSK_ERROR_BAD_MAC]
Message verification failed.

[GSK_ERROR_BAD_MALLOC]
Insufficient storage is available.

[GSK_ERROR_BAD_MESSAGE]
Incorrectly-formatted message received from peer application.

[GSK_ERROR_BAD_PEER]
Peer application has violated the SSL protocol.

[GSK_ERROR_BAD_SSL_HANDLE]
The connection handle is not valid.

[GSK_ERROR_CONNECTION_ACTIVE]
A read request is already active for the connection.

[GSK_ERROR_CRYPTO]
Cryptographic error detected.

[GSK_ERROR_IO]
I/O error communicating with peer application.

[GSK_ERROR_NO_NEGOTIATION]
An attempt was made to renegotiate a session when renegotiation is disabled or the peer rejected an attempted session renegotiation.

[GSK_ERROR_RENEGOTIATION_INDICATION]
Peer did not signal support for TLS Renegotiation Indication.

[GSK_ERROR_SOCKET_CLOSED]
Socket connection closed by peer application.

[GSK_ERROR_WOULD_BLOCK]
A complete SSL record is not available.

[GSK_ERROR_WOULD_BLOCK_WRITE]
An SSL handshake is in progress but data cannot be written to the socket.

Usage

The **gsk_secure_soc_read()** routine reads data from a secure socket connection and returns it in the application buffer. SSL is a record-based protocol and a single call will never return more than a single SSL record. The maximum amount of data returned by **gsk_secure_soc_read()** is 16384 (16K) bytes. If the SSL V2 protocol is used, then the maximum length is 16384 minus the length of the SSL protocol headers. The application can read an entire SSL record in a single call by supplying a buffer large enough to contain the record. Otherwise, multiple calls will be required to retrieve the entire SSL record.

SSL supports multiple threads but only one thread at a time can call the **gsk_secure_soc_read()** routine for a given connection handle. Multiple concurrent threads can call **gsk_secure_soc_read()** if each thread has its own connection handle.

SSL supports sockets in blocking mode and in non-blocking mode. When a socket is in non-blocking mode and a complete SSL record is not available, **gsk_secure_soc_read()** will return with GSK_ERROR_WOULD_BLOCK. No data will be returned in the application buffer when GSK_ERROR_WOULD_BLOCK is returned. The application should call **gsk_secure_soc_read()** again when there is data available to be read from the socket.

The peer application can initiate an SSL handshake sequence after the connection is established. If this is done and the socket is in non-blocking mode, it is possible for **gsk_secure_soc_read()** to return with GSK_ERROR_WOULD_BLOCK_WRITE. This indicates that an SSL handshake is in progress and the application should call **gsk_secure_soc_read()** again when data can be written to the socket. No data will be returned in the application buffer when GSK_ERROR_WOULD_BLOCK_WRITE is returned.

The application should not read data directly from the socket since this can cause SSL protocol errors if the application inadvertently reads part of an SSL record. If the application must read data from the socket, it is responsible for synchronizing this activity with the peer application so that no SSL records are sent while the application is performing its own read operations.

Related topics

- “gsk_initialize()” on page 517
- “gsk_secure_soc_close()” on page 522
- “gsk_secure_soc_init()” on page 523
- “gsk_secure_soc_write()” on page 533

gsk_secure_soc_reset()

Resets the session keys for a secure connection.

Format

```
#include <gskssl.h>

gsk_status gsk_secure_soc_reset(
                                 gsk_soc_data *     soc_handle)
```

Parameters

soc_handle
Specifies the connection handle returned by the **gsk_secure_soc_init()** routine.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ERR_NO_NEGOTIATION]
An attempt was made to renegotiate a session when renegotiation is disabled.

[GSK_ERROR_BAD_MALLOC]
Insufficient storage is available.

[GSK_ERROR_BAD_SSL_HANDLE]
The connection handle is not valid.

[GSK_ERROR_CONNECTION_CLOSED]
The connection was closed by the peer application.

[GSK_ERROR_IO]
I/O error communicating with peer application.

[GSK_ERROR_NOT_SSLV3]
The session is not using the SSL V3 or TLS V1.0 protocol.

[GSK_ERROR_SOCKET_CLOSED]
Socket connection closed by peer application.

Usage

The **gsk_secure_soc_reset()** routine generates new session keys for the connection. A full SSL handshake will be performed if the session has expired. Otherwise a short SSL handshake will be performed. The **gsk_secure_soc_reset()** routine can be called only for a session using the SSL V3 or TLS V1.0 protocol. The **gsk_secure_soc_reset()** routine initiates the SSL handshake but does not wait for it to complete. Any pending handshake messages will be processed when the **gsk_secure_soc_read()** routine is called to process incoming data.

Related topics

- "gsk_secure_soc_init()" on page 523

gsk_secure_soc_write()

Writes data using a secure socket connection.

Format

```
#include <gskssl.h>

int gsk_secure_soc_write(
                          gsk_soc_data *     soc_handle,
                          void *             buffer,
                          int                length)
```

Parameters

soc_handle
Specifies the connection handle returned by the **gsk_secure_soc_init()** routine.

buffer
Specifies the buffer containing the data to write to the secure socket connection.

length
Specifies the amount to write.

Results

The function return value will be the number of bytes written if no error is detected. Otherwise, it will be a negative value representing one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ERROR_BAD_BUFFER_SIZE]
The buffer address or buffer size is not valid.

[GSK_ERROR_BAD_MALLOC]
Insufficient storage is available.

[GSK_ERROR_BAD_SSL_HANDLE]
The connection handle is not valid.

[GSK_ERROR_CONNECTION_ACTIVE]
A write request is already active for the connection.

[GSK_ERROR_CONNECTION_CLOSED]
A close notification alert has been sent for the connection.

[GSK_ERROR_CRYPTO]
Cryptographic error detected.

[GSK_ERROR_IO]
I/O error communicating with peer application.

[GSK_ERROR_SOCKET_CLOSED]
Socket connection closed by peer application.

[GSK_ERROR_WOULD_BLOCK]
The SSL record cannot be written to the socket because of an EWOULDBLOCK condition.

Usage

The **gsk_secure_soc_write()** routine writes data to a secure socket connection. SSL is a record-based protocol with a maximum record length of 16384 bytes. If the SSL V2 protocol is used, then the maximum length is 16384 minus the length of the SSL protocol headers. Application data larger than the size of an SSL record will be sent using multiple records.

SSL supports multiple threads but only one thread at a time can call the **gsk_secure_soc_write()** routine for a given connection handle. Multiple concurrent threads can call **gsk_secure_soc_write()** if each thread has its own connection handle.

SSL supports sockets in blocking mode and in non-blocking mode. When a socket is in non-blocking mode and the SSL record cannot be written to the socket, **gsk_secure_soc_write()** will return with GSK_ERROR_WOULD_BLOCK. The application must call **gsk_secure_soc_write()** again when the socket is ready to accept more data, specifying the same buffer address and buffer size as the original request. A new write request must not be initiated until the pending write request has been completed as indicated by a return value of 0.

The application should not write data directly to the socket since this can cause SSL protocol errors if the application inadvertently intermixes its data with SSL protocol data. If the application must write data to the socket, it is responsible for synchronizing this activity with the peer application so that application data is not intermixed with SSL data.

Related topics

- "gsk_initialize()" on page 517
- "gsk_secure_soc_close()" on page 522
- "gsk_secure_soc_init()" on page 523
- "gsk_secure_soc_read()" on page 530

gsk_srb_initialize()

Initializes SRB support.

Format

```
#include <gskssl.h>

gsk_status gsk_srb_initialize (
                                int       num_tasks)
```

Parameters

num_tasks
Specifies the maximum number of service tasks and must be greater than 0.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_ERR_INIT_PARM_NOT_VALID]
The number of tasks parameter is not valid.

[GSK_ERROR_BAD_STATE]
The SSL environment is not initialized.

[GSK_SRB_INIT_ESTAEX]
Unable to establish ESTAE exit.

[GSK_SRB_INIT_NOT_APF]
The application is not APF-authorized.

[GSK_SRB_INIT_THREAD_CREATE]
Unable to create a thread.

Usage

The **gsk_srb_initialize()** routine will initialize the SRB (Service Request Block) support. The application must be APF-authorized in order to use SRB mode. The **gsk_srb_initialize()** routine must be called after the **gsk_initialize()** routine and before any calls to the GSKSRBRD and GSKSRBWT routines.

The SRB support provided by System SSL is a mode converter which allows an SSL read or write operation to be initiated in SRB mode but processed in TASK mode. This is necessary because SRB mode is not supported by many of the functions invoked by System SSL while processing a read or write request.

The **gsk_srb_initialize()** routine creates a monitor thread and the first service thread. Additional threads are created as needed up to the maximum number of threads specified by the num_tasks parameter. The threads run in FIPS mode if FIPS mode was set by a call to **gsk_fips_state_set()**. These threads will be destroyed and SRB mode support will be terminated when the **gsk_uninitialize()** routine is called.

See *z/OS MVS Programming: Authorized Assembler Services Guide* for more information about service request blocks.

Related topics

- "GSKSRBRD" on page 536
- "GSKSRBWT" on page 537

GSKSRBRD

Reads from a secure connection in SRB mode.

Format

```
LOAD   EP=GSKSRBRD
```

```
LR     15,0
CALL   (15), (SOCHNDLE, BUFPTR, BUFSIZE, RSNCODE)
```

Parameters

SOCHNDLE
Specifies a 4-byte word containing the gsk_soc_data address returned by the **gsk_secure_soc_init()** routine.

BUFPTR
Specifies a 4-byte word containing the address of the data buffer.

BUFSIZE
Specifies a 4-byte word containing the length of the data buffer.

RSNCODE
Specifies a 4-byte word which will contain the reason code if an error is detected. In most cases, this will be the *errno* value at the completion of the read request.

Results

The return value will be the number of bytes read if no error is detected. Otherwise, it will be a negative value representing one of the return codes listed in the **gskssl.h** include file. See the description of the **gsk_secure_soc_read()** routine for more information.

Usage

The GSKSRBRD routine is called to read from a secure connection in SRB mode. The **gsk_srb_initialize()** routine must have been called previously to initialize the SRB support. All of the parameters must be in the application storage key and must reside in the primary address space. The GSKSRBRD routine will pass the read request to one of the SRB service tasks. The service task will then call the **gsk_secure_soc_read()** routine. The GSKSRBRD routine will not return until the **gsk_secure_soc_read()** routine has completed.

Related topics

- "GSKSRBWT" on page 537
- "gsk_initialize()" on page 517
- "gsk_secure_soc_close()" on page 522
- "gsk_secure_soc_init()" on page 523
- "gsk_secure_soc_write()" on page 533
- "gsk_srb_initialize()" on page 535

GSKSRBWT

Writes to a secure connection in SRB mode.

Format

```
    LOAD    EP=GSKSRBRD
```

```
    LR      15,0
    CALL    (15), (SOCHNDLE, BUFPTR, BUFSIZE, RSNCODE)
```

Parameters

SOCHNDLE
Specifies a 4-byte word containing the gsk_soc_data address returned by the **gsk_secure_soc_init()** routine.

BUFPTR
Specifies a 4-byte word containing the address of the data buffer.

BUFSIZE
Specifies a 4-byte word containing the length of the data buffer.

RSNCODE
Specifies a 4-byte word which will contain the reason code if an error is detected. In most cases, this will be the *errno* value at the completion of the read request.

Results

The return value will be the number of bytes written if no error is detected. Otherwise, it will be a negative value representing one of the return codes listed in the **gskssl.h** include file. See the description of the **gsk_secure_soc_write()** routine for more information.

Usage

The GSKSRBWT routine is called to write to a secure connection in SRB mode. The **gsk_srb_initialize()** routine must have been called previously to initialize the SRB support. All of the parameters must be in the application storage key and must reside in the primary address space. The GSKSRBWT routine will pass the write request to one of the SRB service tasks. The service task will then call the **gsk_secure_soc_write()** routine. The GSKSRBWT routine will not return until the **gsk_secure_soc_write()** routine has completed.

Related topics

- "GSKSRBRD" on page 536
- "gsk_initialize()" on page 517
- "gsk_secure_soc_close()" on page 522
- "gsk_secure_soc_init()" on page 523
- "gsk_secure_soc_write()" on page 533
- "gsk_srb_initialize()" on page 535

gsk_uninitialize()

Terminates the SSL environment.

Format

```
#include <gskssl.h>

gsk_status gsk_uninitialize ( void )
```

Parameters

There are no parameters.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. This is a possible error:

[GSK_ERROR_CLOSE_FAILED]
An error occurred while closing the environment.

Usage

The **gsk_uninitialize()** routine will close the SSL environment created by the **gsk_initialize()** routine. New SSL connections cannot be initiated after calling the **gsk_uninitialize()** routine until the **gsk_initialize()** routine is called to initialize a new SSL environment. All resources allocated for the environment will be released unless there are active SSL connections still using the environment. If there are active connections, the environment is not closed until the last connection is closed.

Related topics

- "gsk_initialize()" on page 517
- "gsk_secure_soc_init()" on page 523

gsk_user_set()

Sets an application callback.

Format

```
#include <gskssl.h>

gsk_status gsk_user_set(
                         gsk_user_set_fid     set_id,
                         void *               set_data,
                         void *               reserved)
```

Parameters

set_id
Specifies the set function identifier.

set_data
Specifies the address of the set data.

reserved
Specify NULL for this parameter.

Results

The function return value will be 0 (**GSK_OK**) if no error is detected. Otherwise, it will be one of the return codes listed in the **gskssl.h** include file. These are some possible errors:

[GSK_BAD_PARAMETER]
A parameter is not valid.

[GSK_ERROR_BAD_STATE]
The SSL environment has not been initialized.

Usage

The **gsk_user_set()** routine will set or reset an application callback. The **gsk_initialize()** routine must be called before the **gsk_user_set()** routine can be called.

These set function identifiers are supported:

[GSK_SET_SIDCACHE_CALLBACK]
This function sets the session identifier cache callback. The set data is the address of the gsk_sidcache_callback structure. The application session identifier cache is used only for SSL servers (the internal cache is always used for SSL clients). This sets the session identifier cache for existing connections including new connections created by the **gsk_secure_soc_init()** routine.

The routine specified by the *Get* entry is called to retrieve an entry from the session identifier cache. The *session_id* parameter is the session identifier, the *session_id_length* parameter is the length of the session identifier, and the *ssl_version* parameter is the SSL protocol version number (GSK_SSLVERSION_V2 or GSK_SSLVERSION_V3). The function return value is the address of the session data buffer or NULL if an error is detected. The *FreeDataBuffer* routine will be called to release the session data buffer when it is no longer needed by the SSL runtime.

```
gsk_data_buffer * Get (
                        const unsigned char *     session_id,
                        unsigned int              session_id_length,
                        gsk_sslversion            ssl_version)
```

The routine specified by the *Put* entry is called to store an entry in the session identifier cache. The *ssl_session_data* parameter is the session data, the *session_id* parameter is the session identifier, the *session_id_length* parameter is the length of the session identifier, and the *ssl_version* parameter is the SSL protocol version number (GSK_SSLVERSION_V2 or GSK_SSLVERSION_V3). The function return value is ignored and can be a NULL address. The callback routine must make its own copy of the session data since the SSL structure will be released when the connection is closed.

```
gsk_data_buffer * Put (
                        gsk_data_buffer *         ssl_session_data,
                        const unsigned char *     session_id,
                        unsigned int              session_id_length,
                        gsk_sslversion            ssl_version)
```

The routine specified by the *Delete* entry is called to remove an entry from the session identifier cache. The session_id parameter is the session identifier, the *session_id_length* parameter is the length of the session identifier, and the *ssl_version* parameter is the SSL protocol version number (GSK_SSLVERSION_V2 or GSK_SSLVERSION_V3).

```
void Delete (
              const unsigned char *     session_id,
              unsigned int              session_id_length,
              gsk_sslversion            ssl_version)
```

The routine specified by the *FreeDataBuffer* entry is called to release the data buffer returned by the *Get* routine.

```
void FreeDataBuffer (
                      gsk_data_buffer *     ssl_session_data)
```

[GSK_RESET_SIDCACHE_CALLBACK]
This function resets the session identifier cache callback. The internal session identifier cache is used instead of an application session identifier cache. This resets the session identifier cache for existing connections including new connections created by the **gsk_secure_soc_init()** routine.

[GSK_SET_GETPEER_CALLBACK]
This function sets the peer identification callback. The peer identification callback returns the 32-bit network identifier for the remote partner. The *fd* parameter is the socket descriptor specified when the connection was initialized. The peer identification routine will be called for new connections created by **gsk_secure_soc_init()** but will not be called for existing connections.

```
unsigned long io_getpeerid (
                             int     fd)
```

[GSK_RESET_GETPEER_CALLBACK]
This function resets the peer identification callback. The internal peer identification routine will be used instead of the application routine. This applies to new connections created by **gsk_secure_soc_init()** and does not affect existing connections.

Related topics

- "gsk_initialize()" on page 517
- "gsk_secure_soc_init()" on page 523

Chapter 10. Certificate/Key management

This topic discusses the use of the z/OS shell-based **gskkyman** utility to manage private keys, certificates, and tokens. In addition, see "gskkyman command line mode examples" on page 605 for more detailed examples using the **gskkyman** utility.

This topic also provides an overview of the certificate revocation support provided in System SSL.

Introduction

SSL connections use public/private key mechanisms for authenticating each side of the SSL session and agreeing on bulk encryption keys to be used for the SSL session. To use public/private key mechanisms (termed PKI), public/private key pairs must be generated. In addition, X.509 certificates (which contain public keys) might need to be created, or certificates must be requested, received, and managed.

System SSL supports four methods for managing PKI private keys and certificates:

- A z/OS shell-based program called `gskkyman`. `gskkyman` creates, completes, and manages either a z/OS file or z/OS PKCS #11 token that contains PKI private keys, certificate requests, and certificates. The z/OS file is called a `key database` and by convention, has a file extension of `.kdb`.
- The z/OS Security Server (RACF) RACDCERT command. RACDCERT installs and maintains PKI private keys and certificates in RACF. See *z/OS Security Server RACF Command Language Reference* for details about the RACDCERT command. RACF supports multiple PKI private keys and certificates to be managed as a group. These groups are called `key rings` or z/OS PKCS #11 tokens. RACF key rings or z/OS PKCS #11 tokens are the preferred method for managing PKI private keys and certificates for System SSL.
- PKCS #12 standard files created according to PKCS #12 V3.0. These files must be created as binary format files whose fully qualified file name does not exceed 251 characters in length and does not end with .kdb, .rdb, or .sth.

  System SSL supports PKCS #12 certificate and private key objects types. Any other object types within the file are ignored. All certificates within the files are treated as trusted certificates and no certificate can be identified as a default certificate.

  The PKCS #12 file is protected by a password and the integrity of the file is ensured by a SHA-1 message authentication value.

  Because PKCS #12 files do not have labels as certificates in key database files (SAF key rings or PKCS #11 tokens can have labels as certificates in key database files), when the certificates are read into storage, they are assigned a label that uses either the PKCS #12 friendly name, if one exists, or the certificate's subject distinguished name. When the friendly name or the subject distinguished name value is greater than 127 characters, only the first 127 characters are used. If multiple certificates have the same friendly name value, the first encountered certificate is read into storage. Any other certificate with that friendly name is ignored. If a certificate is encountered that does not contain a friendly name and the subject distinguished name is empty, the processing of the PKCS #12 fails. As with key database files, SAF key rings and PKCS #11 tokens, the label is case-sensitive.
- GSKIT CMS V4 key databases as created by GSKIT can be utilized as a read-only key database.

  System SSL supports the following certificate types (key algorithms and key size) stored in GSKIT CMS V4 databases.

Table 21. Certificate types (key algorithms and key size)

Certificate key algorithm	**Key size**
Diffie-Hellman	512-2048
DSA	512-2048

Table 21. Certificate types (key algorithms and key size) (continued)

Certificate key algorithm	**Key size**
ECC	Brainpool 160-512
	NIST 192-521
RSA	512-4096

The certificate signature algorithms that are supported:

– RSA MD5, SHA-1, 224, 256, 384, 512

– DSA SHA-1, 224, 256

– ECDSA SHA-1, 224, 256, 384, 512

– RSA-SSA-PSS SHA256, 384, 512

The GSKIT CMS V4 key database is protected by a password. The maximum supported password length is 31 characters. The use of the GSKIT CMS V4 stash file is allowed and follows the same guidelines as System SSL key database files.

The GSKIT CMS V4 key database type and certificate content can be displayed by using the System SSL database utility **gskkyman** in command line mode.

When you display certificate content by using the **gskkyman** command line, unsupported certificates are tolerated when displayed. Unsupported algorithms with the certificate are displayed as object identifiers (OIDS).

Command line **gskkyman** does not support import, export, signing, or stash file creation for GSKIT CMS V4 key databases.

Interactive **gskkyman** does not support GSKIT CMS V4 key databases.

GSKIT CMS V4 key databases are supported for use through the **gsk_environment_init()** or **gsk_initialize()** routines for the establishment of secure SSL/TLS connections. Attempting to establish a secure SSL/TLS connection by using certificates that contain unsupported algorithms results in connection establishment failure.

The System SSL application uses the GSK_KEYRING_FILE parameter of the **gsk_attribute_set_buffer()** API or the GSK_KEYRING_FILE environment variable to specify the locations of the PKI private keys and certificates to System SSL. If you are using a z/OS key database, GSKIT CMS V4 key database, or a PKCS #12 file, the name is passed in this parameter. If you are using a RACF key ring or PKCS #11 token, the name of the key ring or token is passed in this parameter.

x.509 certificate revocation

When x.509 certificates are issued, they are assigned a validity period that defines a start and end (expiration) date and time for the certificate. Certificates are considered valid if used during the validity period. If the certificate is deemed to be no longer trustable prior to its expiration date, it can be revoked by the issuing Certificate Authority (CA). The process of revoking the certificate is known as certificate revocation. There are a number of reasons why certificates are revoked. Some common reasons for revocation are:

- Encryption keys of the certificate have been compromised.
- Errors within an issued certificate.
- Change in usage of the certificate.
- Certificate owner is no longer deemed trusted.

Two methods of revocation are supported:

Certificate Revocation List (CRL)

A CRL is a list of revoked certificates (by serial number) that have been issued and then subsequently revoked by a given CA. CRLs are generally published on a periodic interval or can be published only when a certificate is revoked by the CA.

The CRL, like a certificate, is signed by the owning CA to ensure the authenticity of the CRL contents and has a start and end (expiration) date and time. The start date and time is known as thisUpdate and the end date and time is known as the nextUpdate.

Supported CRLs can be obtained from a dedicated LDAP server or through a certificate's CRL Distribution Point (CDP) extension. HTTP Uniform Resource Identifier (URIs) values within the CDP may be used.

System SSL uses the HTTP/1.1 protocol to send the HTTP CRL request and requires an HTTP/1.0 or HTTP/1.1 protocol response. The HTTP response must include a valid content-length field that has the length of the CRL in bytes.

For information about configuring your SSL application to perform CRL revocation checking for SSL secure connections, see "SSL/TLS partner certificate revocation checking" on page 52.

For information about utilizing CRL revocation information from a CMS application, see "gsk_validate_certificate_mode()" on page 491.

Online Certificate Status Protocol (OCSP) responses

OCSP is an internet protocol used for obtaining the revocation status of an x.509 certificate. The protocol defines the type of data that is exchanged between the requester of the revocation status (OCSP client) and the server (OCSP responder) providing the revocation status information. Certificate revocation information is provided by the OCSP responder through an OCSP response.

The OCSP response, like a CRL, is signed by the owning CA (or designated CA) to ensure the authenticity of the OCSP response contents. If the OCSP response signing certificate contains a key usage extension, the crlSign, digitalSignature, or nonRepudiation bits must be set. If the OCSP response signing certificate contains an extended key usage extension, the ocspSigning capability must be specified. The OCSP response has a start and end (expiration) date and time. The start date and time is known as thisUpdate and the end date and time is known as the nextUpdate.

Supported OCSP responses can be obtained from a dedicated OCSP responder or through OCSP responders identified through a certificate's Authority Information Access (AIA) extension. AIA extensions may be used when the extension contains an entry with an OCSP access method and a URI access location. The AIA extension can contain multiple entries.

System SSL uses the HTTP/1.1 protocol to send the OCSP request and requires the OCSP response to use either an HTTP/1.0 or HTTP/1.1 protocol response. The HTTP response must include a valid content-length field that has the length of the OCSP response in bytes.

For information about configuring your SSL application to perform OCSP revocation checking for SSL secure connection, see "SSL/TLS partner certificate revocation checking" on page 52.

For information about utilizing OCSP revocation information from a CMS application, see "gsk_validate_certificate_mode()" on page 491.

gskkyman overview

gskkyman is a z/OS shell-based program that creates, completes, and manages a z/OS file or z/OS PKCS #11 token that contains PKI private keys, certificate requests, and certificates. The z/OS file is called a **key database** and, by convention, has a file extension of **.kdb**. There is also an **.rdb** file that is a counterpart to the **.kdb** file.

The **gskkyman** utility only supports clear key operations.

The **gskkyman** utility only supports certificates that conform to RFC 2459 (tools.ietf.org/html/rfc2459) or RFC 3280 (tools.ietf.org/html/rfc3280).

RFC 5280 (tools.ietf.org/html/rfc5280) certificates can be used with **gskkyman** provided they conform to RFC 3280 rules for the certificate issuer name and subject name comparisons. Specifically, RFC 3280 indicates that UTF-8 values in the distinguished names must pass a case-sensitive (exact match) comparison to be considered equal. The **gskkyman** utility uses the issuer name and subject name values in the certificate to determine if a certificate is self-signed, and to perform certificate chaining. Therefore, **gskkyman** expects distinguished name attribute values to match according to a case-sensitive comparison when they are encoded as UTF-8 strings. Certificates that contain distinguished names with UTF-8 encoded attribute values for either the issuer name, the subject name, or both, that match through a case-insensitive comparison, can be created according to RFC 5280. Such certificates cause the **gskkyman** utility to fail checking for self-signed certificates and fail to correctly build certificate chains. Therefore, these certificates cannot be used with **gskkyman**.

The interface to **gskkyman**, while command-line based, is an interactive dialog between you (the user) and the utility. At each step, the interactive **gskkyman** utility prompts you with one or more lines of output and expects a numeric choice to be supplied as input at the prompt. When a choice is made, the **gskkyman** utility prompts you for the individual pieces of information that is needed to fulfill the request. You are prompted for each piece of information. Many times there is a default choice that is listed between parentheses at the end of the command prompt. If the default choice is acceptable, press Enter to select the default. If you want other than the default, enter the value at the prompt and press Enter. If a value is entered that is outside of the acceptable range of inputs, you are prompted again for the information.

Note: For a description of command-line mode functions and options, see “gskkyman command line mode syntax” on page 601.

Setting up the environment to run gskkyman

gskkyman uses the DLLs that are installed with System SSL and must have access to these at run time. **gskkyman** must also have access to the message catalogs. The /bin directory includes a symbolic link to **gskkyman**; therefore, if your PATH environment variable contains this directory, **gskkyman** is located. If your PATH environment variable does not contain this directory, add /usr/lpp/gskssl/bin to your PATH using:

```
PATH=$PATH:/usr/lpp/gskssl/bin
```

`/usr/lib/nls/msg/En_US.IBM-1047` (and `/usr/lib/nls/msg/Ja_JP.IBM-939` for JCPT45J installations) include symbolic links to the message catalogs for **gskkyman**. If they do not include these links, add `/usr/lpp/gskssl/lib/nls/msg` to your NLSPATH using this command:

```
export NLSPATH=$NLSPATH:/usr/lpp/gskssl/lib/nls/msg/%L/%N
```

This setting assumes that your environment has the LANG environment variable set to `En_US.IBM-1047` (`or Ja_JP.IBM-939` for JCPT45J installations that expect Japanese messages and prompts). If LANG is not set properly, set the NLSPATH environment variable using this command:

```
export NLSPATH=/usr/lpp/gskssl/lib/nls/msg/En_US.IBM-1047/%N:$NLSPATH
```

or for JCPT45J installations that expect Japanese messages and prompts:

```
export NLSPATH=/usr/lpp/gskssl/lib/nls/msg/Ja_JP.IBM-939/%N:$NLSPATH
```

The DLLs for System SSL are installed into a partitioned data set (PDSE) in HLQ.SIEALNKE. These DLLs are not installed in SYS1.LPALIB by default. If System SSL is to execute in FIPS mode, the DLLs in the HLQ.SIEALNKE data set cannot be put into the LPA.

If the System SSL DLLs are not in either the dynamic LPA or system link list, you must set the STEPLIB environment variable to find the DLLs. For example:

```
export STEPLIB=$STEPLIB:<HLQ>.SIEALNKE
```

During installation, the sticky bit is set on for the **gskkyman** utility. If the sticky is turned off, attempts to invoke the **gskkyman** utility results in message GSK00009E indicating that a problem exists with the installation of the SSL utility, **gskkyman**.

To check the sticky bit setting, issue:

```
ls -l /usr/lpp/gskssl/bin/gskkyman
```

The first part of the output should be:

```
-rwxr-xr-t
```

The t indicates that the sticky bit is on.

To set the sticky bit on, from an authorized id, issue:

```
chmod +t /usr/lpp/gskssl/bin/gskkyman
```

If access to the ICSF callable services are protected with CSFSERV class profiles on your system, the user ID issuing the **gskkyman** utility might need to be given READ authority to call ICSF callable services CSFIQA, CSFPPRF, CSFPGKP, CSFPGSK, CSFPGAV, CSFPTRD, CSFPTRC, CSFPPKS, and CSFPPKV. If these callable services are protected with a generic CSF* profile in the CSFSERV class, access can be granted by entering:

```
PERMIT CSF* CLASS(CSFSERV) ID(user-ID) ACCESS(READ)
SETROPTS RACLIST(CSFSERV) REFRESH
```

Key database files

Key database files are password protected because they contain the private keys that are associated with some of the certificates that are contained in the key database. Private keys, as their name implies, should be protected because their value is used in verifying the authenticity of requests made during PKI operations.

It is suggested that key database files be set with these string file permissions:

```
-rw------- (600)   (read-write for only the owner of the key database)
```

The owner of the key database should be the user managing the key database. The program using System SSL (and the key database) must have at least read permission to the key database file at run time. If the program is a server program that runs under a different user ID than the administrator of the key database file, you should set up a group to control access to the key database file. In this case, it is suggested that you set the permissions on the key database file to:

```
-rw-r---- (640)   (read-write for owner and read-only for group)
```

The owner of the key database file is set to the administrator user ID and the group owner of the key database file is set to the group that contains the server that is using the key database file.

A key database that is created as a FIPS mode database, can only be updated by **gskkyman** or by using the CMS APIs executing in FIPS mode. Such a database, however, may be opened as read-only when executing in non-FIPS mode. Key databases created while in non-FIPS mode cannot be opened when executing in FIPS mode.

z/OS PKCS #11 tokens

z/OS PKCS #11 tokens are managed and protected by ICSF. ICSF uses the CRYPTOZ SAF class to determine if the issuer of **gskkyman** is permitted to perform the operation against a z/OS PKCS #11 token. The resources for this class are:

- USER.tokenname
- SO.tokenname

The **gskkyman** utility provides limited functionality for PKCS #11 token certificates that have secure private keys. If a PKCS #11 certificate has a secure private key, the following functions are allowed:

- Showing certificate and key information.
- Setting the key as default.
- Exporting a certificate to a file.
- Deleting a certificate and key.
- Changing the label.

If a PKCS #11 token certificate has a secure private key, the following functions are not allowed:

- Copying certificate and key to another token.
- Exporting certificate and key to a file.
- Creating a signed certificate and key.
- Creating a certificate renewal request.

A PKCS #11 token certificate with a clear private key is allowed full **gskkyman** functionality.

When displaying token key information for a PKCS #11 certificate's private key, the private key type indicates the private key is either clear or secure.

Table 22 on page 546 illustrates the SAF access levels required to perform certain functions. The 3 SAF levels in order of increasing accessibility are READ, UPDATE, and CONTROL. The higher levels each retain all the permissions of the previous level including gaining additional capability. For more information, see the Token Access Levels table under Overview of z/OS support for PKCS #11 in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

Table 22. SAF access levels

Resource	Function	SAF access level
USER.token-name CRYPTOZ	Create/delete/modify CA certificate and private key	Control
	Create/delete/modify user certificate and private key	Update
	Read certificate and private key	Read
	Set default key	Update
SO.token-name CRYPTOZ	Create or delete token	Update
	Read/create/delete/modify certificate (but not the private key)	Read
	Read/create/delete/modify private key	Control
	Set default key	Read

gskkyman interactive mode descriptions

Interactive mode is entered when the **gskkyman** utility is entered without any parameters. A series of menus are presented to allow you to select the database functions to be performed. Leading and trailing blanks are removed from data entries but embedded blanks are retained. Blanks are not removed from passwords.

Interactive mode does not support opening GSKIT CMS V4 key databases. Command line **gskkyman** may be used to display GSKIT CMS V4 key database information.

Database menu

This is the main menu that is displayed when the **gskkyman** utility starts:

```
        Database Menu

    1  - Create new database
    1b - Create new empty database
    2  - Open database
    3  - Change database password
    4  - Change database record length
    5  - Delete database
    6  - Create key parameter file
    7  - Display certificate file (Binary or Base64 ASN.1 DER)

   11  - Create new token
   12  - Delete token
   13  - Manage token
   14  - Manage token from list of tokens

    0  - Exit program

Enter option number:
```

Figure 2. Database menu

Create new database or Create new empty database

These options create a new key database and the associated request database. Choosing option **1** creates a new database with several initial CA certificates. Choosing option **1b** creates a new empty database without any initial CA certificates in it. In both cases, you are prompted to enter the key database name, the database password, the password expiration interval, and the database record length and choose either a FIPS or non-FIPS database (see "Key database files" on page 545 for a discussion of FIPS mode databases).

The fully-qualified key database name must be between 2 and 251 characters. The file can contain an extension consisting of 1 to 3 characters. The suggested extension is ".kdb". The maximum database name is 247 characters if the name does not end with an extension to allow for the addition of an extension when creating the request database or the password stash file. The key database name may not end with ".rdb" or ".sth" as these extensions are reserved for the request and the password stash file.

The database password must be between 1 and 128 characters. A password exceeding 128 characters will be truncated to 128 characters.

The password expiration interval must be between 0 and 9999 days (a value of 0 indicates that the password does not expire). The expiration date cannot exceed February 6, 2106, at 23:59:59 UTC.

The record length must be large enough to contain the largest certificate to be stored in the database and must be between 2500 and 65536.

Two files will be created: the key database and the request database. The request database has an extension of '.rdb'. The file access permissions will be set so only the owner has access to the files.

Open database

This option will open an existing database. You will be prompted to enter the key database name and the database password.

The fully-qualified key database name must be between 2 and 251 characters and should either have no extension or an extension of '.kdb' (the maximum database name is 247 characters if the name does not end with an extension of 1-3 characters to allow for the addition of an extension when accessing the request database or the password stash file). The key database name may not end with '.rdb' or '.sth' as these extensions are reserved for the request database and the password stash file.

Change database password

This option will change the database password. You can change the password at any time but you must change it once it has expired in order to access the database once more. You will be prompted to enter the key database name, the current database password, the new database password, and the new password expiration interval.

The new database password must be between 1 and 128 characters.

The password expiration interval must be between 0 and 9999 days (a value of 0 indicates that the password does not expire). The expiration date cannot exceed February 6, 2106, at 23:59:59 UTC.

Change database record length

This option will change the database record length. All database records have the same length and database entries cannot span records. You can increase the record length if you find it is too small to store a new certificate. You can decrease the record length to reduce the database size if the original record length is too large. You cannot reduce the record length to a value smaller than the largest certificate currently in the database. You will be prompted to enter the key database name, the database password, and the new record length.

The new record length must be between 2500 and 65536.

Delete database

This option will delete the key database, the associated request database, and the database password stash file. You will be prompted to enter the key database name.

Create key parameter file

This option will create a file containing a set of key generation parameters. Key generation parameters are used when generating Digital Signature Standard (DSS) and Diffie-Hellman (DH) keys. The parameters will be stored in the specified file as an ASN.1-encoded sequence in Base64 format. This file can then be used when creating a signed certificate. The same key generation parameters can be used to generate multiple public/private key pairs. Using the same key generation parameters significantly reduces the time required to generate a public/private key pair. In addition, the Diffie-Hellman key agreement method requires both sides to use the same group parameters in order to compute the key exchange value. See FIPS 186-4: Digital Signature Standard (DSS) (nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf) and RFC 2631 (tools.ietf.org/html/rfc2631) for more information about the key generation parameters. The key parameter generation process can take from 1 to 10 minutes depending upon key size, processor speed, and system load.

Display certificate file (Binary or Base64 ASN.1 DER)

This option displays information about an X.509 certificate file. You will be prompted to enter the certificate file name. The fully-qualified certificate file name must be between 2 and 251 characters. The specified file must contain either a binary ASN.1 DER-encoded certificate or the Base64-encoding of a binary ASN.1 stream. A Base64-encoded certificate must be in the local code page.

Note: Information retrieved for z/OS PKCS #11 tokens is not cached. Each time a menu is displayed, the information is retrieved from the ICSF TKDS (token key dataspace). This is also true when displaying the list of available z/OS PKCS #11 tokens. On return from displaying a subordinate menu, the current list of tokens is retrieved and the menu refreshed.

Create new token

This option will create a new token. You will be prompted to enter the token name.

The name must be a unique non-empty string and consist of characters that are alphanumeric, national (@ -x5B, # -x7B, $ -x7C) and period (x4B).

The name is specified in the local code page.

The first character must be alphabetic or national. Lowercase letters are permitted but will be folded to uppercase.

Once the token is created the Database menu is displayed.

Delete token

This option will delete the key token. You will be prompted to enter the token name. If the token exists, the user is prompted again to re-enter the full token name as confirmation before deletion of the specified token.

Note: If name consists of lowercase characters it will be uppercased when processed.

Manage token

This option manages the token. You will be prompted to enter the token name. The token that matches the entered name is then used in the Token Management Menu that is subsequently displayed.

Note: If name consists of lowercase characters it will be uppercased when processed.

Manage token from list of tokens

This option displays a list of existing tokens by name from which an entry can be chosen for use in the Token Management Menu that is subsequently displayed.

Note: If name consists of lowercase characters it will be uppercased when processed.

Key/Token management

The Key/Token Management menus allow for the creation/deletion/management of certificates within a key database file or z/OS PKCS #11 token. Once the key database or token is created, the management of the certificates within the repository is very similar. This is shown throughout this topic by the key database menu and token menu being displayed side by side in the figures.

Key Management Menu and Token Management Menu

The **Key Management Menu** is displayed once the key database has been created or opened. The key database and the associated request database are opened for update and remain open until you return to the **Database Menu**.

The **Token Management Menu** is displayed once a z/OS PKCS #11 token has been opened.

```
       Key Management Menu

       Database: Database_name
       Expiration Date: Expiration Date

   1 - Manage keys and certificates
   2 - Manage certificates
   3 - Manage certificate requests
   4 - Create new certificate request
   5 - Receive requested certificate or a renewal certificate
   6 - Create a self-signed certificate
   7 - Import a certificate
   8 - Import a certificate and a private key
   9 - Show the default key
  10 - Store database password
  11 - Show database record length

   0 - Exit program

Enter option number (press ENTER to return to previous menu):
```

Figure 3. Key Management Menu

```
       Token Management Menu

       Token: Token_name

       Manufacturer:  z/OS PKCS11 API
       Model:  HCR77D2
       Flags:  0x00000509 (INITIALIZED,PROT AUTH
               PATH,USER PIN INIT,RNG)

   1 - Manage keys and certificates
   2 - Manage certificates
   3 - Manage certificate requests
   4 - Create new certificate request
   5 - Receive requested certificate or a renewal certificate
   6 - Create a self-signed certificate
   7 - Import a certificate
   8 - Import a certificate and a private key
   9 - Show the default key
  10 - Delete token

   0 - Exit program

Enter option number (press ENTER to return to previous menu):
```

Figure 4. Token Management Menu

Manage keys and certificates

This option manages certificates with private keys. A list of key labels is displayed. Pressing the ENTER key without making a selection will display the next set of labels. Selecting one of the label numbers will display this menu:

```
       Key and Certificate Menu

       Label: Certificate_label_name

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database/token
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate and key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu):
```

Figure 5. Key and Certificate Menu

```
       Token Key and Certificate Menu

       Label: Certificate_label_name

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database/token
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate and key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu):
```

Figure 6. Token Key and Certificate Menu

Show certificate information
This option displays information about the X.509 certificate associated with the private key.

Show key information
This option displays information about the private key.

Set key as default
This option makes the current key the default key for the database.

Set certificate trust status
This option sets or resets the trusted status for the X.509 certificate. A certificate cannot be used for authentication unless it is trusted.

Note: All z/OS PKCS #11 token certificates are automatically created with the status set to trusted. Changing of the trust status is not supported for z/OS PKCS #11 token certificates.

Copy certificate and key to another database/token
This option copies the certificate and key to another token or a database. An error is returned if the certificate is already in the token/database or if the label is not unique. A certificate and key may only be copied into a FIPS mode database from another FIPS mode database. A certificate and key may not be copied from a non-FIPS mode database or a PKCS #11 token to a FIPS mode database.

Export certificate to a file
This option exports just the X.509 certificate to a file. The supported export formats are ASN.1 Distinguished Encoding Rules (DER) and PKCS #7 (Cryptographic Message Syntax)

Export certificate and key to a file
This option exports the X.509 certificate and its private key to a file. The private key is encrypted when it is written to the file. The password you select will be needed when you import the file. The supported export formats for a key database file are PKCS #12 Version 1 (obsoleted) and PKCS #12 Version 3. For z/OS PKCS #11 tokens and FIPS mode databases, the export format supported is PKCS #12 Version 3. The strong encryption option uses Triple DES to encrypt the private key while the export encryption option uses 40-bit RC2. Strong encryption is the only supported option when exporting from a FIPS database. The export file will contain the requested certificate and its certification chain.

Delete certificate and key
The certificate and its associated private key are deleted.

Change label
This option will change the label for the database record.

Create a signed certificate and key
This option will create a new certificate and associated public/private key pair. The new certificate will be signed using the certificate in the current record and then stored in either the key database file or z/OS PKCS #11 token.

DSS and DH key generation parameters must be compatible with the requested key type and key size.

Keys are in the same domain if they have the same set of key generation parameters. See FIPS 186-2: Digital Signature Standard (DSS) (csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf) and RFC 2631 (tools.ietf.org/html/rfc2631) for more information about the key generation parameters. The subject name and one or more subject alternate names can be specified for the new certificate.

The subject name is always an X.500 directory name while a subject alternate name can be an X.500 directory name, a domain name, an email address, an IP address, or a uniform resource identifier. An X.500 directory name consists of common name, organization, and country attributes with optional organizational unit, city/locality, and state/province attributes. A domain name is one or more tokens separated by periods. An email address consists of a user name and a domain name separated by '@'. An IP address is an IPv4 address (nnn.nnn.nnn.nnn) or an IPv6 address (nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn). A uniform resource identifier consists of a scheme name, a domain name, and a scheme-specific portion.

The signature algorithm used when signing the certificate is derived from the key algorithm of the signing certificate and the following digest type:

- For RSA signatures, the digest type matches that used in the signature algorithm of the signing certificate. If the digest type is not a SHA-based digest, then SHA-1 is used.
- For RSASSA-PSS signatures, the digest type is dependent upon the signer’s certificate RSA key size. If the signer key size is greater than or equal to 2048 and less than 3072, the RSASSA-PSS signature digest type is SHA-256. If the signer key size is greater than or equal to 3072 and less than 4096, the RSASSA-PSS signature digest type is SHA-384. Otherwise if the signer key size is 4096, the RSASSA-PSS signature digest type is SHA-512. Signing with a key size less than 2048 or greater than 4096 is not supported.
- For DSA signatures using a 1024-bit DSA key, the digest type is SHA-1. When using a 2048-bit DSA key, the digest type is SHA-256.
- For ECC Signatures, the digest type is the suggested digest for the key size of the ECC private key, as specified in Table 2 on page 13.

Possible signature algorithms are:

- x509_alg_sha1WithRsaEncryption
- x509_alg_sha224WithRsaEncryption
- x509_alg_sha256WithRsaEncryption
- x509_alg_sha384WithRsaEncryption
- x509_alg_sha512WithRsaEncryption
- x509_alg_dsaWithSha1
- x509_alg_dsaWithSha224
- x509_alg_dsaWithSha256
- x509_alg_ecdsaWithSha256
- x509_alg_ecdsaWithSha384
- x509_alg_ecdsaWithSha512
- x509_alg_mgf1Sha256WithRsaSsaPss
- x509_alg_mgf1Sha384WithRsaSsaPss
- x509_alg_mgf1Sha512WithRsaSsaPss

Create a certificate renewal request

This option will create a certification request using the subject name and public/private key pair from an existing certificate. The certificate request will be exported to a file in Base64 format. This file can then be sent to a certification authority for processing. The certificate returned by the certification authority can then be processed using option 5 (Receive requested certificate or a renewal certificate) on the **Key Management Menu** or **Token Management Menu**. The new certificate will replace the existing certificate.

Manage certificates

This option manages certificates without private keys. A list of key labels is displayed. Pressing the ENTER key without making a selection will display the next set of labels. Selecting one of the label numbers will display this menu:

```
       Certificate Menu

       Label: Certificate_label_name

   1 - Show certificate information
   2 - Set certificate trust status
   3 - Copy certificate to another database/token
   4 - Export certificate to a file
   5 - Delete certificate
   6 - Change label

   0 - Exit program

Enter option number (press ENTER to return to previous menu):
```

Figure 7. Certificate Menu

```
       Token Certificate Menu

       Label: Certificate_label_name

   1 - Show certificate information
   2 - Set certificate trust status
   3 - Copy certificate to another database/token
   4 - Export certificate to a file
   5 - Delete certificate
   6 - Change label

   0 - Exit program

Enter option number (press ENTER to return to the previous menu):
```

Figure 8. Token Certificate Menu

Show certificate information
This option displays information about the X.509 certificate.

Set certificate trust status
This option sets or resets the trusted status for the X.509 certificate. A certificate cannot be used for authentication unless it is trusted.

Note: All z/OS PKCS #11 token certificates are automatically created with the status set to trusted. Changing of the trust status is not supported for z/OS PKCS #11 token certificates.

Copy certificate to another database/token
This option copies the certificate to another token or a key database. An error is returned if the certificate is already in the token/database or if the label is not unique. A certificate may only be copied into a FIPSmode database from another FIPSmode database. A certificate may not be copied from a non-FIPSmode database or a PKCS #11 token to a FIPSmode database.

Export certificate to a file
This option exports the X.509 certificate to a file. The supported export formats are ASN.1 DER (Distinguished Encoding Rules) and PKCS #7 (Cryptographic Message Syntax). The export file will contain just the requested certificate when the DER format is selected. The export file will contain the requested certificate and its certification chain when the PKCS #7 format is selected.

Delete certificate
The certificate is deleted.

Change label
This option will change the label for the certificate.

Manage certificate requests

This option manages certificate requests. A list of request labels is displayed. Pressing the ENTER key without making a selection displays the next set of labels. Selecting one of the label numbers will display this menu:

```
       Request Menu

        Label: label_name

   1 - Show key information
   2 - Export certificate request to a file
   3 - Delete certificate request and key
   4 - Change label

   0 - Exit program

Enter option number (press ENTER to return to previous menu):
```

Figure 9. Request Menu 1

```
        Token Certificate Request Menu

        Label: label_name

   1 - Show key information
   2 - Export certificate request to a file
   3 - Delete certificate request and key
   4 - Change label

   0 - Exit program

Enter option number (press ENTER to return to previous menu):
```

Figure 10. Request Menu 2

Show key information
This option displays information about the private key associated with the certificate request along with the certificate request subject name and optional subject alternate name values.

Export certificate request to a file
This option exports the certificate request to a file in Base64 format. This file can then be sent to a certification authority for processing.

Delete certificate request and key
The certificate request and its associated private key are deleted.

Change label
This option will change the label for the certificate request.

Create new certificate request

This option creates a certificate request using either RSA, DSA, or ECC encryption for the public and private keys. The certificate request is exported to a file in Base64 format. This file can then be sent to a certification authority for processing.

For key databases:
The label has a maximum length of 127 characters and is used to reference the certificate in the request database. The label is also used when the certificate is received, so it must be unique in both the request and key databases. It must consist of characters that can be represented as 7-bit ASCII characters (letters, numbers, and punctuation) in the ISO8859-1 code page.

For tokens:
The label has a maximum length of 32 characters and is used to reference the certificate request. The label is also used when the certificate is received, so it must be unique in the token. It must consist of characters that can be represented in the IBM1047 code page.

The subject name and one or more subject alternate names can be specified for the new certificate. The subject name is always an X.500 directory name while a subject alternate name can be an X.500 directory name, a domain name, an email address, an IP address, or a uniform resource identifier. An X.500 directory name consists of common name, organization, and country attributes with optional organizational unit, city/locality, and state/province attributes. A domain name is one or more tokens that are separated by periods. An email address consists of a user name and a domain name that is separated by '@'. An IP address is an IPv4 address (nnn.nnn.nnn.nnn) or an IPv6 address (nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn). A uniform resource identifier consists of a scheme name, a domain name, and a scheme-specific portion; for example:

```
http://www.endicott.mycompany.com/main.html
```

Receive requested certificate or a renewal certificate

This option receives the signed certificate returned by the certification authority. The certificate can be either a new or renewal certificate issued in response to a certificate request or a renewal of an existing certificate without a corresponding certificate request. If the certificate was issued in response to a certificate request, the certificate request must still be in the request database or token. If this is a renewal certificate without a certificate request, the old certificate must still be in the key database or token and must have the same issuer name and public key. If the key database or token does not contain the private key of the old certificate or contains certificates signed by the old certificate, then the subject name must also be the same when renewing the certificate.

The certificate file must contain either an ASN.1 DER-encoded sequence as defined in RFC 2459 (tools.ietf.org/html/rfc2459), RFC 3280 (tools.ietf.org/html/rfc3280), RFC 5280 (tools.ietf.org/html/rfc5280), or a signed data message as defined in PKCS #7 (Cryptographic Message Syntax). The data can either be the binary value or the Base64 encoding of the binary value.

If the import file is in PKCS #7 format, the first certificate in the file must be the request certificate, otherwise the request will fail with 'unable to locate matching request'. The certification chain will be imported if it is contained in the import file. The certificate subject name will be used as the label for certificates added from the certification chain. A chain certificate will not be added if the label is not unique or if the certificate is already in the database or token.

Base64 data is in the local code page. A DER-encoded sequence must start with the encoding header '-----BEGIN CERTIFICATE-----' and end with the encoding footer '-----END CERTIFICATE-----'. A PKCS #7 signed data message must start with the encoding header '-----BEGIN CERTIFICATE-----' and end with the encoding footer '-----END CERTIFICATE-----' or start with the encoding header '-----BEGIN PKCS #7 SIGNED DATA-----' and end with the encoding footer '-----END PKCS #7 SIGNED DATA-----'.

An intermediate CA or end-entity certificate is a certificate signed by another entity. The key database or token must already contain a certificate for the issuer. The certificate will not be imported if the certificate authenticity cannot be validated or if the database or token already contains the certificate.

The certificate request entry will be deleted once the certificate has been received.

Create a self-signed certificate

This option creates a self-signed certificate using either RSA, DSA, or ECC encryption for the public and private keys, and a certificate signature that is based on a SHA digest algorithm. The SHA digest algorithm that is used depends on the key algorithm that is chosen for the certificate:

- If an RSA certificate is requested, the user is prompted to choose the SHA digest algorithm required.
- For RSASSA-PSS signatures, the digest type is dependent upon the signer’s certificate RSA key size. If the RSA key size is 4096, the RSASSA-PSS signature digest type is SHA-512. Otherwise, the RSASSA-PSS signature digest type is SHA-256.
- An ECC certificate uses the suggested digest for the key size of the ECC key, as specified in Table 2 on page 13.
- A 1024-bit DSA certificate uses SHA-1. For a 2048-bit DSA certificate, the user is prompted to choose the SHA digest algorithm required.

Possible signature algorithms are:

- x509_alg_sha1WithRsaEncryption
- x509_alg_sha224WithRsaEncryption
- x509_alg_sha256WithRsaEncryption
- x509_alg_sha384WithRsaEncryption
- x509_alg_sha512WithRsaEncryption
- x509_alg_dsaWithSha1
- x509_alg_dsaWithSha224
- x509_alg_dsaWithSha256
- x509_alg_ecdsaWithSha256
- x509_alg_ecdsaWithSha384
- x509_alg_ecdsaWithSha512
- x509_alg_mgf1Sha256WithRsaSsaPss
- x509_alg_mgf1Sha384WithRsaSsaPss
- x509_alg_mgf1Sha512WithRsaSsaPss

The certificate can be created for use by a certification authority or an end user. A CA certificate can be used to sign other certificates and certificate revocation lists while an end user certificate can be used for authentication, digital signatures, and data encryption.

Only an RSA CA certificate with a key size of 2048 through 4096 inclusive can sign using RSASSA-PSS signature algorithm.

An RSA end user or server certificate using the RSASSA-PSS signature algorithm must have a key size of 2048 through 4096 inclusive.

For key databases:
The label has a maximum length of 127 characters and is used to reference the certificate in the request database. The label is also used when the certificate is received, so it must be unique in both the request and key databases. It must consist of characters that can be represented as 7-bit ASCII characters (letters, numbers, and punctuation) in the ISO8859-1 code page.

For tokens:
The label has a maximum length of 32 characters and is used to reference the certificate request. The label is also used when the certificate is received, so it must be unique in the token. It must consist of characters that can be represented in the IBM1047 code page.

The number of days until the certificate expires must be between 1 and 9999. The expiration date cannot exceed February 6, 2106, at 23:59:59 UTC.

The subject name and one or more subject alternate names can be specified for the new certificate. The subject name is always an X.500 directory name while a subject alternate name can be an X.500 directory name, a domain name, an email address, an IP address, or a uniform resource identifier. An X.500 directory name consists of common name, organization, and country attributes with optional organizational unit, city/locality, and state/province attributes. A domain name is one or more tokens separated by periods. An email address consists of a user name and a domain name that is separated by '@'. An IP address is an IPv4 address (nnn.nnn.nnn.nnn) or an IPv6 address (nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn). A uniform resource identifier consists of a scheme name, a domain name, and a scheme-specific portion; for example:

```
http://www.endicott.mycompany.com/main.html
```

Note: A self-signed end-entity certificate (server or client certificate) is not suggested for use in production environments and should only be used to facilitate test environments before production. Self-signed certificates do not imply any level of security or authenticity of the certificate because, as their name implies, they are signed by the same key that is contained in the certificate. However, certificates that are signed by a certificate authority indicate that, at least at the time of signature, the certificate authority approved the information that is contained in the certificate.

Import a certificate

This option will add the contents of the import file to a key database file or z/OS PKCS #11 token. The import file may contain one or more certificates without private keys. When each certificate is added to the key database, it is marked as trusted. The expiration date associated with each certificate cannot exceed February 6, 2106, at 23:59:59 UTC.

When adding certificates from the import file to a FIPS key database file only certificates signed with FIPS signature algorithms using FIPS-approved key sizes may be imported. When processing a chain of certificates, processing of the chain will terminate if a non-FIPS certificate is encountered. Certificates processed before the failing certificate will be added to the key database file. It is the responsibility of the importer to ensure that the file came from a FIPS source in order to maintain meeting FIPS 140- 2 criteria.

The import file must contain either an ASN.1 DER-encoded sequence as defined in RFC 2459 (tools.ietf.org/html/rfc2459), RFC 3280 (tools.ietf.org/html/rfc3280), RFC 5280 (tools.ietf.org/html/rfc5280), or a signed data message as defined in PKCS #7 (Cryptographic Message Syntax). The data can either be the binary value or the Base64 encoding of the binary value.

If the import file is in PKCS #7 format, only the first certificate and its certification chain will be imported. The certificate subject name will be used as the label for certificates added from the certification chain. A certification chain certificate will not be added to the database or z/OS PKCS #11 token if the label is not unique or if the certificate is already in the database or z/OS PKCS #11 token.

Base64 data is in the local code page. A DER-encoded sequence must start with the encoding header '-----BEGIN CERTIFICATE-----' and end with the encoding footer '-----END CERTIFICATE-----'. A PKCS #7 signed data message must start with the encoding header '-----BEGIN CERTIFICATE-----' and end with the encoding footer '-----END CERTIFICATE-----' or start with the encoding header '-----BEGIN PKCS #7 SIGNED DATA-----' and end with the encoding footer '-----END PKCS #7 SIGNED DATA-----'.

A root certificate is a self-signed certificate and is imported if the certificate is not already in the key database or z/OS PKCS #11 token.

An intermediate CA or end-entity certificate is a certificate signed by another entity. The key database or z/OS PKCS #11 token must already contain a certificate for the issuer. The certificate will not be imported if the certificate authenticity cannot be validated or if the database already contains the certificate.

An existing certificate can be replaced by specifying the label of the existing certificate. The issuer name, subject name, and subject public key in the new certificate must be the same as the existing certificate. If the existing certificate has a private key, the private key is not changed when the certificate is replaced.

Import a certificate and a private key

This option imports a certificate and the associated private key and adds it to the key database or z/OS PKCS #11 token. The certificate will be marked as trusted when it is added. When importing a certificate, the expiration date cannot exceed February 6, 2106, at 23:59:59 UTC.

The import file must contain an ASN.1 DER-encoded sequence as defined in PKCS #12 (Personal Information Exchange Syntax). The data can be either the binary value or the Base64 encoding of the binary value. Base64 data is in the local code page and must start with the encoding header '-----BEGIN CERTIFICATE-----' and end with the encoding footer '-----END CERTIFICATE-----'.

A root certificate is a self-signed certificate and is imported if the certificate is not already in the key database or z/OS PKCS #11 token.

An intermediate CA or end-entity certificate is a certificate signed by another entity. The key database or z/OS PKCS #11 token must already contain a certificate for the issuer. The certificate will not be imported if the certificate authenticity cannot be validated or if the database or z/OS PKCS #11 token already contains the certificate.

Each certificate in the certification chain will be imported if it is present in the import file. The certificate subject name will be used as the label for certificates added from the certification chain. A certification chain certificate will not be added to the database or z/OS PKCS #11 token if the label is not unique or if the certificate is already in the database or z/OS PKCS #11 token.

Only certificates and keys encoded according to PKCS #12 Version 3 and protected with strong encryption can be imported into a FIPS database. Furthermore, only certificates and keys comprising FIPS signature algorithms and using FIPS-approved key sizes may be imported into a FIPS database.

Show the default key

The private key information for the default key is displayed.

Store database password

The database password is masked and written to the key stash file. The file name is the same as the key database file name but has an extension of '.sth'.

gskkyman interactive mode examples

`gskkyman` can be run from either a rlogin z/OS shell environment or from the OMVS shell command-line environment. The examples that follow were performed from the rlogin environment. If you use the OMVS shell command-line environment, the only difference is that all input will be done at the command prompt on the screen.

These tasks will be performed in this topic:

- Creating, opening, and deleting a key database file
- Changing a key database password
- Storing an encrypted key database password
- Creating, opening, and deleting a z/OS PKCS #11 token
- Creating a self-signed server or client certificate
- Creating a certificate request and processing the signed request
- Creating a certificate to be used with Diffie-Hellman key exchange (key database only)
- Managing keys and certificates:
  - Show certificate/key information
  - Marking a certificate (and private key) as the default certificate for the key database
  - Copying a certificate (and private key) to a different key database or z/OS PKCS #11 Token:
    - Copying a certificate without its private key
    - Copying a certificate with its private key
    - Copying a certificate with its private key to a key database on the same system
    - Copying a certificate with its private key to another z/OS PKCS #11 token or key database on the same system
  - Removing a certificate (and private key) from a key database or z/OS PKCS #11 token
  - Changing a certificate label
- Importing a certificate from a file as a trusted CA certificate
- Importing a certificate from a file with its private key
- Using **gskkyman** to be your own certificate authority (CA) (key database only)
- Migrating key database files to RACF key rings (key database only)
- Migrating key database files to z/OS PKCS #11 tokens

Starting gskkyman

To start **gskkyman**, enter **gskkyman** at the command prompt (see Figure 11 on page 559).

Note: In the examples that follow, your input is shown in **bold**, and places where you press the Enter key are noted with **enter**.

Figure 11 on page 559 shows the **gskkyman** start menu.

```
# gskkyman <enter>

       Database Menu

   1  - Create new database
   1b - Create new empty database
   2  - Open database
   3  - Change database password
   4  - Change database record length
   5  - Delete database
   6  - Create key parameter file
   7  - Display certificate file (Binary or Base64 ASN.1 DER)

  11  - Create new token
  12  - Delete token
  13  - Manage token
  14  - Manage token from list of tokens

   0  - Exit program

Enter option number:
```

Figure 11. Starting Menu for gskkyman

From the **Database Menu** for **gskkyman**, you can create a new key database, create new empty database, open an existing key database, display the contents of a certificate file, change a database password, change a database record length, delete a database, create, delete, and manage a z/OS PKCS #11 token, or exit **gskkyman**.

Creating, opening, and deleting a key database file

To create a new key database, enter **1** or **1b** at the command prompt on the **Database Menu**:

```
       Database Menu

   1  - Create new database
   1b - Create new empty database
   2  - Open database
   3  - Change database password
   4  - Change database record length
   5  - Delete database
   6  - Create key parameter file
   7  - Display certificate file (Binary or Base64 ASN.1 DER)

  11  - Create new token
  12  - Delete token
  13  - Manage token
  14  - Manage token from list of tokens

   0  - Exit program

Enter option number: 1 <enter>
Enter key database name (press ENTER to return to menu): mykey.kdb <enter>
Enter database password (press ENTER to return to menu): <enter password>
Re-enter database password: <enter password>
Enter password expiration in days (press ENTER for no expiration): 35 <enter>
Enter database record length (press ENTER to use 5000): <enter>

Enter 1 for FIPS mode database or 0 to continue: 0 <enter>


Key database /home/sufwl1/ssl_cmd/mykey.kdb created.

Press ENTER to continue.
```

Figure 12. Creating a New Key Database

Figure 12 on page 560 shows the input prompts that **gskkyman** produces when you choose **1** or **1b** to create a new key database. As you can see, default choices are listed in parentheses. In the example, by pressing the Enter key at the **Enter database record length** prompt, the default of 5000 was chosen.

Note:

1. When dealing with certificates which may be large or have large key sizes, for example 2048 or 4096, an initial key record length of 5000 may be required.
2. The maximum length of the password specified for a key database file is 128 characters.
3. When creating a new key database file, you will be prompted whether you want a FIPS or non-FIPS database file created. For more information about FIPS mode databases, see "Key database files" on page 545.

After entering the database record length, a message displays confirming that your database was created (see Figure 12 on page 560). You are prompted to press Enter to continue. Doing so displays the **Key Management Menu** for the database you have created:

```
        Key Management Menu

        Database: /home/sufwl1/ssl_cmd/mykey.kdb
        Expiration Date: 2025/12/02  10:11:12
        Type: non-FIPS

    1 - Manage keys and certificates
    2 - Manage certificates
    3 - Manage certificate requests
    4 - Create new certificate request
    5 - Receive requested certificate or a renewal certificate
    6 - Create a self-signed certificate
    7 - Import a certificate
    8 - Import a certificate and a private key
    9 - Show the default key
   10 - Store database password
   11 - Show database record length

    0 - Exit program

Enter option number (press ENTER to return to previous menu):
```

Figure 13. Key Management Menu for gskkyman

Figure 13 on page 561 shows the **Key Management Menu**. Entering **0** at this prompt exits the **gskkyman** program. Pressing Enter at the prompt returns you to the **Database Menu**.

To open an existing key database file, on the **Database Menu**, enter option number **2** (see Figure 14 on page 561). You are then prompted for the key database name and password.

Note: Do not lose the key database password. There is no method to reset this password if you lose or forget the password. If the password is lost, the private keys stored in the key database are inaccessible, therefore, unusable.

```
        Database Menu

    1  - Create new database
    1b - Create new empty database
    2  - Open database
    3  - Change database password
    4  - Change database record length
    5  - Delete database
    6  - Create key parameter file
    7  - Display certificate file (Binary or Base64 ASN.1 DER)

   11  - Create new token
   12  - Delete token
   13  - Manage token
   14  - Manage token from list of tokens

    0  - Exit program

Enter option number: 2 <enter>
Enter key database name (press ENTER to return to menu): mykey.kdb <enter>
Enter database password (press ENTER to return to menu): <enter password>
```

Figure 14. Opening an Existing Key Database File

The key database name is the file name of the key database. The input file name is interpreted relative to the current directory when `gskkyman` is invoked. You may also specify a fully qualified key database name.

After you enter the key database name and password, the **Key Management Menu** displays for the database you have selected to open, (see Figure 15 on page 562).

```
        Key Management Menu

        Database: /home/sufwl1/ssl_cmd/mykey.kdb
        Expiration Date: 2025/12/02  10:11:12
        Type: non-FIPS

    1 - Manage keys and certificates
    2 - Manage certificates
    3 - Manage certificate requests
    4 - Create new certificate request
    5 - Receive requested certificate or a renewal certificate
    6 - Create a self-signed certificate
    7 - Import a certificate
    8 - Import a certificate and a private key
    9 - Show the default key
   10 - Store database password
   11 - Show database record length

    0 - Exit program

Enter option number (press ENTER to return to previous menu):
```

Figure 15. Key Management Menu

To delete an existing database, from the **Database Menu**, select option **5** (see Figure 16 on page 562):

```
        Database Menu

    1  - Create new database
    1b - Create new empty database
    2  - Open database
    3  - Change database password
    4  - Change database record length
    5  - Delete database
    6  - Create key parameter file
    7  - Display certificate file (Binary or Base64 ASN.1 DER)

   11  - Create new token
   12  - Delete token
   13  - Manage token
   14  - Manage token from list of tokens

    0  - Exit program

Enter option number: 5 <enter>
Enter key database name (press ENTER to return to menu): mykey.kdb <enter>

Enter 1 to confirm delete, 0 to cancel delete: 1 <enter>

Key database /home/sufwl1/ssl_cmd/mykey.kdb deleted.

Press ENTER to continue.
```

Figure 16. Deleting an Existing Key Database

You are prompted to enter the key database name that you want to delete. Then you must enter **1** to confirm the delete, or **0** to cancel the delete. If you choose **1**, a message displays to confirm the file has been deleted.

Note: If you delete an existing key database, the associated request database and database password stash file (if existent) is also deleted. It's important to note that anyone with write access to a key database can delete that database either by removing it with the **rm** command or by using **gskkyman** subcommand.

Changing a key database password

You can change a key database password. From the **Database Menu**, select option **3**:

```
       Database Menu

   1  - Create new database
   1b - Create new empty database
   2  - Open database
   3  - Change database password
   4  - Change database record length
   5  - Delete database
   6  - Create key parameter file
   7  - Display certificate file (Binary or Base64 ASN.1 DER)

  11  - Create new token
  12  - Delete token
  13  - Manage token
  14  - Manage token from list of tokens

   0  - Exit program

Enter option number: 3 <enter>
Enter key database name (press ENTER to return to menu): mykey.kdb <enter>
Enter database password (press ENTER to return to menu): <enter current password>
Enter new database password (press ENTER to return to menu): <enter new password>
Re-enter database password: <enter new password>
Enter password expiration in days (press ENTER for no expiration): <enter>

Database password changed.

Press ENTER to continue.
```

Figure 17. Changing a Key Database Password

Figure 17 on page 563 displays the prompts you are given. You first enter your current password. Then you select a new password, and enter it again to confirm. You can choose your password expiration in days or press Enter to have no expiration. A message displays to confirm the transaction.

Storing an encrypted key database password

In order for applications to use the key database file, the application must specify both the file name and its associated password. The password can either be specified directly or through a stash file containing the encrypted password. The stash file provides a level of security where the password does not have to be explicitly specified. To save the encrypted key database password, enter option **10** from the **Key Management Menu**:

Note: In these task descriptions, it is assumed that you opened the key database and are displaying the **Key Management Menu** panel.

```
       Key Management Menu

       Database: /home/sufw1/ssl_cmd/mykey.kdb
       Expiration Date: 2025/12/02  10:11:12
       Type: non-FIPS

   1 - Manage keys and certificates
   2 - Manage certificates
   3 - Manage certificate requests
   4 - Create new certificate request
   5 - Receive requested certificate or a renewal certificate
   6 - Create a self-signed certificate
   7 - Import a certificate
   8 - Import a certificate and a private key
   9 - Show the default key
  10 - Store database password
  11 - Show database record length

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 10 <enter>

Database password stored in /home/sufwl1/ssl_cmd/mykey.sth.

Press ENTER to continue.
```

Figure 18. Key Management Menu

Figure 18 on page 564 shows the message you receive after entering option **10** to store the database password. In this example, the database password was stored in a file called `mykey.sth`.

Creating, opening, and deleting a z/OS PKCS #11 token

To create a z/OS PKCS #11 token, enter **11** at the command prompt on the **Database Menu**:

```
       Database Menu

   1  - Create new database
   1b - Create new empty database
   2  - Open database
   3  - Change database password
   4  - Change database record length
   5  - Delete database
   6  - Create key parameter file
   7  - Display certificate file (Binary or Base64 ASN.1 DER)

  11  - Create new token
  12  - Delete token
  13  - Manage token
  14  - Manage token from list of tokens

   0  - Exit program

Enter option number: 11 <enter>
Enter token name (press ENTER to return to menu): TOKEN1 <enter>

Token successfully created

Press ENTER to continue.
```

Figure 19. Creating a z/OS PKCS #11 token

The only input required when creating a new z/OS PKCS #11 token is the token name.

Note: Only users with SAF access level of UPDATE or CONTROL to the CRYPTOZ resource "so.tokenname" have the authority to create the z/OS PKCS #11 token with the name "tokenname".

Note: A z/OS PKCS #11 token contains no certificates or keys when first created.

After entering the token name, a message displays confirming that the z/OS PKCS #11 token was created (see Figure 19 on page 564). You are prompted to press Enter to continue. Doing so re-displays the **Database Menu**.

To open an existing z/OS PKCS #11 token, enter either option **13** or option **14** on the **Database Menu**. If option **13** is used:

```
       Database Menu

   1  - Create new database
   1b - Create new empty database
   2  - Open database
   3  - Change database password
   4  - Change database record length
   5  - Delete database
   6  - Create key parameter file
   7  - Display certificate file (Binary or Base64 ASN.1 DER)

  11  - Create new token
  12  - Delete token
  13  - Manage token
  14  - Manage token from list of tokens

   0  - Exit program

Enter option number: 13 <enter>
Enter token name (press ENTER to return to menu): TOKEN1 <enter>
```

Figure 20. Opening a z/OS PKCS #11 token from token name

If option **14** is used:

```
       Database Menu

   1  - Create new database
   1b - Create new empty database
   2  - Open database
   3  - Change database password
   4  - Change database record length
   5  - Delete database
   6  - Create key parameter file
   7  - Display certificate file (Binary or Base64 ASN.1 DER)

  11  - Create new token
  12  - Delete token
  13  - Manage token
  14  - Manage token from list of tokens

   0  - Exit program

Enter option number: 14 <enter>
```

```
    Token List

   1 - TOKEN1

   0 - Return to selection menu

Enter list-entry number (press ENTER to return to previous menu): 1 <enter>
```

Figure 21. Opening a z/OS PKCS #11 token from token list

After either entering the token name (if option **13** used) or selecting the token from a list of tokens (if option **14** is used), the **Token Management Menu** displays the z/OS PKCS #11 token selected.

```
        Token Management Menu

        Token: TOKEN1

        Manufacturer:  z/OS PKCS11 API
        Model:  HCR77D2
        Flags:  0x00000509 (INITIALIZED,PROT AUTH PATH,USER PIN INIT,RNG)

    1 - Manage keys and certificates
    2 - Manage certificates
    3 - Manage certificate requests
    4 - Create new certificate request
    5 - Receive requested certificate or a renewal certificate
    6 - Create a self-signed certificate
    7 - Import a certificate
    8 - Import a certificate and a private key
    9 - Show the default key
   10 - Delete token

    0 - Exit program

Enter option number (press ENTER to return to previous menu):
```

Figure 22. Token Management Menu

Note: Only users with SAF access level of READ, UPDATE, or CONTROL to the CRYPTOZ resource "so.tokenname" or "user.token.name" have the authority to open the z/OS PKCS #11 token with the name "tokenname".

To delete an existing z/OS PKCS #11 token, enter either option **12** on the **Database Menu**, or select option **10** from the **Token Management Menu.**

If option **12** on the **Database Menu** is used:

```
       Database Menu

   1  - Create new database
   1b - Create new empty database
   2  - Open database
   3  - Change database password
   4  - Change database record length
   5  - Delete database
   6  - Create key parameter file
   7  - Display certificate file (Binary or Base64 ASN.1 DER)

  11  - Create new token
  12  - Delete token
  13  - Manage token
  14  - Manage token from list of tokens


   0  - Exit program


Enter option number:12 <enter>
Enter token name (press ENTER to return to menu):TOKEN1 <enter>
To confirm token delete, enter token name again (press ENTER to cancel delete):TOKEN1 <enter>

Token successfully deleted

Press ENTER to continue.
```

Figure 23. Deleting an existing z/OS PKCS #11 Token

If option **10** on the **Token Management Menu** is used:

```
        Token Management Menu

        Token: TOKEN1

        Manufacturer:  z/OS PKCS11 API
        Model:  HCR77D2
        Flags:  0x00000509 (INITIALIZED, PROT AUTH PATH, USER PIN INIT, RNG)

    1 - Manage keys and certificates
    2 - Manage certificates
    3 - Manage certificate requests
    4 - Create new certificate request
    5 - Receive requested certificate or a renewal certificate
    6 - Create a self-signed certificate
    7 - Import a certificate
    8 - Import a certificate and a private key
    9 - Show the default key
   10 - Delete token


    0 - Exit program


Enter option number (press ENTER to return to previous menu): 10 <enter>
To confirm token delete, enter token name again (press ENTER to cancel delete): TOKEN1 <enter>

Token successfully deleted

Press ENTER to continue.
```

Figure 24. Deleting an existing z/OS PKCS #11 token

Using either approach you are prompted to enter the token name in order to confirm the correct token is deleted. A message is displayed to confirm that the z/OS PKCS #11 token has been deleted. The token does not have to be empty before performing the delete.

Note: Only users with SAF access level of UPDATE or CONTROL to the CRYPTOZ resource "so.tokenname" have the authority to delete the z/OS PKCS #11 token with the name "tokenname".

Creating a self-signed server or client certificate

If your organization does not use a certificate authority (within the organization or outside the organization), a self-signed certificate can be generated for use by the program acting as an SSL server or client. In addition, since root CA certificates are also self-signed certificates that are permitted to be used to sign other certificates (certificate requests), these procedures can also be used to create a root CA certificate. See "Marking a certificate (and private key) as the default certificate" on page 580.

Programs acting as SSL servers (i.e. acting as the server side of the SSL handshake protocol) must have a certificate to use during the handshake protocol. A program acting as an SSL client requires a certificate when the SSL server requests client authentication as part of the SSL handshake.

Note: This is not suggested for production environments and should only be used to facilitate test environments before production. Self-signed certificates do not imply any level of security or authenticity of the certificate because, as their name implies, they are signed by the same key that is contained in the certificate. However, certificates that are signed by a certificate authority indicate that, at least at the time of signature, the certificate authority approved the information contained in the certificate.

Note: gskkyman supports the creation of X.509 Version 3 certificates.

When creating a self-signed certificate to be used to identify a server or client, from the **Key Management Menu** or **Token Management Menu**, enter **6**. You are prompted for a number of items to define the certificate, including the intended use of the certificate, the key algorithm and key size, and possibly the digest algorithm for the certificate signature.

```
       Key Management Menu

       Database: /home/sufwl1/ssl_cmd/mykey.kdb
       Expiration Date: 2025/12/02  10:11:12
       Type: non-FIPS

   1 - Manage keys and certificates
   2 - Manage certificates
   3 - Manage certificate requests
   4 - Create new certificate request
   5 - Receive requested certificate or a
       renewal certificate
   6 - Create a self-signed certificate
   7 - Import a certificate
   8 - Import a certificate and a private key
   9 - Show the default key
  10 - Store database password
  11 - Show database record length

   0 - Exit program

Enter option number (press ENTER to return to
previous menu): 6 <enter>
```

Figure 25. Key Management Menu

```
       Token Management Menu

       Token: TOKEN1

       Manufacturer:  z/OS PKCS11 API
       Model:  HCR77D2
       Flags:  0x00000509 (INITIALIZED,PROT AUTH
               PATH,USER PIN INIT,RNG)

   1 - Manage keys and certificates
   2 - Manage certificates
   3 - Manage certificate requests
   4 - Create new certificate request
   5 - Receive requested certificate or a renewal certificate
   6 - Create a self-signed certificate
   7 - Import a certificate
   8 - Import a certificate and a private key
   9 - Show the default key
  10 - Delete token

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 6 <enter>
```

Figure 26. Token Management Menu

Certificates that are intended to be used directly by a server or client are considered to be end user certificates. Certificates intended to be used to sign other certificates are considered to be CA certificates. RSA key certificates are the most common. DSA key certificates represent certificates that follow the FIPS-186 government standard. ECC key certificates represent certificates that use Elliptic Curve Cryptography. The larger the key size, the more secure the generated key will be. Note that CPU usage increases as the key size increases.

If an RSA certificate is selected, you will be prompted to select the key size and the digest type for the signature algorithm. See Figure 27 on page 569 for an example of selecting the key size and digest type.

If a 1024-bit DSA certificate is selected, SHA-1 will be used for the signature algorithm. If a 2048-bit DSA certificate is selected, you will be prompted to select the digest type for the signature algorithm from a list of SHA-based digest types.

If an ECC certificate is selected, you will be prompted to select the ECC key type and curve type. The suggested digest for the key size of the ECC key will be used for the signature algorithm, as specified in Table 2 on page 13. See "Creating a signed ECC certificate and key" on page 590 for more information.

If an RSA certificate with a RSASSA-PSS signature is selected, you are prompted to select the key size. The suggested digest for the key size of the RSA key is used for the signature algorithm, as specified in Table 4 on page 14.

Once the certificate type and signature algorithm is determined, you will be prompted to enter:

- a label to uniquely identify the key and certificate within the key database
- the individual fields within the subject name
- certificate expiration. The valid expiration range is 1 to 9999 days. The default value is 365 days. The expiration date cannot exceed February 6, 2106, at 23:59:59 UTC.
- the subject alternate names (optional)

Figure 27 on page 569 shows the creation of a self-signed certificate to be used as a server or client certificate in a key database file or z/OS PKCS #11 token.

```
      Certificate Usage

   1 - CA certificate
   2 - User or server certificate

Select certificate usage (press ENTER to return to menu): 2 <enter>

       Certificate Key Algorithm

   1 - Certificate with an RSA key
   2 - Certificate with a DSA key
   3 - Certificate with an ECC key
   4 – Certificate with an RSA key and RSASSA-PSS signature algorithm

Select certificate key algorithm (press ENTER to return to menu): 1 <enter>

       RSA Key Size

   1 - 1024-bit key
   2 - 2048-bit key
   3 - 4096-bit key

Select RSA key size (press ENTER to return to menu): 2 <enter>

       Signature Digest Type

   1 - SHA-1
   2 - SHA-224
   3 - SHA-256
   4 - SHA-384
   5 - SHA-512

Select digest type (press ENTER to return to menu): 3 <enter>
Enter label (press ENTER to return to menu): Server Cert <enter>
Enter subject name for certificate
  Common name (required): My Server Certificate <enter>
  Organizational unit (optional): ID <enter>
  Organization (required): IBM <enter>
  City/Locality (optional): Endicott <enter>
  State/Province (optional): NY <enter>
  Country/Region (2 characters - required): US <enter>
Enter number of days certificate will be valid (default 365): 244 <enter>

Enter 1 to specify subject alternate names or 0 to continue: 0 <enter>

Please wait .....

Certificate created.

Press ENTER to continue.
```

Figure 27. Creating a Self-Signed Certificate

Once the certificate is created, the next step is to determine whether the certificate should be marked as the database's or z/OS PKCS #11 tokens default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label. For more information about setting the default certificate, see "Marking a certificate (and private key) as the default certificate" on page 580.

In order for the SSL handshake to successfully validate the use of the self-signed certificates, the partner application needs to know about the signer of the certificate. For self-signed certificates, this means that the self-signed certificate must be imported into the partner's database or z/OS PKCS #11 token. For more information about importing certificates, see "Importing a certificate from a file as a trusted CA certificate" on page 594.

Creating a certificate request

A program may require a certificate, associated with itself, depending on what side of the SSL connection the program is running. This requirement also depends on whether ***client authentication*** is requested as part of the SSL handshake. Programs acting as SSL servers (act as the server side of the SSL handshake protocol) `must` have a certificate to use during the handshake protocol. A program acting as an SSL client requires a certificate in the key database if the SSL server requests ***client authentication*** as part of the SSL handshake operation. The way in which certificates are used within an organization will determine whether you need to create a certificate request. If the organization chooses to use a certificate authority (within the organization or outside of the organization), then you must generate a certificate request.

To create a certificate request, enter **4** from the **Key Management Menu** or **Token Management Menu**.

```
       Key Management Menu

       Database: /home/sufwl1/ssl_cmd/mykey.kdb
       Expiration Date: 2025/12/02  10:11:12
       Type: non-FIPS

   1 - Manage keys and certificates
   2 - Manage certificates
   3 - Manage certificate requests
   4 - Create new certificate request
   5 - Receive requested certificate or a renewal certificate
   6 - Create a self-signed certificate
   7 - Import a certificate
   8 - Import a certificate and a private key
   9 - Show the default key
  10 - Store database password
  11 - Show database record length

   0 - Exit program

Enter option number (press ENTER to return to previous menu):  4 <enter>
```

Figure 28. Creating a certificate request-Key Management Menu

```
       Token Management Menu

       Token: TOKENABC

       Manufacturer:  z/OS PKCS11 API
       Model:  HCR77D2
       Flags:  0x00000509 (INITIALIZED,PROT AUTH
               PATH,USER PIN INIT,RNG)

   1 - Manage keys and certificates
   2 - Manage certificates
   3 - Manage certificate requests
   4 - Create new certificate request
   5 - Receive requested certificate or a renewal certificate
   6 - Create a self-signed certificate
   7 - Import a certificate
   8 - Import a certificate and a private key
   9 - Show the default key
  10 - Delete token

   0 - Exit program

Enter option number (press ENTER to return to previous menu):  4 <enter>
```

Figure 29. Creating a certificate request-Key Management Menu

When creating a certificate request, you are prompted to select the key algorithm and the key size for the certificate to be requested. RSA key certificates are the most common. DSA key certificates represent certificates that follow the FIPS-186 government standard. ECC key certificates represent certificates that use Elliptic Curve Cryptography. The larger the key size, the more secure the encryption/decryption generated key is.

If an RSA certificate is selected, you are prompted to select the digest type for the signature algorithm from a list of SHA-based digest types.

If a 1024-bit DSA certificate is selected, SHA-1 is used for the signature algorithm. If a 2048-bit DSA certificate is selected, you are prompted to select the digest type for the signature algorithm from a list of SHA-based digest types.

If an ECC certificate is selected, you will be prompted to select the ECC key type and curve type. The suggested digest for the key size of the ECC key is used for the signature algorithm, as specified in Table 2 on page 13.

If an RSA certificate with a RSASSA-PSS signature is selected, you are prompted to select the key size. The suggested digest for the key size of the RSA key is used for the signature algorithm, as specified in Table 4 on page 14.

After the certificate type is determined, you will be prompted to enter:

- a request file name to store the certificate request
- a label to uniquely identify the certificate request within the key database
- the individual fields within the subject name
- the individual fields within the subject alternate name (optional).

The **Certificate Key Algorithm menu** appears:

```
       Certificate Key Algorithm

   1 - Certificate with an RSA key
   2 - Certificate with a DSA key
   3 - Certificate with an ECC key
   4 - Certificate with a Diffie-Hellman key
   5 - Certificate with an RSA key and RSASSA-PSS signature


Select certificate key algorithm (press ENTER to return to menu): 1 <enter>

       RSA Key Size

   1 – 1024-bit key
   2 – 2048-bit key
   3 – 4096-bit key

Select RSA key size (press ENTER to return to menu): 2 <enter>

       Signature Digest Type

   1 - SHA-1
   2 - SHA-224
   3 - SHA-256
   4 - SHA-384
   5 - SHA-512

Select digest type (press ENTER to return to menu): 3 <enter>
Enter request file name (press ENTER to return to menu): certreq.arm <enter>
Enter label (press ENTER to return to menu): Test Server Cert <enter>
Enter subject name for certificate
  Common name (required): Test Server <enter>
  Organizational unit (optional): ID <enter>
  Organization (required): IBM <enter>
  City/Locality (optional): Endicott <enter>
  State/Province (optional): NY <enter>
  Country/Region (2 characters - required): US <enter>

Enter 1 to specify subject alternate names or 0 to continue: 0 <enter>

Please wait .....

Certificate request created.

Press ENTER to continue.
```

Figure 30. Creating a Certificate Request

Enter option **0** to continue or option **1** to specify the subject alternate names. If option **1** is selected, the **Subject Alternate Name Type** menu appears:

```
       Subject Alternate Name Type

   1 - Directory name (DN)
   2 - Domain name (DNS)
   3 - E-mail address (SMTP)
   4 - Network address (IP)
   5 - Uniform resource identifier (URI)

Select subject alternate name type (press ENTER if name is complete): 1 <enter>
Enter subject name for certificate
  Common name (required): Test server <enter>
  Organizational unit (optional):  IBM <enter>
  Organization (required):  IBM <enter>
  City/Locality (optional): Endicott <enter>
  State/Province (optional): NY <enter>
  Country/Region (2 characters - required): US <enter>

       Subject Alternate Name Type

   1 - Directory name (DN)
   2 - Domain name (DNS)
   3 - E-mail address (SMTP)
   4 - Network address (IP)
   5 - Uniform resource identifier (URI)

Select subject alternate name type (press ENTER if name is complete): <enter>

Please wait .....
```

Figure 31. Specifying subject alternate names

When specifying subject alternate names, you are prompted for the type of the alternate name. After the alternate name type is determined, you will be prompted to enter:

- the individual fields within the subject name.

After the individual fields are completed, press enter to continue or select one of the subject alternate name types. Repeat the process.

Once the certificate request (and associated subject alternate names) is created, a file with the name you specified will exist in the current working directory or directory specified in the file name. If you choose to exit gskkyman, the program ends. Otherwise, the **Key Management Menu** or the **Token Management Menu** (see Key Management Menu) displays, allowing additional operations to be performed.

The certificate request created is stored in a file that is in base64-encoded format. This format is what is typically required by certificate authorities that create certificates. This is the contents of the file created by the steps performed in Creating a Certificate Request:

```
$ cat certreq.arm<enter>
-----BEGIN NEW CERTIFICATE REQUEST-----
MIICozCCAYsCAQAwXjELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAk5ZMREwDwYDVQQH
EwhFbmRpY290dDEMMAoGA1UEChMDSUJNMQswCQYDVQQLEwJJRDEUMBIGA1UEAxML
VGVzdCBTZXJ2ZXIwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDEDtsG
3q94S6w7UxLqXyaTdzsP9cYPeeFlIZRIqXHFDJMH4uGTC5NnOIfuP5X1kCemAroo
2O5CKdXFI7fo+y1uZCtgzVpByFu4AK/dhQA/N1CZI2LgdnySl4p40k0dM/l97E+Y
7scqmBkPLE0U8fQU3Z0fKNuZeS1xkkJQXDBuGKnkiEToFX1kgHsGfMOQuzUAKYw8
VRCwW38ZOJdGHDQusT9r0dm1oXM413UMPejF9DU2K9NqaKptrgc3+W+GCuzbokdW
zHGucovG+4bKKGQNzt+dWccZ1/RlHIAgkWQ+x4B74UYZRCxgewb8JmeTX+d1KWgU
dscR/ch3fQ26eex9AgMBAAGgADANBgkqhkiG9w0BAQsFAAOCAQEAp0hjxyAJhPSO
g5CToark8JaxmA9V1PlHwb6Bk/DYBQcu6QB3QZo4kBivpBrdwfKd2SvKVKnUMuWk
2IA1J7IY8hIshLM5NlbALPfFey2bNCYfPujG6l1s6wzEp3FVrF0J5z0Km60LrjZ3
1ijlSlUVH5Mi6HumncSq1cI1kj57PD8OTy/ZcVpL7/8wleitqw2ltWOvgyOZJzS5
rrZEfoiRtwd7L1gZnodvWCrT5kQaZkqhcNcpHHM4vsxDoy2aeIV2bTtHeg4lBvRy
jZJri/4EzQVmcbEZlEdk1rISyH1K9Y9OXH2BcaKFsD+RwcYZhCSo0spGK/CsDxPx
5JEEQeAgrg==
-----END NEW CERTIFICATE REQUEST-----
$
```

Figure 32. Contents of certreq.arm after Certificate Request Generation

Displaying a newly created certificate request

After creating a certificate request, you can display the contents of the certificate request by entering **3** (Manage certificate requests) from the **Key Management Menu** or the **Token Management Menu.** This displays the **Request List** or the **Token Certificate Request List.** Select the number corresponding to the label associated with the certificate request that you would like to display. Enter **1** (Show key information) from the **Request Menu** or the **Token Certificate Request Menu**. This option displays information about the private key associated with the certificate request along with the certificate request subject name and optional subject alternate name values. If there are no subject alternate names associated with the request, the subject alternate name field will not appear.

```
                       Key Information
                   Label: Cert Request
               Record ID: 13
        Issuer Record ID: 13
             Default key: No
   Private key algorithm: rsaEncryption
        Private key size: 2048
            Subject name: My Certificate Request
                          ID
                          IBM
                          Poughkeepsie
                          NY
                          US
  Subject alternate name: DNS: mydns.mycompany.com
                          EMAIL: myemail@mycompany.com
                          IPV4: 9.9.9.9

Press ENTER to continue.
```

Figure 33. Key information for certificate request

Show database record length

The database record length is displayed. All records in the database have the same length and a database entry cannot span a database record.

Sending the certificate request

The certificate request file can either be transferred to another system (for example, FTP as an ASCII text file) and then transferred to the certificate authority or placed directly into a mail message sent to a certificate authority using cut-and-paste methods.

In addition to the certificate request file that is generated, a request database (.rdb) file is also created or altered. The request database is named the same as the key database file, except it has an extension of .rdb. For example, a key database file of key.kdb causes a request database file of key.rdb to be created. This request database file must be saved along with the key database in order for the response for the certificate request to be successfully processed.

The certificate request must not be deleted from the database while the request is being processed by the signing certificate authority. The database certificate request is required for applicable processing when the signed certificate from the certificate authority is received. The removal of the certificate request from the database causes the private key associated with the certificate request to be lost.

Receiving the signed certificate or renewal certificate

When a certificate is signed by the certificate authority in response to the certificate request, you must receive it into the key database or z/OS PKCS #11 token. This is for new certificates and renewal certificates.

To receive the certificate, you must store the Base64-encoded certificate in a file on the z/OS system to be read in by the **gskkyman** utility. This file should be in the current working directory when **gskkyman** is started. If this file is on another working directory, you must specify the fully qualified name.

Note: To receive the certificate, the CA certificate must also exist in the key database or z/OS PKCS #11 token. To store a CA certificate, see "Importing a certificate from a file as a trusted CA certificate" on page 594.

To receive a certificate that is issued on your behalf, from the **Key Management Menu** or **Token Management Menu**, see Key Management Menu and enter option **5**.

```
       Key Management Menu

       Database: /home/sufwl1/ssl_cmd/mykey.kdb
       Expiration Date: 2025/12/02  10:11:12
       Type: non-FIPS

   1 - Manage keys and certificates
   2 - Manage certificates
   3 - Manage certificate requests
   4 - Create new certificate request
   5 - Receive requested certificate or a renewal certificate
   6 - Create a self-signed certificate
   7 - Import a certificate
   8 - Import a certificate and a private key
   9 - Show the default key
  10 - Store database password
  11 - Show database record length

   0 - Exit program

Enter option number (press ENTER to return to previous menu):  5 <enter>

Enter certificate file name (press ENTER to return to menu):  signed.arm <enter>

Certificate received.

Press ENTER to continue.
```

Figure 34. Key Management Menu

```
      Token Management Menu

      Token: TOKENABC

      Manufacturer:  z/OS PKCS11 API
      Model:  HCR77D2
      Flags:  0x00000509 (INITIALIZED,PROT AUTH
              PATH,USER PIN INIT,RNG)

   1 - Manage keys and certificates
   2 - Manage certificates
   3 - Manage certificate requests
   4 - Create new certificate request
   5 - Receive requested certificate or a renewal certificate
   6 - Create a self-signed certificate
   7 - Import a certificate
   8 - Import a certificate and a private key
   9 - Show the default key
  10 - Delete token

   0 - Exit program

Enter option number (press ENTER to return to previous menu):  5 <enter>

Enter certificate file name (press ENTER to return to menu):  signed.arm <enter>

Certificate received.

Press ENTER to continue.
```

Figure 35. Token Management Menu

You are prompted for the name of the file that contains the Base64-encoded certificate that was returned to you by the certificate authority in response to a previously submitted certificate request (See "Creating a certificate request" on page 570). After you receive the certificate, press Enter to continue working with the **Key Management Menu** or **Token Management Menu**. Upon completion of this step and before the System SSL APIs using the certificate during the SSL handshake processing, you must determine whether the certificate should be marked as the database's default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label. For more information about setting the default certificate, see "Marking a certificate (and private key) as the default certificate" on page 580.

When received into a key database file, the certificate's expiration date should be monitored. When the expiration date is nearing (do not wait until it is expired), a new certificate should be obtained to replace the existing certificate. The new certificate can be a brand new certificate with new public/private keys or a renewal certificate where existing keys and certificate information is used. See "Creating a certificate request" on page 570 for more information about a new or renewal certificate.

Managing keys and certificates

When certificates are added to the key database or z/OS PKCS #11 token, these are some common operations that can be performed with the certificates:

- Show certificate/key information
- Mark a certificate (and private key) as the default certificate for the key database or z/OS PKCS #11 token
- Export a certificate to a file, key database, or z/OS PKCS #11 token
- Remove a certificate (and private key) from a key database or z/OS PKCS #11 token
- Change a certificate label
- Create a signed ECC certificate and key
- Create a certificate to be used with a fixed Diffie-Hellman key exchange
- Create a certificate renewal request

Showing certificate/key information

It is sometimes useful to display the information contained in the certificates that are stored in the key database. The information displayed includes, among others, the label, issuer/subject name, the version number of the certificate, the key size for the public/private key pair, and the expiration date.

To list information about certificates that contain private keys, from the **Key Management Menu** or **Token Management Menu** (see Key Management Menu) select **1**, (Manage keys and certificates). This displays the **Key and Certificate List**.

```
       Key and Certificate List

       Database: /home/sufwl1/ssl_cmd/mykey.kdb

   1 - Test Server Cert
   2 - Server Cert

   0 - Return to selection menu

Enter label number (ENTER to return to selection menu, p for previous list): 2 <enter>
```

Figure 36. Key and Certificate List

```
       Token Key and Certificate List

       Token: TOKENABC

   1 - Test Server Cert
   2 - Server Cert

   0 - Return to selection menu

Enter label number (ENTER to return to selection menu, p for previous list): 2 <enter>
```

Figure 37. Token Key and Certificate List

Select the number corresponding to the label for which you would like to display certificate/key information. The **Key and Certificate Menu** for the label you chose displays next (see Figure 38 on page 577).

```
       Key and Certificate Menu

       Label: Server Cert

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database/token
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate and key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 1 <enter>
```

Figure 38. Key and Certificate Menu

```
      Token Key and Certificate Menu

      Label: Server Cert

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database/token
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate and key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 1 <enter>
```

Figure 39. Token Key and Certificate Menu

On the **Key and Certificate Menu** or the **Token Key and Certificate Menu**, you could choose **1** to display certificate information. This accesses the **Certificate Information** menu (see Figure 40 on page 578):

```
                       Certificate Information

              Label: RSASSA_PSS_CA
          Record ID: 8
   Issuer Record ID: 8
            Trusted: Yes
            Version: 3
      Serial number: 5994505b0008ac16
        Issuer name: RSASSA_PSS_CA
                     IBM
                     US
       Subject name: RSASSA_PSS_CA
                     IBM
                     US
     Effective date: 2017/08/16
    Expiration date: 2018/08/16
  Signature algorithm: rsassa-pss
     Digest Algorithm: sha256Digest
       Mask Algorithm: Mgf1
    Issuer unique ID: None
   Subject unique ID: None
 Public key algorithm: rsaEncryption
      Public key size: 2048
           Public key: 30 82 01 0A 02 82 01 01 00 C5 2F 58 B6 FE A2 66
                       75 64 D9 AF DD 85 B3 89 6D C7 CD 5B 36 99 EF 74
                       93 A1 B6 ED 2E 34 6C E8 7C 04 8A 76 47 23 16 07
                       DB AA 28 46 20 29 28 CB F7 04 2A F8 AA A1 AF 0E
                       78 7A 6B 01 FF 20 D8 7F B9 5B E8 CE ED 32 86 05
                       A7 5E C7 71 6B CB 22 5F 7A 05 82 84 A2 D5 87 7C
                       4E 96 29 82 A2 96 61 6B 59 DB C4 EC 7B 43 DE DA
                       AF 5C 2A 9C FA B9 28 C0 1B 0F 1F F8 EE 40 9F 7E
                       E7 29 B6 16 C8 5F 94 93 B8 04 A9 B1 BB 43 42 B4
                       2B 32 37 C0 A0 60 6E F2 E4 56 B8 F0 D7 F6 5B D6
                       D2 FC 45 DD D8 7E 7D 73 95 30 5B 4D B0 69 3C 16
                       D9 03 14 E2 B4 56 3F 1B 04 C4 78 A2 9C 58 08 97
                       75 0C 8A 30 91 94 A1 31 1A EA FB 9C 71 30 24 8A
                       A9 BD 8A A9 C6 7C 2D 95 D2 E8 7F 8F 85 DF AC 77
                       18 21 7D 58 CF E5 31 4D 54 C1 25 82 19 06 82 04
                       31 FD 49 B1 78 30 8A 4E 03 F6 39 38 06 94 6B 09
                       FD F0 AF A5 C8 96 F9 B3 D3 02 03 01 00 01
  Number of extensions: 4

Enter 1 to display extensions, 0 to return to menu:  1 <enter>
```

Figure 40. Certificate Information

Notes:

- For a z/OS PKCS #11 certificate, the Record ID and Issuer Record ID is N/A.

- For a certificate utilizing the RSASSA-PSS signature algorithm, two extra fields are displayed: Digest Algorithm and Mask Algorithm.

From the **Certificate Information** screen, you can also enter **1** to display certificate extensions:

```
        Certificate Extensions List

    1 - subjectKeyIdentifier
    2 - authorityKeyIdentifier
    3 - keyUsage (critical)
    4 - basicConstraints (critical)

Enter extension number (press ENTER to return to previous menu): 3 <enter>
```

Figure 41. Certificate extensions list

Enter 3 on the Certificate Extensions List to show key usage information:

```
Certificate signature
CRL signature

Press ENTER to continue.
```

Figure 42. Key usage information

To display key information, from the **Key and Certificate Menu** or **Token Key and Certificate Menu**, choose **2**, **Show Key Information**. This accesses the **Key Information** menu (see Figure 43 on page 579) or the **Token key information** menu (see Figure 44 on page 579 or Figure 45 on page 580 :

```
                        Key Information

                 Label: Server Cert
             Record ID: 22
      Issuer Record ID: 22
           Default key: No
 Private key algorithm: rsaEncryption
      Private key size: 2048
          Subject name: My Server Cert
                        ID
                        IBM
                        Endicott
                        NY
                        US

Press ENTER to continue.
```

Figure 43. Key information menu

```
                        Token key information

                 Label: Sample RSA Certificate 1
             Record ID: N/A
      Issuer Record ID: N/A
           Default key: Yes
 Private key algorithm: rsaEncryption
      Private key size: 2048
      Private key type: Secure
          Subject name: Certificate with secure private key
                        ID
                        IBM
                        Endicott
                        NY
                        US

Press ENTER to continue.
```

Figure 44. Token key information menu of a certificate with a secure private key

```
                         Token key information

                  Label: Sample RSA Certificate 2
              Record ID: N/A
       Issuer Record ID: N/A
            Default key: Yes
  Private key algorithm: rsaEncryption
       Private key size: 2048
       Private key type: Clear
           Subject name: Certificate with clear private key
                         ID
                         IBM
                         Endicott
                         NY
                         US

Press ENTER to continue.
```

Figure 45. Token key information menu of a certificate with a clear private key

Note: For a z/OS PKCS #11 certificate, the Record ID and Issuer Record ID is N/A.

Marking a certificate (and private key) as the default certificate

Once a certificate has been added to the key database or z/OS PKCS #11 token through either a certificate request or as a self-signed certificate, it can be marked as the default certificate. Marking a certificate as the default certificate allows it to be used by the programs that are calling the System SSL APIs without having to explicitly supply the certificate's label.

To mark a certificate as the default certificate for the key database, from the **Key Management Menu** or **Token Management Menu** (see Key Management Menu), choose **1**, (Manage keys and certificates), and on the **Key and Certificate List** (see Key and Certificate List, choose the label number you want to work with. The **Key and Certificate Menu** or **Token Key and Certificate Menu** displays:

```
       Key and Certificate Menu

       Label: My Server Certificate

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate and key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 3 <enter>

Default key set.

Press ENTER to continue.
```

Figure 46. Marking a certificate (and private key) as the default certificate-Key and Certificate Menu

```
       Token Key and Certificate Menu

       Label:  My Server Certificate

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database/token
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate and key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to the previous menu): 3 <enter>

Default key set.

Press ENTER to continue.
```

Figure 47. Marking a certificate (and private key) as the default certificate-Token Key and Certificate Menu

Choose **3** to set the certificate and private key as the default certificate for the key database or z/OS PKCS #11 token.

Copying a certificate (and private key) to a different key database or z/OS PKCS #11 token

Once your certificates are created, it might be necessary for you to transfer a certificate to another key database or z/OS PKCS #11 token on your system or a remote system. This transfer maybe necessary for these reasons:

- The remote system requires the signing certificate to be in its key database or z/OS PKCS #11 token for validation purposes. The certificate does not need to contain the private key information. These certificates are normally certificate authority (CA) certificates but might also be a self-signed certificate.
- The server or client certificate is being used by another application in a separate key database file or z/OS PKCS #11 token.

Note: The source key database file or z/OS PKCS #11 token, and the target key database file or z/OS PKCS #11 token must exist before the certificate can be copied. If the target is a FIPS database, then only a FIPS database can be the source.

Copying a certificate without its private key

To copy a certificate to a different platform or to a different system without its private key (certificate validation), from the **Key Management Menu** or the **Token Management Menu**, select **1 - Manage keys and certificates** to display the **Key and Certificate List** or the **Token Key and Certificate List** respectively. Find the label of the certificate to be copied and enter the number associated with the label. In the **Key and Certificate Menu** or the **Token Key and Certificate Menu**, enter option **6** to export the certificate to a file. The **Export File Format** menu appears:

```
       Export File Format

   1 - Binary ASN.1 DER
   2 - Base64 ASN.1 DER
   3 - Binary PKCS #7
   4 - Base64 PKCS #7

Select export format (press ENTER to return to menu): 1 <enter>
Enter export file name (press ENTER to return to menu): expfile.der <enter>

Certificate exported.

Press ENTER to continue.
```

Figure 48. Copying a Certificate Without its Private Key

You are then prompted for what file format you would like for the exported certificate information.

The file format is determined by the support on the receiving system. When the receiving system implementation is z/OS System SSL V1R2 or earlier, the selected format **must** be one of the ASN.1 DER formats.

After selecting the export format, you will be asked for a file name. You can now transfer this file to the system and import the certificate. If copying to a remote system, this file can now be transferred (in binary if option 1 or 3 has been selected or in ASCII (TEXT) if option 2 or 4 has been selected) to the remote system. For information about receiving the certificate into the key database file or z/OS PKCS #11 token, see "Importing a certificate from a file as a trusted CA certificate" on page 594). Upon successfully receiving the certificate, the certificate can now be used to validate the SSL's partner certificate. This means that a client with the imported certificate can now validate the servers certificate, while a server with the imported certificate can validate the clients certificate when client authentication is requested.

You must also determine if the certificate should be marked as the default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label. For more information about setting the default certificate, see "Marking a certificate (and private key) as the default certificate" on page 580.

Copying a certificate with its private key

To copy a certificate to a different key database format or to a different system with its private key, the certificate must be exported to a PKCS #12 formatted file. PKCS #12 files are password-protected to allow encryption of the private key information. From the **Key Management Menu** or **Token Management Menu**, select **1 - Manage keys and certificates** to display a list of certificates with private keys. Find the label of the certificate to be copied and enter the number associated with the label. In the **Key and Certificate Menu** or **Token Key and Certificate Menu**, enter option **7** to export the certificate and private key to a file.

The **Export File Format** menu appears:

```
       Export File Format

     1 - Binary PKCS #12 Version 1
     2 - Base64 PKCS #12 Version 1
     3 - Binary PKCS #12 Version 3
     4 - Base64 PKCS #12 Version 3

Select export format (press ENTER to return to menu): 3 <enter>
Enter export file name (press ENTER to return to menu): expfile.p12 <enter>
Enter export file password (press ENTER to return to menu): <enter password>
Re-enter export file password: <enter password>
Enter 1 for strong encryption, 0 for export encryption: 1 <enter>

Certificate and key exported.

Press ENTER to continue.
```

Figure 49. Copying a Certificate and Private key to a Different Key Database-Export File Format

```
       Export File Format

     1 - Binary PKCS #12 Version 3
     2 - Base64 PKCS #12 Version 3

Select export format (press ENTER to return to menu): 1 <enter>
Enter export file name (press ENTER to return to menu): expfile.p12 <enter>
Enter export file password (press ENTER to return to menu): <enter password>
Re-enter export file password: <enter password>
Enter 1 for strong encryption, 0 for export encryption: 1 <enter>

Certificate and key exported.

Press ENTER to continue.
```

Figure 50. Copying a Certificate and Private key to a Different Key Database-Export File Format

The second display applies to z/OS PKCS #11 tokens.

You are then prompted for what file format you would like for the exported certificate information.

The file format is determined by the support on the receiving system. In most cases the format to be used is Binary PKCS #12 Version 3. When the receiving system implementation is z/OS System SSL V1R2 or earlier, the selected format **must** be Binary PKCS #12 Version 1. z/OS PKCS #11 tokens only support Version 3 PKCS #12 export. Export from a FIPS database must be PKCS #12 Version 3 using strong encryption.

After selecting the export format, you are asked for a file name and password. You then receive a message indicating that the certificate was exported. You can now transfer this file to the system and import the certificate into the key database file or z/OS PKCS #11 token. If copying to a remote system, this file can now be transferred (in binary) to the remote system. For information about receiving the certificate into the key database file, see "Importing a certificate from a file with its private key" on page 597). Upon successfully receiving the certificate, the certificate can now be used to identify the program. For example, the certificate can be used as the SSL server program's certificate or it can be used as the SSL client program's certificate.

Copying a certificate and its private key from a key database on the same system

To copy a certificate and its private key from one key database to another key database or z/OS PKCS #11 token on the same system, you need to know the target key database file name and password, or the z/OS PKCS #11 token name. If the target database is a FIPS database, then the source database must also be a FIPS database. Copying into a FIPS database from a non-FIPS database or z/OS PKCS #11 token is not supported. If the target database is a non-FIPS database or z/OS PKCS #11 token, then the source may be a non-FIPS database, a FIPS database, or a z/OS PKCS #11 token. From the **Key Management Menu**, select **1 - Manage keys and certificates** to display the **Key and Certificate Menu**. Find the label of the certificate to be copied and enter the number associated with the label. From the **Key and Certificate Menu**, enter **5** to copy a certificate and key to another database or z/OS PKCS #11 token.

```
       Key and Certificate Menu

       Label: newimp


   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database/token
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate and key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press Enter to return to previous menu):  5 <enter>
Enter 1 to specify token name or
      2 to specify database name
      (press ENTER to return to menu): 2 <enter>
Enter key database name (press Enter to return to previous menu): target.kdb <enter>
Enter database password (press Enter to return to previous menu): <enter password>

Record copied.

Press ENTER to continue.
```

Figure 51. Copying a Certificate with its Private Key to a Key Database on the Same System

You will then be prompted for the target key database name, and the target key database password. Once the certificate is copied to the other key database file, you will receive a message indicating that the certificate has been successfully copied.

Note: When a certificate with a key marked as default is copied from a key database into another token or database, it is not marked as the default key in that token or database.

Copying a certificate and its private key from a z/OS PKCS #11 token on the same system

To copy a certificate and its private key from a z/OS PKCS #11 token to another z/OS PKCS #11 token or key database file on the same system, from the **Token Management Menu**, select **1 - Manage Keys and Certificates** to display the Token Key and Certificate List. Find the label of the certificate to be copied and enter the number associated with the label. From the **Token Key and Certificate Menu** enter **5** to copy a certificate and key to another token or a key database file. If the target is a key database on the same system, you need to know the targets file name and password.

```
        Token Key and Certificate Menu

        Label: newimp


    1 - Show certificate information
    2 - Show key information
    3 - Set key as default
    4 - Set certificate trust status
    5 - Copy certificate and key to another database/token
    6 - Export certificate to a file
    7 - Export certificate and key to a file
    8 - Delete certificate and key
    9 - Change label
   10 - Create a signed certificate and key
   11 - Create a certificate renewal request

    0 - Exit program

Enter option number (press ENTER to return to previous menu): 5 <enter>

Enter 1 to specify token name or
      2 to specify database name
      (press ENTER to return to menu): 1 <enter>
Enter token name (press ENTER to return to menu): TOKENDEF <enter>

Record copied.

Press ENTER to continue.
```

Figure 52. Copying a Certificate with its Private Key to a z/OS PKCS #11 Token on the Same System

You will then be prompted to choose either a z/OS PKCS #11 token or a key database as the target of the copy. Figure 52 on page 585 shows the prompts if a z/OS PKCS #11 token is chosen as the target. Once the certificate is copied, you will receive a message indicating that the certificate has been successfully copied.

Note: When a certificate with a key marked as default is copied from a key database into another token or database, it is not marked as the default key in that token or database.

Removing a certificate (and private key)

You may want to remove a certificate if:

- The certificate has expired and is no longer useful.
- The certificate has been exported to a different key database or z/OS PKCS #11 token and is no longer needed in the current database or token.

`Caution:` Once you delete a certificate/private key pair, it cannot be recovered unless it has previously been stored somewhere else (another key database file, z/OS PKCS #11 token, a PKCS #12 file for certificate/private key pairs, or a DER-encoded or Base64-encoded file for certificates). Be sure that you no longer require the certificate (and private key if one is associated with the certificate) before you remove it.

From the **Key Management Menu** or **Token Management Menu**, select **1 - Manage keys and certificates** to display the **Key and Certificate List** or **Token Key and Certificate List** respectively. Find the label of the certificate and key to be deleted and enter the number associated with the label. From the **Key and Certificate Menu** or **Token Key and Certificate Menu** (see Figure 53 on page 586), choose **8** to delete the certificate and key.

```
       Key and Certificate Menu

       Label: newimp


   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate and key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 8 <enter>

Enter 1 to confirm delete, 0 to cancel delete: 1 <enter>

Record deleted.

Press ENTER to continue.
```

Figure 53. Delete Certificate and Key-Key and Certificate Menu

```
       Token Key and Certificate Menu

       Label: newimp


   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database/token
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate and key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 8 <enter>

Enter 1 to confirm delete, 0 to cancel delete: 1 <enter>

Record deleted.

Press ENTER to continue.
```

Figure 54. Delete Certificate and Key-Token Key and Certificate Menu

Enter **1** to confirm the deletion of the certificate and key. A message appears, confirming that the record has been deleted. Once the certificate has been deleted, it can no longer be used for identification or verification purposes by the System SSL APIs during SSL handshake processing.

Changing a certificate label

Find the certificate label to be changed and enter the number associated with the label. In the **Key and Certificate Menu** or **Token Key and Certificate Menu** (see Figure 55 on page 587), choose **9** to change the label:

```
       Key and Certificate Menu

       Label: cacert

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 9 <enter>
Enter label (press ENTER to return to menu): cacert2 <enter>

Label changed.

Press ENTER to continue.
```

Figure 55. Changing a Certificate Label-Key and Certificate Menu

```
       Token and Certificate Menu

       Label: cacert

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database/token
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 9 <enter>
Enter label (press ENTER to return to menu): cacert2 <enter>

Label changed.

Press ENTER to continue.
```

Figure 56. Changing a Certificate Label-Token and Certificate Menu

Enter the new label name and press Enter. A message confirms that the label name has been changed.

Creating a signed certificate and key

Creating a signed certificate and key allows for a fast path method for creating a signed certificate in the same key database file or z/OS PKCS #11 token as the displayed signing Certificate Authority certificate. From the **Key Management Menu** or **Token Management Menu**, select **1 - Manage keys and certificates** to display the **Key and Certificate List** or **Token Key and Certificate List**. Find the label of the signing Certificate Authority certificate and enter the number associated with the label. From the **Key and Certificate Menu** or **Token Key and Certificate Menu** (see Figure 57 on page 588), choose option **10** to create a signed certificate and key.

Note: This requires the displayed certificate to have signing capability.

```
       Key and Certificate Menu

       Label: cacert

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate and key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 10 <enter>
```

Figure 57. Select 10 to Create a Signed Certificate and Key-Key and Certificate Menu

```
       Token Key and Certificate Menu

       Label: cacert

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database/token
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate and key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 10 <enter>
```

Figure 58. Select 10 to Create a Signed Certificate and Key-Token Key and Certificate Menu

The **Certificate Usage** menu appears, followed by menus to select the certificate key algorithm and key size (or ECC key type and EC named curve if ECC is selected as the certificate key algorithm. See "Creating a signed ECC certificate and key" on page 590.) Once these details are determined, you will be prompted to enter:

- a label to uniquely identify the key and certificate within the key database or z/OS PKCS #11 token
- the individual fields within the subject name
- certificate expiration. The valid range for a self-signed certificate is 1 to 9999 days. The default is 365 days. The expiration date cannot exceed February 6, 2106, at 23:59:59 UTC.

```
       Certificate Usage

   1 – CA certificate
   2 – User or server certificate

Select certificate usage (press ENTER to return to menu): 2 <enter>

       Certificate Key Algorithm

   1 - Certificate with an RSA key
   2 - Certificate with a DSA key
   3 - Certificate with an ECC key
   4 - Certificate with a Diffie-Hellman key
   5 - Certificate with an RSA key and RSASSA-PSS signature

Select certificate key algorithm (press ENTER to return to menu): 1 <enter>

       RSA Key Size

   1 – 1024-bit key
   2 – 2048-bit key
   3 – 4096-bit key

Select RSA key size (press ENTER to return to menu): 2 <enter>
Enter label (press ENTER to return to menu): signedcert <enter>
Enter subject name for certificate
  Common name (required): My signed Certificate <enter>
  Organizational unit (optional): ID <enter>
  Organization (required): IBM <enter>
  City/Locality (optional): Endicott <enter>
  State/Province (optional): NY <enter>
  Country/Region (2 characters - required): US <enter>
Enter number of days certificate will be valid (default 365): 300 <enter>

Enter 1 to specify subject alternate names or 0 to continue: 1

Please wait .....
```

Figure 59. Enter Certificate Details

Enter option **0** to continue or option **1** to specify the subject alternate names. If option **1** is selected, the **Subject Alternate Name Type** menu appears.

```
 Subject Alternate Name Type

   1 - Directory name (DN)
   2 - Domain name (DNS)
   3 - E-mail address (SMTP)
   4 - Network address (IP)
   5 - Uniform resource identifier (URI)

Select subject alternate name type (press ENTER if name is complete): 1 <enter>
Enter subject name for certificate
  Common name (required): Test server <enter>
  Organizational unit (optional): ID <enter>
  Organization (required): IBM <enter>
  City/Locality (optional): Endicott <enter>
  State/Province (optional): NY <enter>
  Country/Region (2 characters - required): US <enter>

       Subject Alternate Name Type

   1 - Directory name (DN)
   2 - Domain name (DNS)
   3 - E-mail address (SMTP)
   4 - Network address (IP)
   5 - Uniform resource identifier (URI)

Select subject alternate name type (press ENTER if name is complete): <enter>

Please wait .....
```

Figure 60. Subject Alternate Name Type

When specifying subject alternate names, you are prompted for the type of the alternate name. After the alternate name type is determined, you will be prompted to enter:

- the individual fields within the subject name.

After the individual fields are completed, enter option **0** to continue or option **1** to specify another subject alternate name (repeat the process).

Creating a signed ECC certificate and key

If ECC is selected as the certificate key algorithm in the **Certificate Key Algorithm menu**, you are prompted to choose the ECC key type (for user or server certificates only) to be set in the new certificate and the EC named curve to be used when generating the ECC key. Supported EC named curves are outlined in "Elliptic Curve Cryptography support" on page 12.

The following example creates an end-entity certificate with an ECDSA key using a 256-bit NIST suggested named curve.

```
       Certificate Usage

   1 – CA certificate
   2 – User or server certificate

Select certificate usage (press ENTER to return to menu): 2 <enter>

       Certificate Key Algorithm

    1 - Certificate with an RSA key
    2 - Certificate with a DSA key
    3 - Certificate with an ECC key
    4 - Certificate with a Diffie-Hellman key

Select certificate key algorithm (press ENTER to return to menu): 3 <enter>

       ECC Key Type

   1 – General ECC key
   2 – ECDSA Key
   3 - ECDH key

Select ECC key type (press ENTER to return to menu): 2 <enter>
```

Figure 61. Selecting the ECC Key Type

The selected key type determines the setting of the keyUsage extension in the new certificate. A general ECC key allows Digital Signature, Non-repudiation and Key Agreement. An ECDSA key allows Digital Signature and Non-repudiation. An ECDH key allows Key Agreement only.

If option **1** is selected in the **Certificate Usage menu**, requesting a CA certificate, the **ECC Key Type menu** does not appear. The keyUsage extension of the new certificate is set to allow the certificate to be used to sign certificates and certificate revocation lists.

Once the key type has been selected, you are prompted to select the ECC curve type. For a FIPS database, Brainpool standard curves are not supported.

```
       ECC Curve Type

   1 - NIST recommended curve
   2 - Brainpool standard curve

Select ECC curve type (press ENTER to return to menu): 1 <enter>

       NIST Recommended Curve Type

   1 - secp192r1
   2 - secp224r1
   3 - secp256r1
   4 - secp384r1
   5 - secp521r1

Select NIST recommended curve (press ENTER to return to menu): 3 <enter>

Enter label (press ENTER to return to menu): signedECCcert <enter>
Enter subject name for certificate
  Common name (required): My signed ECC Certificate <enter>
  Organizational unit (optional): ID <enter>
  Organization (required): IBM <enter>
  City/Locality (optional): Endicott <enter>
  State/Province (optional): NY <enter>
  Country/Region (2 characters - required): US <enter>
Enter number of days certificate will be valid (default 365): 300 <enter>

Enter 1 to specify subject alternate names or 0 to continue: 0 <enter>

Please wait .....

Certificate created.

Press ENTER to continue.
```

Figure 62. Selecting the ECC Curve Type

For a FIPS database, some curves may not be recommended for use and may not appear in the **ECC Curve Type menu**. After selecting the curve type you are prompted to enter the certificate label, subject name, expiration and (optionally) subject alternate names. See "Creating a signed certificate and key" on page 587 for more information.

Creating a certificate to be used with a fixed Diffie-Hellman key exchange

Create a server certificate to be used during an SSL handshake using a fixed Diffie-Hellman key exchange. Fixed Diffie-Hellman requires the certificates being used by both sides of the exchange to be based off the same generation parameters. In order for each side to use the same generation parameters, a key parameter file must be created to be used as input to the certificate being signed.

To create a key parameter file, from the **Database Menu**, enter **6**. You are asked to select the key type and key size. Only 1024-bit DSA keys, 2048-bit DSA keys, or 2048-bit fixed Diffie-Hellman keys are valid for use in a FIPS database. When the key type is determined, you are prompted to enter a key parameter file name. The file name is interpreted relative to the current directory when **gskkyman** is invoked. You may also specify a fully qualified file name.

```
       Database Menu

   1  - Create new database
   1b - Create new empty database
   2  - Open database
   3  - Change database password
   4  - Change database record length
   5  - Delete database
   6  - Create key parameter file
   7  - Display certificate file (Binary or Base64 ASN.1 DER)

  11  - Create new token
  12  - Delete token
  13  - Manage token
  14  - Manage token from list of tokens

   0  - Exit program

Enter option number: 6 <enter>
```

```
       Key Type

   1 - DSA key
   2 - Diffie-Hellman key

Select key type (press ENTER to return to menu: 2 <enter>

       Diffie-Hellman Key Size

   1 - 1024-bit key
   2 - 2048-bit key

Select Diffie-Hellman key size (press ENTER to return to menu): 1 <enter>
Enter key parameter file name (press ENTER to return to menu): dh_key_1024.keyfile <enter>

Please wait ......

Key parameter file created.

Press ENTER to continue
```

Figure 63. Creating a key parameter file to be used with Diffie-Hellman

When the key parameter file is created, the next step is to create the signed certificate by using an existing certificate in the key database file or z/OS PKCS #11 token to sign the server certificate. From the **Key Management Menu** or **Token Management Menu**, select **1 - Manage keys and certificates** to display the **Key and Certificate List**. From the **Key and Certificate List**, select a CA certificate by entering the appropriate selection number, and then choose option **10** to create a signed certificate and key. This requires the displayed certificate to contain an RSA or a DSA key and have signing capability.

Select User or server certificate by choosing option **2** in the **Certificate Usage menu**, followed by option **4 - Certificate with a Diffie-Hellman key** in the **Certificate Key Algorithm menu**, and then select the Diffie-Hellman key size. The key size must match the key size of the key parameters created previously.

When the certificate type is determined, you are prompted to enter:

- Key parameter file created previously.
- A label to uniquely identify the key and certificate within the key database.
- The individual fields within the subject name.
- Certificate expiration (Valid expiration range is 1 to 9999 days. Default value is 365 days). The expiration date cannot exceed February 6, 2106, at 23:59:59 UTC.
- The subject alternate names (optional).

```
       Certificate Usage

   1 – CA certificate
   2 – User or server certificate

Select certificate usage (press ENTER to return to menu): 2 <enter>

       Certificate Key Algorithm

   1 - Certificate with an RSA key
   2 - Certificate with a DSA key
   3 - Certificate with an ECC key
   4 - Certificate with a Diffie-Hellman key

Select certificate key algorithm (press ENTER to return to menu): 4 <enter>

       Diffie-Hellman Key Size

   1 – 1024-bit key
   2 – 2048-bit key

Select key size (press ENTER to return to menu): 1 <enter>
Enter key parameter file name (press ENTER to return to menu):   dh_key_1024.keyfile <enter>
Enter label (press ENTER to return to menu):   DSA_cert_with_DH_1024_key <enter>
Enter subject name for certificate:
       Common name (required):   DSA cert with DH 1024 key <enter>
       Organizational unit (optional):   Test <enter>
       Organization (required):   Test <enter>
       City/Locality (optional):   Poughkeepsie <enter>
       State/Province (optional):   NY <enter>
       Country/Region (2 characters - required):   US <enter>
Enter number of days certificate will be valid (default 365):   5000 <enter>

Enter 1 to specify subject alternate names or 0 to continue:   0 <enter>

Please wait .....

Certificate created.

Press ENTER to continue.
```

Figure 64. Creating a certificate to be used with Diffie_Hellman

When the certificate is created, the next step is to determine if the certificate must be transferred to another database. If the certificate does not need to reside elsewhere, you must determine whether the certificate should be marked as the database's default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label. For more information about setting the default certificate, see "Marking a certificate (and private key) as the default certificate" on page 580. If the certificate must be transferred, see "Copying a certificate (and private key) to a different key database or z/OS PKCS #11 token" on page 581 for more information.

Creating a certificate renewal request

Certificate renewal requests allow for existing signed certificates that have expired or are nearing their expiration dates to be renewed without having to create a brand new certificate request. The renewed certificate continues to contain the same subject name and public/private key pair. From the **Key Management Menu** or **Token Management Menu** , select **1 - Manage keys and certificates** to display the **Key and Certificate List** or **Token Key and Certificate List**. Find the label of the certificate to be renewed and enter the number associated with the label. From the **Key and Certificate Menu** or **Token Key and Certificate Menu** (see Figure 65 on page 594) choose option **11** to create a certificate renewal request.

```
       Key and Certificate Menu

       Label: cacert

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 11 <enter>
Enter request filename (press ENTER to return to menu): renewal.arm <enter>

Certificate request created.

Press ENTER to continue.
```

Figure 65. Select 11 to Create a Certificate Renewal Request-Key and Certificate Menu

```
       Token Key and Certificate Menu

       Label: cacert

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database/token
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  10 - Create a signed certificate key
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 11 <enter>
Enter request filename (press ENTER to return to menu): renewal.arm <enter>

Certificate request created.

Press ENTER to continue.
```

Figure 66. Select 11 to Create a Certificate Renewal Request-Token Key and Certificate Menu

Enter the request file name (press ENTER to return to menu). The certificate request is created. Press enter to continue. After creating the certificate renewal request, perform the following steps:

1. If you want a certificate authority (CA) to sign the certificate, send the certificate request to the CA. See "Sending the certificate request" on page 574. If you are acting as your own CA, use the **gskkyman** command line interface to sign the certificate. See "Using gskkyman to be your own certificate authority (CA)" on page 598.
2. Receive the renewed certificate into your key database. See "Receiving the signed certificate or renewal certificate " on page 575.

Importing a certificate from a file as a trusted CA certificate

If you are using a certificate authority for generating your certificates that are not one of the default certificate authorities for which certificates are already stored in the key database, or if you are using a z/OS PKCS #11 token for which no default certificates exist, then you must import the certificate authority's certificate into your key database file or z/OS PKCS #11 token before you use the System

SSL APIs. If you are using **client authentication**, then the CA certificate must be imported into the key database or z/OS PKCS #11 token of the server program. The client program's key database file or z/OS PKCS #11 token must have the CA certificate that is imported regardless of whether the SSL connection uses **client authentication**.

If you are using a self-signed certificate as the SSL server program's certificate and your SSL client program is also using the System SSL APIs, then you must import the server's self-signed certificate without its private key into the client program's key database file or z/OS PKCS #11 token.

If you are using a self-signed certificate as the SSL client program's certificate and your SSL server program is also using the System SSL APIs with client authentication requested, then you must import the client's self-signed certificate without its private key into the server program's key database file or z/OS PKCS #11 token.

If the CA certificate that is being imported was signed by another CA certificate, the complete chain must be present in the key database file or z/OS PKCS #11 token before the import. If the CA certificates chain consists of more than one certificate and the certificates exist in individual files, you must import the certificates starting with the root CA certificate.

If you are using a key database file, a number of well-known certificate authority (CA) certificates are stored in the key database when the key database is created. To get a certificate list, select **2 - Manage certificates** from the **Key Management Menu**. The following figures contain lists of CAs for which certificates are stored on key database creation:

```
       Certificate List

       Database: /home/sufwl1/ssl_cmd/mykey.kdb

   1 - VeriSign Class 1 Public Primary CA - G2
   2 - VeriSign Class 2 Public Primary CA - G2
   3 - VeriSign Class 3 Public Primary CA - G2
   4 - VeriSign Class 4 Public Primary CA - G2
   5 - VeriSign Class 1 Public Primary CA - G3
   6 - VeriSign Class 2 Public Primary CA - G3
   7 - VeriSign Class 3 Public Primary CA - G3
   8 - VeriSign Class 4 Public Primary CA - G3
   9 - VeriSign Class 5 Public Primary CA - G5

   0 - Return to selection menu

Enter label number (ENTER to return to selection menu, p for previous list):
```

Figure 67. Certificate List

To import a certificate without a private key into your key database file or z/OS PKCS #11 token, first get the certificate in a file with the file in either Base64-encoded, Binary encoded or PKCS #7 format. From the **Key Management Menu** or the **Token Management Menu** enter **7** to import a certificate:

```
       Key Management Menu

       Database: /home/sufwl1/ssl_cmd/mykey.kdb
       Expiration Date: 2025/12/02  10:11:12
       Type: non-FIPS

   1 - Manage keys and certificates
   2 - Manage certificates
   3 - Manage certificate requests
   4 - Create new certificate request
   5 - Receive requested certificate or a renewal certificate
   6 - Create a self-signed certificate
   7 - Import a certificate
   8 - Import a certificate and a private key
   9 - Show the default key
  10 - Store database password
  11 - Show database record length

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 7 <enter>
Enter import file name (press ENTER to return to menu): cert.arm <enter>
Enter label (press ENTER to return to menu): cacert2 <enter>

Certificate imported.

Press ENTER to continue.
```

Figure 68. Importing a Certificate from a File-Key Management Menu

```
       Token Management Menu

       Token: TOKENABC

     Manufacturer: z/OS PKCS11 API
     Model: HCR77D2
     Flags: x00000509 (INITIALIZED, PROT AUTH PATH, USER PIN INIT, RNG)

   1 - Manage keys and certificates
   2 - Manage certificates
   3 - Manage certificate requests
   4 - Create new certificate request
   5 - Receive requested certificate or a renewal certificate
   6 - Create a self-signed certificate
   7 - Import a certificate
   8 - Import a certificate and a private key
   9 - Show the default key
  10 - Delete Token

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 7 <enter>
Enter import file name (press ENTER to return to menu): cert.arm <enter>
Enter label (press ENTER to return to menu): cacert2 <enter>

Certificate imported.

Press ENTER to continue.
```

Figure 69. Importing a Certificate from a File-Token Management Menu

You are prompted to enter the certificate file name and your choice of a unique label that are assigned to the certificate.

When the certificate is imported, you receive a message that indicates the import was successful. The certificate is treated as "trusted" so that it can be used in verifying incoming certificates. For a program that is acting as an SSL server, this certificate is used during the verification of a client's certificate. For a program that is acting as an SSL client, this certificate is used to verify the server's certificate that is sent to the client during SSL handshake processing.

Importing a certificate from a file with its private key

To store a certificate into a different key database format or to a different system with its private key, the certificate must be exported from the source system into a PKCS #12 format file (See "Copying a certificate with its private key" on page 582 for more information). PKCS #12 files are password-protected to allow encryption of the private key information. If the CA certificate that is being imported was signed by another CA certificate, the complete chain must be present in the key database file or z/OS PKCS #11 token before the import. From the **Key Management Menu** or **Token Management Menu** , enter **8** to import a certificate and a private key:

```
       Key Management Menu

       Database: /home/sufwl1/ssl_cmd/anne.kdb
       Expiration Date: 2025/12/02  10:11:12
       Type: non-FIPS

   1 - Manage keys and certificates
   2 - Manage certificates
   3 - Manage certificate requests
   4 - Create new certificate request
   5 - Receive requested certificate or a renewal certificate
   6 - Create a self-signed certificate
   7 - Import a certificate
   8 - Import a certificate and a private key
   9 - Show the default key
  10 - Store database password
  11 - Show database record length

   0 - Exit program



Enter option number (press ENTER to return to previous menu): 8 <enter>
Enter import file name (press ENTER to return to menu): cert.p12 <enter>
Enter import file password (press ENTER to return to menu): <enter password>
Enter label (press ENTER to return to menu): newcert <enter>

Certificate and key imported.

Press ENTER to continue.
```

Figure 70. Importing a Certificate and Private Key from a File-Key Management Menu

```
        Token Management Menu

        Token: TOKENABC

      Manufacturer: z/OS PKCS11 API
      Model: HCR77D2
      Flags: x00000509 (INITIALIZED, PROT AUTH PATH, USER PIN INIT, RNG)

    1 - Manage keys and certificates
    2 - Manage certificates
    3 - Manage certificate requests
    4 - Create new certificate request
    5 - Receive requested certificate or a renewal certificate
    6 - Create a self-signed certificate
    7 - Import a certificate
    8 - Import a certificate and a private key
    9 - Show the default key
   10 - Delete Token

    0 - Exit program

Enter option number (press ENTER to return to previous menu): 8 <enter>
Enter import file name (press ENTER to return to menu): cert.p12 <enter>
Enter import file password (press ENTER to return to menu): <enter password>
Enter label (press ENTER to return to menu): newcert <enter>

Certificate and key imported.

Press ENTER to continue.
```

Figure 71. Importing a Certificate and Private Key from a File-Token Management Menu

You are prompted to enter the certificate file name, password, and your choice of a unique label to be assigned to the certificate.

Once the certificate is imported, you receive a message indicating that import was successful. The next step is to determine whether the certificate should be marked as the database's or tokens default certificate. Setting the certificate as the default certificate allows the certificate to be used by the SSL APIs without having to specify its label. For more information about setting the default certificate, see "Marking a certificate (and private key) as the default certificate" on page 580).

A certificate and key can be imported into a FIPS key database providing it is a PKCS #12 Version 3 with strong encryption format. When adding certificates from the import file to a FIPS key database file only certificates signed with FIPS signature algorithms using FIPS-approved key sizes may be imported. When processing a chain of certificates, processing of the chain terminates if a non-FIPS certificate is encountered. Certificates that are processed before the failing certificate is added to the key database file. It is the responsibility of the importer to ensure that the file came from a source meeting FIPS 140-2 criteria to maintain adherence to the FIPS criteria.

Using gskkyman to be your own certificate authority (CA)

The gskkyman utility provides the capability for you to act as your own Certificate Authority (CA). If your own CA, you are authorized to sign certificate requests for yourself or others. This is convenient if you need certificates within your private web network and not for outside Internet commerce.

To be your own CA in a web network, you must create a CA database and self-signed CA certificate using gskkyman. A server or client that wants you to sign a certificate must supply you with their certificate request. After signing the certificate, the server or client must receive the CA certificate and the newly signed certificate. The CA-signed certificate must then be received into either the client or server key database.

This table describes the steps that are needed to become your own CA to allow secure communication between a client and a server. This example reflects the steps that are followed when the CA is on a different system or is a different user than the issuer of the certificate request.

Certificate Authority (System A)	Server or Client (System B)
Step 1 - Create a key database	
Create a key database using the **gskkyman** utility: • From the **Database Menu**, select option **1 - Create new database** See "Creating, opening, and deleting a key database file" on page 559 for details.	Create a key database using the **gskkyman** utility: • From the **Database Menu**, select option **1 - Create new database** See "Creating, opening, and deleting a key database file" on page 559 for details.
Step 2 - Create a Root Certificate Authority certificate	
Create a Certificate Authority certificate: • From the **Key Management Menu**, select option **6 - Create a self-signed certificate** • From the **Certificate Usage** menu, select option **1 - CA certificate** See "Creating a self-signed server or client certificate" on page 567 for details.	No action required.
Step 3 - Create a certificate request	
No action required.	Create a certificate request: • From the **Key Management Menu**, select option **4 - Create new certificate request** See "Creating a certificate request" on page 570 for details.
Step 4 - Send the certificate request to the CA	
No action required.	Send the certificate request to the CA: See "Sending the certificate request" on page 574.
Step 5 - Sign the certificate request	

Certificate Authority (System A)	**Server or Client (System B)**

Before signing a certificate for a client or server, you must make sure that the requester has a legitimate claim to request the certificate. After verifying the claim, you can create a signed certificate.
 	
To sign the certificate request, the **gskkyman** utility must be issued using command-line options (see "gskkyman command line mode syntax" on page 601 for a description of the options). The **gskkyman** utility must be issued with these parameters:
 	
`gskkyman -g -x` *`num-of-valid-days`*
`-cr` *`certificate-request-file-name`*
`-ct` *`signed-certificate-file-name`*
`-k` *`CA-key-database-file-name`*
`-l` *`label`*
 	
Example: This command allows you to sign a request certificate and allow the certificate to be valid for 360 days.
 	
`gskkyman -g -x` *`360`* `-cr` *`server_request.arm`*
`-ct` *`server_signed_cert.arm`* `-k` *`CA.kdb`*
`-l` *`labelname`*
 	
After you entered the command, you are prompted to enter the database password.
 	
Note:
1. The signed certificate is an end user certificate unless the -ca option is specified.
2. The file name that is specified on the -ct option is created for you by the utility, and is the actual signed certificate file.
3. The valid certificate lifetime range is between 1 and 9999 days. The certificate end date is set to the end date for the CA certificate if the requested certificate lifetime exceeds the CA certificate lifetime. The expiration date cannot exceed February 6, 2106, at 23:59:59 UTC.
	No action required.

Step 6 - Send the signed CA certificate and the newly signed certificate to the requester	
Export the signed CA certificate (created in Step 2) to a Base64 file (DER or PKCS #7) See "Copying a certificate without its private key" on page 581. Send (for example, without its private key FTP) the Base64 file and the newly signed certificate (created in Step 4) to the requester.	No action required.
Step 7 - Import the CA certificate	
No action required.	Import the CA certificate. See "Importing a certificate from a file as a trusted CA certificate" on page 594.
Step 8 - Receive the signed certificate	

Certificate Authority (System A)	Server or Client (System B)
No action required.	Receive the signed certificate. See "Receiving the signed certificate or renewal certificate " on page 575. **Note:** Depending upon the SSL application, you might have to either send the CA certificate to the client, or the server application might present the certificate to the client for them during SSL session setup.

Migrating from key database files to z/OS PKCS #11 token

If you need to migrate keys and certificates stored in an existing key database into a z/OS PKCS #11 token, follow these steps

1. Export the certificate/private key to a password protected PKCS #12 file using **gskkyman**. See "Copying a certificate with its private key" on page 582 for details about the steps for exporting certificates/private keys to a PKCS #12 file.
2. Import the certificate/private key from the PKCS #12 file into the z/OS PKCS #11 token using **gskkyman**. See "Importing a certificate from a file with its private key" on page 597 for more information.

Migrating key database files to RACF key rings

If you need to migrate keys and certificates stored in an existing key database into a RACF key ring, follow these steps:

1. Export the certificate/private key to a password protected PKCS #12 file using **gskkyman**. See "Copying a certificate with its private key" on page 582 for details on the steps for exporting certificates/private keys to a PKCS #12 file.
2. Copy the newly created PKCS #12 file to a z/OS data set.
3. Use the RACDCERT command with the ADD operand and the data set name created in step 2 to add the certificate/private key to the RACF database. The certificate should be added as TRUSTED. If the private key is to be stored in the ICSF PKDS, the ICSF keyword also needs to be specified on the RACDCERT command.
4. Use the RACDCERT command with the ADDRING operand to create a new key ring in RACF. Use the RACDCERT command with the CONNECT operand to add the certificate/private key to one or more existing RACF key rings.

gskkyman command line mode syntax

This topic describes the format and options of the **gskkyman** command.

gskkyman

The **gskkyman** utility is used for key database management, z/OS PKCS #11 token management, and to display the certificates within GSKIT CMS V4 key databases and PKCS #12 files.

Format

```
gskkyman
```

```
gskkyman -dc|-dcv [-k filename|-t tokenname|-p12 filename|-der filename] [-l label]
gskkyman -dk [-k filename]
gskkyman -e|-i [-k filename|-t tokenname] [-l label] [-p filename]
gskkyman -g [-x days] [-cr filename] [-ct filename] [-k filename|-t tokenname]
  [-l label] [-kt {ecgen|ecdsa|ecdh}] [-ca] [-ic] [-pss]
gskkyman -h|-?
gskkyman -s [-k filename]
```

Parameters

function
The function to be performed. It must follow the command name. The acceptable values are:

-dc
Display certificate details.

-dcv
Display certificate verbose details.

-dk
Display key database expiration, record length and type.

-e
Export a certificate and its associated private key.

-g
Sign a certificate for a certificate request.

-h
Display the command syntax.

-i
Import a certificate and its associated private key.

-s
Store the database password in the stash file.

-?
Display the command syntax.

option
The parameters necessary to accomplish the function. If the option provides a value, then the value must follow the option:

The acceptable values are:

-ca
A certification authority certificate is generated if **-ca** is specified. An end user certificate is generated if **-ca** is not specified.

-cr
Specifies the name of the certificate request file. You are prompted for the file name if this option is not specified.

-ct
Specifies the name of the output generated signed certificate file. You are prompted for the file name if this option is not specified. You may specify any name. If you specify an existing file name, the file is overwritten.

-der
Specifies the name of the X.509 certificate file. The specified file must contain either a binary ASN.1 DER-encoded certificate or the Base64-encoding of a binary ASN.1 stream. A Base64-encoded certificate must be in the local code page. The fully-qualified certificate file name must be less than 251 characters. This option is mutually exclusive with the **-k** option, the **-t** option, and the **-p12** option. The **-l** option is not supported.

-ic
The certification chain certificates are included in the certificate file if **-ic** is specified. Otherwise, just the signed certificate is included in the certificate file.

-k
: Specifies the name of the key database or GSKIT CMS V4 key database. This option is mutually exclusive with the **-t** option, the **-p12** option, and the **-der** option. You are prompted for the key database file name if neither this option nor the **-t** option, the **-p12** option, or the **-der** option is specified. The length of the fully qualified file name cannot exceed 251 characters. If the file name does not end with an extension of 1-3 characters, the length of the fully qualified file name cannot exceed 247 characters. Finally, the key database name cannot end with .rdb or .sth.

-kt
: Specifies the key type of the certificate to be created. This option is valid when signing an end user certificate or certificate request containing an ECC public key and affects the settings of the keyUsage extension of the certificate created. Valid key type options are ecgen, ecdsa and ecdh. ecgen creates a certificate with digitalSignature, nonRepudiation and keyAgreement set, ecdsa creates a certificate with digitalSignature and nonRepudiation set, and ecdh creates a certificate with keyAgreement set. If the **-kt** option is not specified for an end user ECC certificate or certificate request, the default option is ecgen. For other certificate types the **-kt** option is ignored.

-l
: Specifies the certificate label. This option is not supported with the **-der** option. The label must be enclosed in double quotation marks if it contains one or more spaces. If the certificate is being used to sign a certificate request (sign function), the certificate must be a CA. The label for the default key is used if this option is not specified (export or sign function) or you are prompted for the label (import function). If more than one certificate with the specified label exists (can occur for tokens), the user is prompted to either cancel or choose the required certificate from a list that summarizes significant fields in the certificate.

-p
: Specifies the name of the PKCS #12 file to be used to import or export certificates. You are prompted for the file name if this option is not specified.

-p12
: Specifies the name of the PKCS #12 file containing the certificates to be displayed. This option is mutually exclusive with the **-k** option, the **-t** option, and the **-der** option. The length of the fully qualified file name cannot exceed 251 characters. If the file name does not end with an extension of 1-3 characters, the length of the fully qualified file name cannot exceed 247 characters. Lastly, the PKCS #12 file cannot end with .kdb, .rdb or .sth.

-pss
: Specifies the signature algorithm to be used when signing the certificate must be RSASSA-PSS. The signature requires that the signing certificate have a RSA key. The key size must be 2048 through 4096 bits inclusive and the certificate request being signed must be either RSA (2048 - 4096 bits) or ECC (256 - 521 bits).

-t
: Specifies the name of the token to be managed. This option is mutually exclusive with the **-k** option and the **-p12** option. The name must consist of characters that are alphanumeric, national (@ x5B, # x7B, $ x7C) or period (.x4B). The first character must be alphabetic or national. Lowercase letters are allowed, but are folded to uppercase.

-x
: Specifies the number of days until the signed certificate expires and must be between 1 and 9999 days. The certificate expires in 365 days if this option is not specified. The expiration date cannot exceed February 6, 2106, at 23:59:59 UTC.

Results

If **gskkyman** is specified with no arguments the interactive menu-driven interface is used.

Usage

The **gskkyman** utility is used to manage a token, a key database and its associated request database, or to list the contents of a GSKIT CMS V4 key database or PKCS #12 file. Interactive menus are displayed if no command options are specified. Otherwise, the requested token/database/PKCS #12 file function is performed and the **gskkyman** utility exits.

Note: The ability to display the contents of a PKCS #12 file or a GSKIT CMS V4 key database is not supported through the interactive menu-driven interface.

If the command specifies the **-t** (token name) option, then the requested function is performed for the identified token. If the specified PKCS #11 token certificate contains a secure private key, then only display functions **-dc** and **-dcv** are supported. If the **gskkyman** utility supplies both the **-t** and **-l** (label name) options, then only the PKCS #11 certificate with the matching label is checked for a secure private key. If the certificate does not have a secure private key, then both the **-e** (export) or **-g** (sign) functions can be processed.

If the command specifies the **-p12** (PKCS #12 file) option on the display functions **-dc** or **-dcv**, if **-l** option is used, the certificate with matching label is displayed. If **-l** option is not used, all certificates within the file are displayed.

If the command does not specify the **-t** or the **-p12** option, then it is assumed that the function is to be performed for a key database. If the **-k** option, the **-t** option, and the **-p12** option are not supplied, the user is prompted for a key database file name.

If any combination of **-k**, **-t**, and **-p12** is specified, the command is rejected and an error message is displayed.

For commands applied to a GSKIT CMS V4 key database:

- If the command specifies a **-k** option on the display functions **-dc** or **-dcv**, if **-l** option is used, the certificate with matching label is displayed. If **-l** option is not used, all certificates within the file are displayed.

  If the command specifies a **-k** option on the display function **-dk**, the key database information is displayed.

For commands applied to a key database:

- The key database contains certificates and private keys and normally has the .kdb file name extension. The request database contains requests for new certificates and always has a .rdb file name extension. The database stash file contains the masked database password and always has a .sth file name extension. Access to these files should be restricted to the database owner.
- A certificate or request database consists of fixed-length records. The record length is specified when the database is created and must be large enough to contain the largest certificate entry. A record length of 5000 should be sufficient for most applications. The record length can be increased if necessary after the database is created.
- A temporary database file is created when a database is updated during **gskkyman** processing. The temporary database file is created using the same name as the database file with .new appended to the name. The database file is then rewritten and the temporary database file is deleted upon successful completion of the rewrite operation. The temporary database file is not deleted if an error occurs while rewriting the database file. If this happens, you can replace the database file with the temporary database file to recover from the error. If an error does occur and you do not rename or delete the temporary file, you receive an error on the next database update operation indicating the backup file exists.
- If all certificates in a key database are displayed with the **-dc** or **-dcv** command, then all certificates with private keys are outputted, followed by all certificates without private keys. When displaying all certificates in a token, the certificates are displayed in the order that is returned from the token so that certificates with private keys might be interspersed with certificates without private keys.
- Allow the **gskkyman** command line to display, export, and import certificates using RSA encryption keys and RSASSA-PSS signatures. **gskkyman** command line can also sign certificate requests with certificates using RSA keys and RSASSA-PSS signatures.

gskkyman command line mode examples

Command mode is entered when the **gskkyman** utility is entered with parameters. The requested token/ database function is performed and then the utility exits.

- Store the database password in the stash file

  ```
  gskkyman -s -k filename
  ```

  The database password is masked and written to the key stash file. The file name is the same as the key database file name but has an extension of '.sth'. You are prompted for the key database file name if the '-k' option is not specified. The '-t' option is invalid for the '-s' function.

- Export a certificate and the associated private key

  ```
  gskkyman -e -k filename -l label -p filename
  ```

  The certificate and associated private key that is identified by the record label are exported to a file in PKCS #12 Version 3 format using strong encryption. The default key is exported if the '-l' option is not specified. You are prompted for the key database file name if the '-k' and the '-t' option is not specified. You are prompted for the export file name if the '-p' option is not specified.

- Import a certificate and associated private key

  ```
  gskkyman -i -t token-name -l label -p filename
  ```

  A certificate and associated private key are imported from a file in PKCS #12 format. You are prompted for the label if the '-l' option is not specified. You are prompted for the key database file name if the '-k' and the '-t' option is not specified. You are prompted for the import file name if the '-p' option is not specified.

- Create a signed certificate for a certificate request

  ```
  gskkyman -g -x days -cr filename -ct filename -k filename -l label -kt keytype -ca -ic -pss
  ```

  The certificate request that is identified by the -cr parameter is processed and a signed certificate is created and written to the certificate file identified by the -ct parameter. The -x parameter specifies the number of days until the certificate expires and defaults to 365 days. The certificate is signed using the default key if the -l parameter is not specified. You are prompted for the key database file name if the '-k' option is not specified. You are prompted for the certificate request file name if the '-cr' option is not specified. You are prompted for the signed certificate file name if the '-ct' option is not specified.

  The signed certificate is an end user certificate unless the -ca option is specified. A certification authority certificate has basic constraints and key usage extensions that allow the certificate to be used to sign other certificates and certificate revocation lists. An end user certificate has basic constraints and key usage extensions that allow the certificate to be used as follows:

  - An RSA key can be used for authentication, digital signature, and data encryption.
  - A DSS key can be used for authentication and digital signature.
  - An ECC key depends on the keytype option supplied. A general ECC key (-kt ecgen) can be used for authentication, digital signature, and key agreement. An ECDSA key (-kt ecdsa) can be used for authentication and digital signature. An ECDH key (-kt ecdh) can be used for key agreement. The default option is ecgen.

  Any certificate can be used to sign the new certificate if the certificate has a private key, the basic constraints certificate extension (if present) has the CA indicator set, and the key usage certificate extension (if present) allows signing certificates except a DSA signing certificate cannot sign a ECC certificate, an ECC signing certificate cannot sign a DSA certificate, and an RSASSA-PSS signed certificate must be signed with a RSA signing certificate.

  The signature algorithm that is used to sign the new certificate is based on the key algorithm of the signing certificate. If an RSASSA-PSS signature is to be used, the -pss option must be specified and is only supported when the certificate request is either a RSA (2048 - 4096) or ECC (256 - 521) request and the signing certificate has a RSA private key. An RSA signature uses the most secure and compatible SHA-based hash in use in the signature algorithm of either the signing certificate or the certificate request. An RSASSA-PSS signature uses the suggested digest for the key size of the signing RSA private key, as specified in Table 4 on page 14. A DSA signature with a 1024-bit DSA key uses SHA-1. A DSA signature with a 2048-bit DSA key uses SHA-256. An ECC signature uses the suggested digest for the key size of the ECC private key, as specified in Table 2 on page 13.

Possible signature algorithms are:

– x509_alg_sha1WithRsaEncryption
– x509_alg_sha224WithRsaEncryption
– x509_alg_sha256WithRsaEncryption
– x509_alg_sha384WithRsaEncryption
– x509_alg_sha512WithRsaEncryption
– x509_alg_dsaWithSha1
– x509_alg_dsaWithSha256
– x509_alg_ecdsaWithSha256
– x509_alg_ecdsaWithSha384
– x509_alg_ecdsaWithSha512
– x509_alg_mgf1Sha256WithRsaSsaPss
– x509_alg_mgf1Sha384WithRsaSsaPss
– x509_alg_mgf1Sha512WithRsaSsaPss

The certificate file contains the generated X.509 certificate in DER-encoded Base64 format if the -ic option is not specified. The certificate file contains the generated X.509 certificate and the certification chain certificates as a PKCS #7 message in Base64 format if the -ic option is specified.

- Display all certificates in a key database

```
gskkyman -dc -k filename
```

After you are prompted for the key database password, the certificates will be displayed. You are prompted for the key database file name if the **-k** option is not specified. Because of the number of certificates that can exist in a key database file, it is suggested that you redirect the output to a file. This allows for easy review of the certificates and any post-processing of the certificate output.

- Display key database expiration date:

```
gskkyman -dk -k filename
```

After you are prompted for the key database password, the full key database path and file name, expiration date, record length, and database type are displayed. You are prompted for the key database file name if the -k option is not specified.

gskkyman command line mode displays

Command mode is entered when **gskkyman** is entered with parameters. The requested token/database function is performed and then the utility exits.

- **gskkyman** command-mode key database file display

When the key database password is correctly entered:

Command:

```
gskkyman -dk -k example.kdb
```

Output:

```
Database: /home/sufwl1/ssl_cmd/example.kdb
Expiration Date: 2025/12/02  10:11:12
Record length: 5000
Type: non-FIPS
```

Command:

```
gskkyman -dk -k GSKIT.kdb
```

Output:

```
Database: /home/suapca/kdbs/GSKIT.kdb
Expiration: None
Record length: 5000
Type: GSKIT CMS V4
```

- **gskkyman** command-mode certificate display

  Command:

```
gskkyman -dc -k example.kdb -l 'Test User'
```

  Output for a single certificate:

```
Label:
        <Test User>
Trusted:
        Yes
Version:
        3
Serial number:
        57addd1b00073381
Issuer's Name:
        <CN=Test CA,OU=Test unit,O=IBM,L=Endicott,ST=NY,C=US>
Subject's Name:
        <CN=Test User,O=IBM,L=Endicott,ST=NY,C=US>
Effective Date:
        2016/08/12 14:28:43
Expiration Date:
        2022/08/11 13:58:22
Signature algorithm:
        sha256WithRsaEncryption
Issuer unique ID:
        None
Subject unique ID:
        None
Public key algorithm:
        rsaEncryption
Public key size:
        2048
Public key:
        30 82 01 0A 02 82 01 01 00 CE B6 F9 B1 04 F7 4E
        04 83 75 64 5D 22 35 E8 12 F8 DC A9 7A 0D BC A8
        32 BD 36 BD CD 80 63 C6 FD 4D D1 41 C9 68 C4 AA
        EE 0E 02 B6 7C DD 67 24 33 2F A9 71 5D FA F7 15
        E7 D9 10 2A CD 01 6A B7 34 E6 96 CA 6F B8 90 E3
        4E C9 81 A6 16 66 25 11 7F 14 17 62 DD 52 7B D1
        D0 52 61 E4 ED 3D DB 4F EE 98 D3 83 E2 82 8F 1A
        B8 20 70 93 1D 06 BD 12 FE E1 C6 24 F2 C6 C7 9F
        6D CB 4E E5 22 2F 3A D8 94 6D B5 40 24 37 D8 51
        4A F0 F6 E2 43 15 17 B9 E7 F5 BF 74 4F 48 23 05
        E1 47 C3 B8 3F 47 35 BA 89 4E 1A 58 2B 51 4C 40
        E2 2D 08 D7 A4 AD 53 7C B9 D0 A0 FF 16 F2 98 96
        C9 9C 19 3F 71 74 2D 83 D9 BD 5C B7 3A AE CD 37
        FA C8 A6 B5 60 BF 15 8C 0B A3 51 38 1F A5 3E 1A
        B0 51 67 22 66 4B AB 1C D1 78 4A C5 45 39 0B 9B
        1A 13 73 F4 88 18 31 63 E7 F3 48 07 A0 D2 4B D0
        A6 76 B9 E7 E9 E5 2F A7 AD 02 03 01 00 01
Private key:
        Yes
Default key:
        No
```

```
Certificate extensions:
        4
```

- **gskkyman** command-mode PKCS #11 token certificate display

  Command:

  ```
  gskkyman -dc -t my.token -l rsa2048CASecure
  ```

  Output:

  ```
  Label:
          <rsa2048CASecure>
  Trusted:
          Yes
  Version:
          3
  Serial number:
          57addae000031e31
  Issuer's Name:
          <CN=rsa2048CASecure,OU=ibm,O=stg,C=US>
  Subject's Name:
          <CN=rsa2048CASecure,OU=ibm,O=stg,C=US>
  Effective Date:
          2016/08/12 14:19:12
  Expiration Date:
          2022/08/11 14:19:12
  Signature algorithm:
          sha256WithRsaEncryption
  Issuer unique ID:
          None
  Subject unique ID:
          None
  Public key algorithm:
          rsaEncryption
  Public key size:
          2048
  Public key:
          30 82 01 0A 02 82 01 01 00 E3 22 9D E9 88 8E 46
          6C B9 39 78 C7 FD F3 0B 2B 38 F2 DF 8C 40 AF 26
          93 AC 11 94 F0 C9 37 AB BD 86 3B 6C 62 59 06 A4
          C6 E0 4A 7A ED CC 0E D9 A7 77 AD E0 15 67 24 94
          CE 1D 95 31 02 EA C8 9E 34 37 BB EB F9 16 55 06
          C1 CF 2E 86 75 F4 46 95 A4 A3 CA B8 66 FC 9E D1
          EC E8 86 FC B2 59 D6 FE 3D 07 BF 1C 29 E0 82 E2
          74 0F 14 91 F6 42 DC 59 1D 49 66 0C 78 60 7D 9F
          B8 2E 82 2D 68 F6 96 A1 33 AF 87 11 32 7A 7D 31
          6A 53 90 A5 7C 8A D0 8D 6E 90 C8 7C B4 F4 42 52
          43 31 86 35 78 CC 11 23 60 62 4F 71 3D 43 85 A5
          A4 CA D6 F7 6F 65 C3 E1 88 E5 5C 62 B2 CF BC 11
          56 43 29 A2 E3 E1 20 E9 3B 7E C5 4E 7B A8 BA CA
          A1 39 5E 7D 7D B7 C2 69 BF 4B EB 80 7D 53 CC FB
          CB F0 6C 54 BF 96 39 73 D6 FB 30 21 A1 E3 4D A4
          4D CF AF A3 96 EB 8A 29 34 85 FD 8B C3 42 92 11
          03 3F 05 97 87 A5 62 D7 1B 02 03 01 00 01
  Private key:
          Yes
  Private key type:
          Secure
  Default key:
          Yes
  Certificate extensions:
          4
  ```

- **gskkyman** command-mode certificate display (verbose)

  Command:

  ```
  gskkyman -dcv -k example.kdb -l 'Test User'
  ```

  Verbose output for a single certificate:

  ```
  Label:
          <Test User>
  Trusted:
          Yes
  Version:
  ```

```
        3
Serial number:
        57addd1b00073381
Issuer's Name:
        <CN=Test CA,OU=Test unit,O=IBM,L=Endicott,ST=NY,C=US>
Subject's Name:
        <CN=Test User,O=IBM,L=Endicott,ST=NY,C=US>
Effective Date:
        2016/08/12 14:28:43
Expiration Date:
        2022/08/11 13:58:22
Signature algorithm:
        sha256WithRsaEncryption
Issuer unique ID:
        None
Subject unique ID:
        None
Public key algorithm:
        rsaEncryption
Public key size:
        2048
Public key:
        30 82 01 0A 02 82 01 01 00 CE B6 F9 B1 04 F7 4E
        04 83 75 64 5D 22 35 E8 12 F8 DC A9 7A 0D BC A8
        32 BD 36 BD CD 80 63 C6 FD 4D D1 41 C9 68 C4 AA
        EE 0E 02 B6 7C DD 67 24 33 2F A9 71 5D FA F7 15
        E7 D9 10 2A CD 01 6A B7 34 E6 96 CA 6F B8 90 E3
        4E C9 81 A6 16 66 25 11 7F 14 17 62 DD 52 7B D1
        D0 52 61 E4 ED 3D DB 4F EE 98 D3 83 E2 82 8F 1A
        B8 20 70 93 1D 06 BD 12 FE E1 C6 24 F2 C6 C7 9F
        6D CB 4E E5 22 2F 3A D8 94 6D B5 40 24 37 D8 51
        4A F0 F6 E2 43 15 17 B9 E7 F5 BF 74 4F 48 23 05
        E1 47 C3 B8 3F 47 35 BA 89 4E 1A 58 2B 51 4C 40
        E2 2D 08 D7 A4 AD 53 7C B9 D0 A0 FF 16 F2 98 96
        C9 9C 19 3F 71 74 2D 83 D9 BD 5C B7 3A AE CD 37
        FA C8 A6 B5 60 BF 15 8C 0B A3 51 38 1F A5 3E 1A
        B0 51 67 22 66 4B AB 1C D1 78 4A C5 45 39 0B 9B
        1A 13 73 F4 88 18 31 63 E7 F3 48 07 A0 D2 4B D0
        A6 76 B9 E7 E9 E5 2F A7 AD 02 03 01 00 01
Private key:
        Yes
Default key:
        No
Critical Extension:
        keyUsage:
                Digital signature
                Non-repudiation
                Key encipherment
                Data encipherment
Non-critical Extension: 1
        subjectAltName:
                EMAIL:
                        <test@mycompany.com>
Non-critical Extension: 2
        subjectKeyIdentifier:
                D1 71 EC 48 C4 FB A5 C0 C8 0A 96 A2 E7 23 03 D7
                92 64 A3 A4
Non-critical Extension: 3
        authorityKeyIdentifier:
                Key ID:
                        7F C0 9E 88 99 6D E9 F2 12 2E 18 4B C6 EA 4A E3
                        AF AA 01 2E
```

Chapter 11. SSL started task

The SSL started task (GSKSRVR) provides sysplex session cache support, sysplex session ticket cache support, dynamic trace support, and notification when changing from hardware to software cryptography. The SSL started task is an optional component of System SSL and does not need to be configured and started to use System SSL.

The default home directory for the SSL started task is /etc/gskssl/server. A different home directory can be specified by changing the definition of the HOME environment variable in the GSKSRVR procedure. The SSL started task reads the envar file in the home directory to set the environment variables. This file is a variable-length file where each line consists of a variable name and variable value separated by '='. Trailing blanks are removed from the variable value. Blanks lines and lines beginning with '#' are ignored.

GSKSRVR environment variables

These environment variables are processed by the System SSL started task:

GSK_LOCAL_THREADS
Specifies the maximum number of threads which is used to handle program call requests from SSL applications running on the same system as the GSKSRVR started task. The default value is 5 and the minimum value is 2. The default of 5 is used if a valid value is not specified.

GSK_SESSION_TICKET_CACHE_SIZE
Specifies the size of the sysplex session ticket cache in megabytes and is between 1 and 512 with a default of 256. The default of 256 is used if a valid value is not specified.

GSK_SESSION_TICKET_CACHE_TIMEOUT
Specifies the sysplex session ticket cache entry timeout in minutes and is between 1 and 10080 (7 days) with a default of 10. The default of 10 is used if a valid value is not specified.

GSK_SESSION_TICKET_CACHE_NUM_TICKETS
Specifies the number of session tickets that are stored in the sysplex session ticket cache entry per session and is between 16 and 320 with a default of 16. If a multiple of 16 is not specified, the number of session tickets per cache entry is rounded to the nearest multiple of 16. If number specified is less than 16 or greater than 320, the default of 16 is used.

GSK_SIDCACHE_SIZE
Specifies the size of the sysplex session cache in megabytes and is between 1 and 512 with a default of 256. The default of 256 is used if a valid value is not specified.

GSK_SIDCACHE_TIMEOUT
Specifies the sysplex session cache entry timeout in minutes and is between 1 and 1440 with a default of 60. The default of 60 is used if a valid value is not specified.

GSK_FIPS_STATE
Specifies that the System SSL started task is to execute in FIPS mode. The only value that is supported is **GSK_FIPS_STATE_ON**. If any other value is specified, message GSK01054E is issued with a status code of zero, and GSKSRVR executes in non-FIPS mode.

In order for the started task to perform sysplex session ID caching for FIPS mode application servers, the envar file must contain **GSK_FIPS_STATE=GSK_FIPS_STATE_ON**. If the started task executes in FIPS mode, then message GSK01057I is output to STDOUT. See Chapter 4, "System SSL and FIPS 140-2," on page 19 for setup requirements necessary to execute in FIPS mode.

To have GSKSRVR execute in non-FIPS mode and provide sysplex session ID caching for non-FIPS application servers, remove or comment out this environment variable. GSKSRVR starts in non-FIPS mode without issuing GSK01054E or GSK01057I messages.

Note: The **GSK_SESSION_TICKET_CACHE_SIZE**, **GSK_SESSION_TICKET_CACHE_TIMEOUT**, and **GSK_SESSION_TICKET_CACHE_NUM_TICKETS** environment variables were added in z/OS V2R5 with APAR OA63252 and later releases in the sample gsksrvr.envar. If the System SSL started task was configured prior to these environment variables being introduced, the environment variables file may need to be updated to add these environment variables if non-default values need to be used. Additionally, the default for the GSK_SIDCACHE_SIZE has been updated from 20 megabytes to 256 megabytes in z/OS V2R5 with APAR OA63252 and later releases. An older version of the sample gsksrvr.envar file continues to use 20 megabytes by default. You may want to consider updating your current gsksrvr.envar file to use a larger GSK_SIDCACHE_SIZE setting. See "Configuring the SSL started task" on page 612 for information about the sample gsksrvr.envar file.

Configuring the SSL started task

1. Create the home directory for the SSL started task (the default is /etc/gskssl/server)
2. Copy the sample envar file (gsksrvr.envar) from /usr/lpp/gskssl/examples/ to /etc/gskssl/server/ with a new file name of "envar". By default, the full path is /etc/gskssl/server/envar (change the directory name to match the home directory created). Modify the LANG, TZ, and NLSPATH values to meet local installation requirements.
3. Copy the sample started procedure from GSK.SGSKSAMP(GSKSRVR) to SYS1.PROCLIB(GSKSRVR)

   Note: The sample started task procedure routes informational messages, such as GSK01001I, to standard out, while error messages, such as GSK01015E are routed to standard error. If you want to route informational and error messages to the same place in the job log, change:

   ```
   // / 1>DD:STDOUT 2>DD:STDERR')
   ```

   to

   ```
   // / >DD:STDOUT 2>&1')
   ```

4. Create the GSKSRVR user and associate it with the GSKSRVR started procedure. Replace 'nnnnnn' in the ADDUSER command with a non-zero value which is not assigned to another user.

   ```
   ADDUSER GSKSRVR DFLTGRP(SYS1) NOPASSWORD OMVS(UID(nnnnnn) PROGRAM(/bin/sh) HOME(/etc/gskssl/
   server))

   RDEFINE STARTED GSKSRVR.** STDATA(USER(GSKSRVR) GROUP(SYS1) TRUSTED(YES))

   SETROPTS RACLIST(STARTED) REFRESH
   ```

5. Ensure that the pdsename.SIEALNKE and CEE.SCEERUN data sets are APF-authorized and are either in the link list concatenation or are specified as a STEPLIB for the GSKSRVR procedure.
6. Optionally, set up a message processing exit to automatically start the GSKSRVR started task. The GSK.SGSKSAMP(GSKMSGXT) program is a sample message processing exit for this purpose. To activate the exit, add this to the appropriate MPFLSTxx member in SYS1.PARMLIB.

   ```
   BPXI004I,SUP(NO),USEREXIT(STARTSSL)
   ```

   This starts GSKSRVR when OMVS initialization is complete, assuming the GSKMSGXT program was linked as STARTSSL and placed in a LNKLST data set.
7. Optionally, set up an automatic restart management (ARM) policy for the GSKSRVR started task if the default ARM policy values are not appropriate. The element type is SYSSSL and should be assigned to restart level 2. The element name is GSKSRVR_sysname. For example, the element name for the GSKSRVR started task on system DCESEC4 would be GSKSRVR_DCESEC4. Since the normal operating mode is to run the GSKSRVR started task on each system in the sysplex, the GSKSRVR started task registers with ARM to be restarted only if the started task fails and not if the current system fails. The TERMTYPE parameter of the ARM policy can be used to override this registration if you want.
8. If access to the ICSF callable services are protected with CSFSERV class profiles on your system, the GSKSRVR user ID might need to be given READ authority to call the ICSF CSFIQA and CSFPPRF callable services. These services are protected by the CSFIQA and CSFRNG profiles. If these callable services are protected with a generic CSF* profile in the CSFSERV class, access can be granted by entering:

```
PERMIT CSF* CLASS(CSFSERV) ID(GSKSRVR) ACCESS(READ)
SETROPTS RACLIST(CSFSERV) REFRESH
```

Server operator commands

These operator commands are supported by the System SSL server:

STOP GSKSRVR or P GSKSRVR
Causes an orderly shutdown of the server.

MODIFY GSKSRVR,parameters or F GSKSRVR,parameters
Causes a command to be executed by the server. Some parameters are:

DISPLAY CRYPTO
Displays the available encryption algorithms, whether hardware cryptographic support is available and the maximum encryption key size. '' is displayed if the encryption algorithm is not available.

This command can be abbreviated as 'D CRYPTO'.

DISPLAY LEVEL
Displays the current System SSL service level.

This command can be abbreviated as 'D LEVEL'.

DISPLAY SIDCACHE
Displays the current and maximum data space sizes in megabytes followed by the session ID cache users, the number of cache entries for each user, and the maximum cache entry size in bytes. The count includes expired cache entries until they are removed from the cache during an update to the hash list containing the expired entry. The maximum cache entry size specifies the largest cache entry size for that user even if that specific entry has been removed from the cache.

Each GSKSRVR started task maintains its own session cache for sessions created on that system. The 'DISPLAY SIDCACHE' command must be issued for each started task to display the cache entries for the entire sysplex. This can be done by issuing 'RO *ALL,F GSKSRVR,DISPLAY SIDCACHE'.

This command can be abbreviated as 'D SIDCACHE'.

DISPLAY STATS,SIDCACHE
Displays the current time, start time, last reset time, number of cache resets, current cache size, maximum cache size, cache timeout, add and retrieval statistics for the session ID cache. The add and retrieval statistics include session cache requests targeted for the local GSKSRVR started task and session cache requests from other GSKSRVR instances in the sysplex. See "Interpreting the session cache statistics" on page 615 for more information.

Each GSKSRVR started task maintains its own session cache for sessions created on that system. The 'DISPLAY STATS,SIDCACHE' command must be issued for each started task to display the cache entries for the entire sysplex. This can be done by issuing 'RO *ALL,F GSKSRVR,DISPLAY STATS,SIDCACHE'.

This command can be abbreviated as 'D ST,SIDCACHE'.

DISPLAY STATS,TICKETCACHE
Displays the current time, start time, last reset time, number of cache resets, current cache size, maximum cache size, cache timeout, cache number of tickets, add, update, and retrieval statistics for the session ticket cache. The add, update, and retrieval statistics include session ticket cache requests targeted for the local GSKSRVR started task and session ticket cache requests from other GSKSRVR instances in the sysplex. See "Interpreting the session ticket cache statistics" on page 619 for more information.

Each GSKSRVR started task maintains its own session ticket cache for sessions created on that system. The 'DISPLAY STATS,TICKETCACHE' command must be issued for each started task to display the ticket cache entries for the entire sysplex. This can be done by issuing 'RO *ALL,F GSKSRVR,DISPLAY STATS,TICKETCACHE'.

This command can be abbreviated as ‘D ST,TICKETCACHE’.

DISPLAY TICKETCACHE

Displays the current and maximum data space sizes in megabytes followed by the session ticket cache users, the number of cache entries for each user, and the maximum cache entry size in bytes. The count includes expired session ticket cache entries until they are removed from the cache during an update to the hash list containing the expired entry. The maximum cache entry size specifies the largest cache entry size for that user even if that specific entry has been removed from the cache.

Each GSKSRVR started task maintains its own session ticket cache for sessions created on that system. The 'DISPLAY TICKETCACHE' command must be issued for each started task to display the cache entries for the entire sysplex. This can be done by issuing 'RO *ALL,F GSKSRVR,DISPLAY TICKETCACHE'.

This command can be abbreviated as 'D TICKETCACHE'.

DISPLAY XCF

Displays the status of all instances of the GSKSRVR started task in the sysplex.

This command can be abbreviated as 'D XCF'.

RESET STATS,SIDCACHE

Resets the add and retrieval statistics for the session cache. Prior to resetting the statistics, the current time, start time, last reset time, number of cache resets, current cache size, maximum cache size, cache timeout, add and retrieval statistics are displayed for the session ID cache. The add and retrieval statistics include session cache requests targeted for the local GSKSRVR started task and session cache requests from other GSKSRVR instances in the sysplex. See “Interpreting the session cache statistics” on page 615 for more information.

Each GSKSRVR started task maintains its own session cache for sessions created on that system. The 'RESET STATS,SIDCACHE’ command must be issued for each started task to display and reset the cache entries for the entire sysplex. This can be done by issuing 'RO *ALL,F GSKSRVR,RESET STATS,SIDCACHE'.

After successfully resetting the statistics, the last reset time is set to the current time and the number of resets is incremented by 1.

This command can be abbreviated as ‘R ST,SIDCACHE’.

RESET STATS,TICKETCACHE

Resets the add, update, and retrieval statistics for the session ticket cache. Prior to resetting the statistics, the current time, start time, last reset time, number of cache resets, current cache size, maximum cache size, cache timeout, cache number of tickets, add, update, and retrieval statistics are displayed for the session ticket cache. The add, update, and retrieval statistics include session ticket cache requests targeted for the local GSKSRVR started task and session ticket cache requests from other GSKSRVR instances in the sysplex. See “Interpreting the session ticket cache statistics” on page 619 for more information.

Each GSKSRVR started task maintains its own session ticket cache for sessions created on that system. The 'RESET STATS,TICKETCACHE' command must be issued for each started task to display and reset the ticket cache entries for the entire sysplex. This can be done by issuing 'RO *ALL,F GSKSRVR,RESET STATS,TICKETCACHE'.

After successfully resetting the statistics, the last reset time is set to the current time and the number of resets is incremented by 1.

This command can be abbreviated as ‘R ST, TICKETCACHE’.

STOP

Causes an orderly shutdown of the server. This is the same as entering the "STOP GSKSRVR" command.

TRACE OFF

Turns off tracing for the System SSL started task.

TRACE ON,level

Turns on tracing for the System SSL started task. The trace output is written to the file specified by the GSK_TRACE_FILE environment variable or to the default trace file if the GSK_TRACE_FILE environment variable is not defined. The level value specifies the trace level. See the descriptions of the GSK_TRACE and GSK_TRACE_FILE environment variables for more information about SSL tracing.

Sysplex session cache support

The sysplex session cache support makes SSL server session information available across the sysplex. An SSL session established with a server on one system in the sysplex can be resumed using a server on another system in the sysplex if the SSL client presents the session identifier obtained for the first session when initiating the second session. This also works with multiple servers running on the same system that are presented to the client as a single server. A server executing in FIPS mode cannot resume a session cached in non-FIPS mode. SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 protocol server session information can be stored in the sysplex session cache while SSL V2 server session information and all client session information is stored only in the local SSL cache for the application process.

A client which established a TLS V1.0, TLS V1.1, or TLS V1.2 protocol session with negotiated TLS extensions to a server can only be resumed on a server which supports the same set of TLS extensions established in the original session. For example, if the original session negotiates the use of the maximum fragment length TLS extension, but the session is later resumed with a server that does not support the maximum fragment length TLS extension, a full rehandshake occurs.

In order to use the sysplex session cache, each system in the sysplex must be using the same external security manager (for example, z/OS Security Server RACF) and a user ID on one system in the sysplex must represent the same user on all other systems in the sysplex (that is, user ID ZED on System A has the same access rights as user ID ZED on System B). The external security manager must support the RACROUTE REQUEST=EXTRACT,TYPE=ENVRXTR and RACROUTE REQUEST=FASTAUTH functions.

The sysplex session cache must be enabled for each application server that is to use the support. This can be done by defining the GSK_SYSPLEX_SIDCACHE environment variable or by calling the **gsk_attribute_set_enum()** routine to set the GSK_SYSPLEX_SIDCACHE attribute. The session information for each new SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 session created by the SSL server is then stored in the sysplex session cache and can be referenced by other SSL servers in the sysplex. The RACF user associated with the SSL server becomes the owner of the session information. Any SSL server running with the same RACF user can access the session information. SSL servers running with a different RACF user can access the session information if they have at least READ access to the GSK.SIDCACHE.<owner> profile in the FACILITY class.

For example, session information created by RACF user APPLSRV1 can be accessed by RACF user APPLSRV2 if APPLSRV2 has READ access to the GSK.SIDCACHE.APPLSRV1 profile in the FACILITY class. These RACF commands grant this access:

```
RDEFINE FACILITY GSK.SIDCACHE.APPLSRV1 UACC(NONE)
PERMIT GSK.SIDCACHE.APPLSRV1 CLASS(FACILITY) ID(APPLSRV2) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Interpreting the session cache statistics

The F GSKSRVR,DISPLAY STATS,SIDCACHE operator command displays session cache statistics. These statistics can be used to provide insights into how the sysplex session ID cache is performing. These insights give you the ability to do further tuning on the cache and whether the System SSL server applications are able to successfully use cached SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 sessions as expected.

The following is example output from the F GSKSRVR,DISPLAY STATS,SIDCACHE operator command. This display is written to the console and to the GSKSRVR joblog. Similar output is also displayed in response to a F GSKSRVR,RESET STATS,SIDCACHE operator command prior to the session cache statistics being reset.

```
   GSK01079I Session cache
    statistics

  1 Current time:        2023/04/10 10:43:25 EDT
  2 Start time:          2023/04/10 10:29:43 EDT
  3 Reset time:          2023/04/10 10:30:06 EDT
  4 Number of resets:    1

  5 Current cache size:  5M
  6 Maximum cache size:  5M
  7 Cache timeout:       1


   Add statistics                                    Count
   --------------                                    -----
  8 Requested:                                        7028
  9 Successful:                                       5112
 10 Failed adds due to full dataspace:                1916
 11 Success percentage:                            72.738%
 12 Replaced entries:                                 2556
 13 Purged entries:                                      0


   Retrieval statistics                              Count
   --------------------                              -----
 14 Requested:                                       75181
 15 Successful:                                      75161
 16 Failed expired:                                      4
 17 Failed not found:                                   16
 18 Failed not authorized:                               0
 19 Success percentage:                            99.973%
```

Notes:

- The superscript numbers in the example above do not appear in the actual command output. They are used to depict what each line of the statistics means. These numbers are referenced below in the statistics display explanations.
- The add and retrieval statistics are reset to 0 when a F GSKSRVR,RESET STATS,SIDCACHE command is performed. The number of resets is incremented by 1 and the reset time is set to the time that the command is ran. If the GSKSRVR is running a very long time, it is possible for these statistics to wrap. In these cases, it is recommended that you perform periodic resets so that the gathered cache statistics have more meaning.

General statistics:

1

Indicates the current time that the statistics were displayed.

2

Indicates the time that the System SSL started task, GSKSRVR, was started.

3

Indicates the time that the session cache statistics were last reset. If the statistics were not reset since the GSKSRVR started, the start and reset times are the same. In the example above, session cache statistics have been reset after the GSKSRVR has started.

4

Indicates the number of cache statistics resets that have occurred since the GSKSRVR has started. In this example, there has been one reset of the session cache statistics.

5

The current cache size in megabytes. The cache grows in increments of 1MB up to the maximum size specified in the GSK_SIDCACHE_SIZE environment variable. In this example, the current and maximum cache sizes are the same and may indicate a full cache scenario.

6

The maximum cache size in megabytes. This is the value specified in the GSK_SIDCACHE_SIZE environment variable.

7

The cache timeout setting in minutes. This is the value specified in the GSK_SIDCACHE_TIMEOUT environment variable. In this example, the cache timeout setting is 1 minute.

Add statistics: These statistics depict how successful the local GSKSRVR instance can cache new entries. The initial cache entry add consists of caching the session information in the cache after completing a full SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 handshake. These statistics are only updated for adds on the local GSKSRVR instance.

8

Requested – Indicates the number of successful and failure add requests.

9

Successful – Indicates the number of successful add requests.

10

Failed adds due to full dataspace – Indicates the number of adds that failed due to the GSKSRVR dataspace being full. There is no room in the cache to add a new cache entry. The session information cannot be added. This can indicate that the maximum cache size specified in the GSK_SIDCACHE_SIZE is not large enough to store the session cache entries. If this number is large, it is an indication that a larger value may be needed for the GSK_SIDCACHE_SIZE environment variable. When session cache add failures occur, this can ultimately result in additional full handshakes occurring in System SSL server applications.

In this example, the cache full condition has probably occurred because the session cache data space has been exceeded as evidenced by the 1916 failed adds. This can be inferred from the current and maximum cache sizes in the general statistics.

11

Success percentage – This indicates a percentage of successful versus failure requests with a precision of three decimal points.

12

Replaced entries – This indicates how many older cache entries have been replaced in the cache during an add. In this example, 2556 older cache entries were replaced by newer cache entries. The number of replaced entries do not count directly towards the success percentage. However, if there are a large number of replacements, future cache retrievals may not work as expected.

13

Purged entries - This indicates how many cache entries were purged or removed from the cache due to be expired or previously marked for deletion. In this example, there were no cache entries removed from the cache due to expiration or being marked for deletion. The number of purged entries do not count directly towards the success percentage.

Retrieval statistics – These statistics depict how successful session cache retrievals are. Cache retrievals are performed when a System SSL server application attempts to retrieve a session ID presented by a client during a SSL V3, TLS V1.0, TLS V1.1, or TLS V1.2 cached handshake attempt. The System SSL server application presents the session ID from the client to retrieve the cached session information. These statistics are updated for local targeted requests and incoming requests from other GSKSRVR instances in the sysplex.

14

Requested – Indicates the number of successful and failure retrieval requests.

15

Successful – Indicates the number of successful retrieval requests.

16

Failed expired – This indicates on the retrieval of a cache entry that it has already expired and it cannot be retrieved.

17

Failed not found – The cache entry that is attempted to be retrieved cannot be found in the cache.

18

Failed not authorized – The server application's userid that is attempting to retrieve the existing cache entry does not have the appropriate authority to the GSK.SIDCACHE.<*userid*> profile in the FACILITY class. These failures may be expected if the appropriate authority was not given to the cache entries created by other userids.

19

Success percentage - This indicates a percentage of successful versus failure requests with a precision of three decimal points.

If the F GSKSRVR,DISPLAY SIDCACHE command is ran, its output shows the current and maximum size of the dataspace followed by lines displaying the cache users, the number of cache users, and the maximum cache entry size in bytes. This output shows what applications have allocated entries within the cache and can be used for further tuning of the maximum cache size or timeout settings based on an organization's security policy. The actual cache entry size is rounded up to the nearest multiple of 1024 bytes (1k) from what is depicted in the output.

```
GSK01032I Session cache status
User              Count                Max size
*DSPACE*            5/5                     N/A
APPL1              2546                    1120
APPL2                10                    1120
```

Sysplex session ticket cache support

The sysplex session ticket cache support stores the TLS V1.3 session ticket information available across the sysplex. A TLS V1.3 session established with a server on one system in the sysplex can be resumed using a server on another system in the sysplex if the TLS client includes the session ticket obtained for the first session when initiating the second session. This also works with multiple servers running on the same system that are presented to the client as a single server. The session information for the TLS V1.3 session is stored in the GSKSRVR and it is made available to other like-servers in the sysplex. Without access to the session ticket information, other like-servers in the sysplex would be unable to resume a previously established TLS V1.3 session and would need to perform a full TLS V1.3 handshake.

A client which established a TLS V1.3 session with negotiated TLS extensions to a server can only be resumed on a server which supports the same set of TLS extensions established in the original session. For example, if the original session negotiates the use of the maximum fragment length TLS extension, but the session is later resumed with a server that does not support the maximum fragment length TLS extension, a full handshake occurs.

In order to use the sysplex session ticket cache, each system in the sysplex must be using the same external security manager (for example, z/OS Security Server RACF) and a user ID on one system in the sysplex must represent the same user on all other systems in the sysplex (that is, user ID ZED on System A has the same access rights as user ID ZED on System B). The external security manager must support the RACROUTE REQUEST=EXTRACT,TYPE=ENVRXTR and RACROUTE REQUEST=FASTAUTH functions.

The sysplex session ticket cache must be enabled for each application server that is to use the support. This can be done by defining the GSK_SYSPLEX_SESSION_TICKET_CACHE environment variable or by calling the **gsk_attribute_set_enum()** routine to set the GSK_SYSPLEX_SESSION_TICKET_CACHE attribute. The session information for each new TLS V1.3 session created by the SSL server is then stored in the sysplex session ticket cache and can be referenced by other SSL servers in the sysplex. The RACF user associated with the TLS server becomes the owner of the session ticket information. Any TLS server running with the same RACF user can access the session ticket information during TLS V1.3 resumption attempts. TLS servers running with a different RACF user can access the session information if they have at least READ access to the GSK.SIDCACHE.<*owner*> profile in the FACILITY class.

For example, session ticket cache information created by RACF user APPLSRV1 can be accessed by RACF user APPLSRV2 if APPLSRV2 has READ access to the GSK.SIDCACHE.APPLSRV1 profile in the FACILITY class. These RACF commands grant this access:

```
RDEFINE FACILITY GSK.SIDCACHE.APPLSRV1 UACC(NONE)
PERMIT GSK.SIDCACHE.APPLSRV1 CLASS(FACILITY) ID(APPLSRV2) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Interpreting the session ticket cache statistics

The F GSKSRVR,DISPLAY STATS,TICKETCACHE operator command displays session ticket cache statistics. These statistics can be used to provide insights into how the cache is performing. These insights may give you the ability to do further tuning on the cache and whether the System SSL server applications are able to successfully resume TLS V1.3 sessions as expected.

The following is example output from the F GSKSRVR,DISPLAY STATS,TICKETCACHE operator command. This output is written to the console and to the GSKSRVR joblog. Similar output is also displayed in response to a F GSKSRVR,RESET STATS,TICKETCACHE operator command prior to the session ticket cache statistics being reset.

```
   GSK01077I Session ticket cache
    statistics

 1 Current time:        2023/04/10 09:41:23 EDT
 2 Start time:          2023/04/10 09:27:39 EDT
 3 Reset time:          2023/04/10 09:29:26 EDT
 4 Number of resets:    1

 5 Current cache size:  13M
 6 Maximum cache size:  13M
 7 Cache timeout:         1
 8 Cache num tickets:    32


   Add statistics                                Count
   --------------                                -----
 9 Requested:                                     6020
10 Successful:                                    4418
11 Failed full dataspace:                         1602
12 Success percentage:                         73.389%
13 Replaced entries:                              1924
14 Purged entries:                                   0


   Update statistics                             Count
   -----------------                             -----
15 Requested:                                    47239
16 Successful:                                   47238
17 Failed expired:                                   1
18 Failed not found:                                 0
19 Failed not authorized:                            0
20 Success percentage:                         99.998%


   Retrieval statistics                          Count
   --------------------                          -----
21 Requested:                                    47251
22 Successful:                                   47239
23 Failed expired:                                   4
24 Failed not found:                                 8
25 Failed not authorized:                            0
26 Success percentage:                         99.975%
```

Notes:

- The superscript numbers in the example above do not appear in the actual command output. They are used to depict what each line of the statistics means. These numbers are referenced below in the statistics display explanations.
- The add, update, and retrieval statistics are reset to 0 when a F GSKSRVR,RESET STATS,TICKETCACHE command is performed. The number of resets is incremented by 1 and the reset time is set to the time that the command is ran. If the GSKSRVR is running a very long time, it is possible for these statistics to wrap. In these cases, it is recommended that you perform periodic resets so that the gathered cache statistics have more meaning.

General statistics:

1

Indicates the current time that the statistics were displayed.

2

Indicates the time that the System SSL started task, GSKSRVR, was started.

3

Indicates the time that the session ticket cache statistics were last reset. If the statistics were not reset since the GSKSRVR started, the start and reset times are the same. In the example above, session ticket cache statistics have been reset after the GSKSRVR has started.

4

Indicates the number of cache statistics resets that have occurred since the GSKSRVR has started. In this example, there has been one reset of the session ticket cache statistics.

5

The current cache size in megabytes. The cache grows in increments of 1MB up to the maximum size specified in the GSK_SESSION_TICKET_CACHE_SIZE environment variable. In this example, the current and maximum cache sizes are the same and may indicate a full cache scenario.

6

The maximum cache size in megabytes. This is the value specified in the GSK_SESSION_TICKET_CACHE_SIZE environment variable.

7

The cache timeout setting in minutes. This is the value specified in the GSK_SESSION_TICKET_CACHE_TIMEOUT environment variable. In this example, the cache timeout setting is 1 minute.

8

The maximum number of tickets that can be stored per cache entry. This is the value specified in the GSK_SESSION_TICKET_CACHE_NUM_TICKETS environment variable and rounded to the nearest multiple of 16. In this example, the number of tickets has been set to 32 per TLS V1.3 session.

Add statistics: These statistics depict how successful the local GSKSRVR instance can cache new entries. The initial cache entry add consists of caching the TLS V1.3 session information along with assigning the unique ticket IDs, which are used to construct session tickets after completing a full TLS V1.3 handshake. These statistics are only updated for adds on the local GSKSRVR instance.

9

Requested – Indicates the number of successful and failure add requests.

10

Successful – Indicates the number of successful add requests.

11

Failed full dataspace – Indicates the number of adds that failed due to the GSKSRVR dataspace being full. There is no room in the cache to add a new cache entry. The session information cannot be added. This can indicate that the maximum cache size specified in the GSK_SESSION_TICKET_CACHE_SIZE is not large enough to store the session ticket cache entries. If this number is large, it is an indication that a larger value may be needed for the GSK_SESSION_TICKET_CACHE_SIZE environment variable.

In this example, the cache full condition has probably occurred because the session ticket cache data space has been exceeded. This can be inferred from the current and maximum cache sizes in the general statistics.

12

Success percentage – This indicates a percentage of successful versus failure requests with a precision of three decimal points.

13

Replaced entries – This indicates how many older cache entries which have not yet expired were replaced in the cache during an add. In this example, 1924 older cache entries were replaced by newer cache entries. By replacing older cache entries, it allows for a newer cache entry to be successfully added to the cache. The number of replaced entries do not count directly towards the success percentage.

14

Purged entries - This indicates how many cache entries were purged or removed from the cache due to be expired or previously marked for deletion. In this example, there were no cache entries removed from the cache due to expiration.

Update statistics – These statistics depict how successful the local GSKSRVR instance can update an already existing cache entry to assign new ticket IDs for an already cached TLS V1.3 session. Session ticket cache entry updates are done when a new session ticket must be sent by the System SSL server application after completing a successful TLS V1.3 resumption handshake. These statistics are updated for local targeted requests and incoming requests from other GSKSRVR instances in the sysplex.

15

Requested – Indicates the number of successful and failure update requests.

16

Successful – Indicates the number of successful update requests.

17

Failed expired – This indicates the update did not succeed as the cache entry has already expired and it cannot be updated.

18

Failed not found – The cache entry that is attempted to be updated cannot be found in the cache.

19

Failed not authorized – The server application’s userid that is attempting to update the existing cache entry does not have the appropriate authority to the GSK.SIDCACHE.<*userid*> profile in the FACILITY class. These failures may be expected if the appropriate authority was not given to the cache entries created by other userids.

20

Success percentage - This indicates a percentage of successful versus failure requests with a precision of three decimal points.

Retrieval statistics – These statistics depict how successful cache retrievals are. Cache retrievals are performed when a System SSL server application attempts a TLS V1.3 resumption with a client presented session ticket which contains a ticket ID. The System SSL server application presents the ticket ID from the ticket to retrieve the TLS V1.3 session information. These statistics are updated for local targeted requests and incoming requests from other GSKSRVR instances in the sysplex.

21

Requested – Indicates the number of successful and failure retrieval requests.

22

Successful – Indicates the number of successful retrieval requests.

23

Failed expired – This indicates on the retrieval of a cache entry that it has already expired and it cannot be retrieved.

24

Failed not found – The cache entry that is attempted to be retrieved cannot be found in the cache. This may occur because the session cache entry has been removed from the cache.

25

Failed not authorized – The server application’s userid that is attempting to retrieve an existing cache entry does not have the appropriate authority to the GSK.SIDCACHE.<*userid*> profile in the FACILITY class. These failures may be expected if the appropriate authority was not given to the cache entries created by other userids.

26

Success percentage - This indicates a percentage of successful versus failure requests with a precision of three decimal points.

If the F GSKSRVR,DISPLAY TICKETCACHE command is ran at nearly the same time as the F GSKSRVR,DISPLAY STATS,TICKETCACHE command, it is possible to get a breakdown on the cache entries per user. The F GSKSRVR,DISPLAY TICKETCACHE command output shows the current and maximum size of the dataspace followed by lines displaying the cache users, the number of cache users, and the maximum cache entry size in bytes. This output shows what applications have allocated entries within the cache and can be used for further tuning of the cache size or timeout settings based on an organization’s security policy. The GSK_SESSION_TICKET_CACHE_SIZE controls the maximum size of the session ticket cache while the GSK_SESSION_TICKET_CACHE_NUM_TICKETS plays a role in how large the cache entries are. The more tickets that are allowed to be stored in the cache per session or cache entry the larger the cache entries are. The actual cache entry size is rounded up to the nearest multiple of 1024 bytes (1k).

```
GSK01076I Session ticket cache
 status
User              Count              Max size
*DSPACE*          13/13                   N/A
APPL1              2394                  4103
APPL2               100                  4103
```

Component trace support

For information about component trace support, see “Component trace support” on page 624.

Hardware cryptography failure notification

For information about cryptographic hardware failure notification, see Chapter 3, “Using cryptographic features with System SSL,” on page 9.

Chapter 12. Obtaining diagnostic information

All of the information and techniques described in this topic are for use primarily by IBM service personnel in determining the cause of a System SSL problem. If you encounter a problem and call the IBM Support Center, you might be asked to obtain trace information or enable one or more of the diagnostic messages described here.

Any environment variables described in this topic are usually set from the UNIX System Services **export** shell command. For usage information about this command, see the *z/OS UNIX System Services Command Reference*. For information about setting environment variables outside of the shell, see *z/OS XL C/C++ Programming Guide* and the *z/OS Language Environment Programming Guide*.

The following facilities are not intended for use in a production environment and are for diagnostic purposes only.

Obtaining System SSL trace information

You can enable the System SSL trace by using the environment variable GSK_TRACE_FILE to specify the name of the trace file, and the GSK_TRACE environment variable to set the trace level. A single trace file is created, and there is no limit on the size of the trace file.

In order to create a readable copy of the trace information, use the System SSL **gsktrace** command as follows:

```
gsktrace input_trace_file > output_trace_file
```

Capturing trace data through environment variables

In order to capture trace information using environment variables, the trace environment variables **GSK_TRACE** and **GSK_TRACE_FILE** must be exported before the start of the SSL application.

Note: The **GSK_TRACE** and **GSK_TRACE_FILE** environment variables should only be specified or exported by the SSL application when it is necessary to obtain diagnostic information. These environment variables must not be specified within a CEEPRM*xx* file as this results in all System SSL applications to be traced that are running on the system.

- **GSK_TRACE**

  Specifies a bit mask enabling System SSL trace options. No trace option is enabled if the bit mask is 0 and all trace options are enabled if the bit mask is 0xffff. The bit mask can be specified as a decimal (nnn), octal (0nnnn) or hexadecimal (0xhh) value.

  These trace options are available:

  0x01 = Trace function entry
  0x02 = Trace function exit
  0x04 = Trace errors
  0x08 = Include informational messages
  0x10 = Include EBCDIC data dumps
  0x20 = Include ASCII data dumps

- **GSK_TRACE_FILE**

  Specifies the name of the trace file and defaults to /tmp/gskssl.%.trc. The trace file is not used if the **GSK_TRACE** environment variable is not defined or is set to 0.

  The current process identifier is included as part of the trace file name when the name contains a percent sign (%). For example, if GSK_TRACE_FILE is set to /tmp/gskssl.%.trc and the current process identifier is 247, then the trace file name is /tmp/gskssl.247.trc.

Note: Care needs to be taken if the application being traced is multi-processed. If multiple processes write to the same trace file, file corruption might occur. To allow trace information to be obtained, the trace file name specified should contain a '%' character in the file name. This allows the process identifier to be placed within the file name and each process to write to its own trace file.

It is suggested that if the default trace file value is not being used, the trace file name always contain a '%' character. This eliminates the need to know whether the application being traced is multi-processed or not.

Once the trace file is produced, it must be formatted. To format the file, use the System SSL **gsktrace** command as follows:

```
gsktrace input_trace_file > output_trace_file
```

Component trace support

The System SSL started task provides component trace support for any SSL application running on the same system as the GSKSRVR started task. The trace records can be written to a trace external writer or they can be kept in an in-storage trace buffer which is part of the GSKSRVR address space. IPCS is used to format and display the trace records from either a trace data set or an SVC dump of the GSKSRVR address space. Data set *hlq*.SIEAMIGE containing the SSL trace record format routine to be used by IPCS must be accessible through either a steplib or in the lnklst.

Note: The System SSL started task provides component trace support only for SSL applications, therefore, component trace of the System SSL started task itself is not supported. Therefore, the jobname for the System SSL started task should not be used as one of the jobnames in the MVS TRACE command. Tracing for the System SSL started task can be accomplished by setting the GSK_TRACE environment variable.

The Component Trace input command supports the option JOBSUFFIX to enable wildcarding. JOBSUFFIX can be specified as ANY or NONE with NONE being the default. If you specify JOBSUFFIX=ANY, any specified jobnames of seven letters or less are considered to be a wildcard entry and tracing is started for jobs whose names match those entries for the length of the entry.

See *z/OS MVS Diagnosis: Tools and Service Aids* for more information about setting up and using component trace. See *z/OS MVS System Commands* for more information about the TRACE command. See *z/OS MVS IPCS User's Guide* for more information about using IPCS to view a component trace.

Capturing component trace data

The component trace can be started before the job to be traced is started or while the job is running. The trace is active for the first instance of the job. For example, if the same job name is used for multiple jobs, only the first job with that name is traced. Subsequent jobs with the same name are not traced unless the component trace is stopped and then restarted.

A trace external writer is required if the trace records are to be written to a data set. A sample started procedure is shipped as GSK.SGSKSAMP(GSKWTR). Copy this procedure to SYS1.PROCLIB(GSKWTR) and modify as necessary to meet your installation requirements. This MVS operator command starts the trace external writer:

```
TRACE CT,WTRSTART=GSKWTR
```

A single SSL component trace may be active at a time and the trace can include from 1 to 16 separate jobs. The trace buffer size must be between 64K and 512K and defaults to 256K.

System SSL supports these options for CTRACE:

```
OPTIONS=([LEVEL={nnn | 15}][,JOBSUFFIX={NONE | ANY}])
```

LEVEL
A bit mask specifying the types of events that System SSL is to trace. At least one of these indicators must be specified in the supplied bit mask. All trace options are enabled if the bit mask is 0xffff. The

bit mask can be specified as a decimal (nnn), octal (0nnnn) or hexadecimal (0xhh) value. The SSL trace level is set to decimal 15 if level is not specified in the CTRACE options.

These trace options are available:

- 0x01 = Trace function entry
- 0x02 = Trace function exit
- 0x04 = Trace errors
- 0x08 = Include informational messages
- 0x10 = Include EBCDIC data dumps
- 0x20 = Include ASCII data dumps

JOBSUFFIX
A switch specifying how the list of jobnames provided from the JOBNAME parameter are to be filtered:

ANY
Any specified jobnames of 7 letters or less are considered to be wildcard entries and tracing is started for jobs whose names match those entries for the length of the entry.

NONE
Only jobs whose names match precisely one of the entries supplied in the JOBNAME parameter are traced. This is the default value.

For example, to start an SSL component trace for jobs CS390IP and DB1G which includes all non-dump trace entries and writes the trace records using the GSKWTR trace writer:

```
TRACE CT,ON,COMP=GSKSRVR
R n,JOBNAME=(CS390IP,DB1G),OPTIONS=(LEVEL=15),WTR=GSKWTR,END
```

To start an SSL component trace for job CICS1 which includes all trace entries and writes the trace records using the GSKWTR trace writer:

```
TRACE CT,ON,COMP=GSKSRVR
R n,JOBNAME=(CICS1),OPTIONS=(LEVEL=255),WTR=GSKWTR,END
```

These commands stop the SSL component trace and close the trace writer data set:

```
TRACE CT,OFF,COMP=GSKSRVR
TRACE CT,WTRSTOP=GSKWTR
```

System SSL does not require a default trace member in SYS1.PARMLIB since SSL component trace is not activated until the operator enters the TRACE command. SYS1.PARMLIB members can be created for frequently used trace commands and the member name can then be specified on the TRACE command to avoid the operator prompt for trace options.

Starting and stopping the in-storage trace is done the same way as the external writer trace except the external writer name on the trace command should not be specified.

These commands start the SSL component trace using the in-storage trace table:

```
TRACE CT,ON,COMP=GSKSRVR
R n,JOBNAME=(CS390IP,DB1G),OPTIONS=(LEVEL=15,JOBSUFFIX=ANY),END
```

Before stopping the SSL component trace, an SVC dump of the GSKSRVR address space is required to access the in-storage trace table. For more information, see "Capturing component trace data without an external writer" on page 628.

This command stops the SSL component trace using the in-storage trace table:

```
TRACE CT,OFF,COMP=GSKSRVR
```

Displaying the trace data

The trace records are displayed using the IPCS CTRACE command.

The CTRACE ENTIDLIST parameter specifies the trace entries to be included in the display. The trace entry type is the same as the SSL trace level. For example, SSL function entry trace records have entry type 1, SSL function exit trace records have entry type 2, SSL error records have entry type 4, and so on. All trace entries are included if the ENTIDLIST parameter is not specified.

The CTRACE OPTIONS parameter specifies additional filtering for the trace records. The JOB(name), PID(hexid), and TID(hexid) options can be specified to filter the trace entries based on job name, process identifier, or thread identifier. All trace entries are included if the OPTIONS parameter is not specified.

Note that the JOBNAME parameter on the CTRACE command is used to select the address space in a dump. Since the address space is always the GSKSRVR address space, this parameter cannot be used to filter the trace entries. Instead, you must use the OPTIONS((JOB(name))) parameter to select the component trace entries for a specific job.

For example, to display SSL function entry and SSL function exit trace records for job KRBSRV48 thread 6:

```
IPCS CTRACE COMP(GSKSRVR) ENTIDLIST(1,2) OPTIONS((JOB(KRBSRV48),TID(6))) FULL
```

A range can be specified for the entry identifiers. For example, to display just the non-dump trace records:

```
IPCS CTRACE COMP(GSKSRVR) ENTIDLIST(1:15) FULL
```

Event trace records for System SSL

The FULL format of a component trace report is as follows:

```
     COMPONENT TRACE FULL FORMAT
     SYSNAME(C01)
     COMP(GSKSRVR)
     **** 11/14/2005

     SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
     -------   --------  --------  ---------------  -------------

   1 C01       MESSAGE   00000004  20:43:45.522449  SSL_ERROR

   2   Job TCP341    Process 00020032  Thread 00000002  gsk_read_v3_record
   3   Socket closed by 192.168.50.80.1360.
```

1. Standard IPCS header line, which includes the system name (C01), System SSL trace entry format (MESSAGE or DUMP), entry ID, time stamp, and record description.
2. System SSL header line with job name, process id, thread id, and function name information.
3. System SSL detail information. The format of this area's content is determined according to the System SSL record description located on line 1. Trace records may have 0 or more detail lines.

The standard IPCS header line MNEMONIC, ENTRY ID, and DESCRIPTION combinations are as follows:

```
MNEMONIC ENTRY ID   DESCRIPTION         EXPLANATION
MESSAGE  00000001   SSL_ENTRY           Entry into the function named in the following System SSL
                                        header line (i.e. line 2) occurred
MESSAGE  00000002   SSL_EXIT            Exit from the function named in the following System SSL
                                        header line occurred
MESSAGE  00000004   SSL_ERROR           Error was detected by the function named in following line 2
                                        with error description in line 3
MESSAGE  00000008   SSL_INFO            Information generated by the function named in following line 2
                                        - for example, supplied parameters
DUMP     00000010   SSL_EBCDIC_DUMP     Dump of buffer contents formatted in EBCDIC, by the function
                                        named in following line 2
DUMP     00000020   SSL_ASCII_DUMP      Dump of buffer contents formatted in ASCII, by the function name
                                        in following line 2
```

The System SSL header line contains the Job name, Process ID (in hex), Thread ID (in hex), and the name of the System SSL function that created the trace entry. If the trace entry is output while in SRB mode, then the Thread ID is FFFFFFFF.

The format of the System SSL detail line is similar for record descriptions SSL_ENTRY, SSL_EXIT, SSL_ERROR and SSL_INFO.

```
     1  C01        MESSAGE    00000001   20:43:46.694762   SSL_ENTRY

     2    Job TCP341     Process 00020032    Thread 00000004   gsk_secure_socket_read
     3    Handle 7E828198,  Size 1

     4  C01        MESSAGE    00000008  20:43:46.695013    SSL_INFO

     5    Job TCP341     Process 00020032    Thread 00000004   gsk_read_v3_record
     6    Calling read routine for 5 bytes

     7  C01        MESSAGE    00000004  20:43:46.695317    SSL_ERROR

     8    Job TCP341     Process 00020032    Thread 00000004   gsk_read_v3_record
     9    Socket closed by 192.168.50.80.1472.

    10  C01        MESSAGE    00000004  20:43:46:695478    SSL_ERROR

    11    Job TCP341     Process 00020032    Thread 00000004   gsk_secure_socket_read
    12    SSL V3 data read failed with 192.168.50.80.1472.

    13  C01        MESSAGE    00000002  20:43:46.695599    SSL_EXIT

    14    Job TCP341     Process 00020032    Thread 00000004   gsk_secure_socket_read
    15    Exit status 000001A4 (420)
    16    Length 0
```

1. The start of a trace record reporting that a function is entered. Not all functions create a trace record. If a function creates an SSL_ENTRY record, then it also creates a corresponding SSL_EXIT record.

2. The System SSL header record describing the job, process, thread, and function creating the record.

3. The detail for the SSL_ENTRY record. Not all trace records create a detail line. Trace records may have multiple detail lines.

4. The start of a trace record for function gsk_read_v3_record. The fact that an SSL_EXIT record is not encountered for function gsk_secure_socket_read (the previous trace record), indicates that gsk_read_v3_record is invoked either by gsk_secure_socket_read or another function invoked by gsk_secure_socket_read.

7. The start of an error trace record created by gsk_read_v3_record.

10. An error trace record created by gsk_secure_socket_read. The error occurred because of the error detected in gsk_read_v3_record.

13. The start of the trace record created by gsk_secure_socket_read on exit. It corresponds with the trace entry record on Line 1.

15. The first detail line and reports the return code returned by gsk_secure_socket_read.

16. The second detail line for the trace record. It is an example of a trace record with multiple detail lines.

The format of the System SSL detail for record descriptions SSL_EBCDIC_DUMP and SSL_ASCII_DUMP is as follows:

```
 1 C01       DUMP      00000020  20:43:45.724056  SSL_ASCII_DUMP

 2   Job TCP341    Process 00020032  Thread 00000004  send_v3_server_messages
 3   SERVER-HELLO message
 4     00000000: 02000046 03014373 B1011649 3E508E04   *...F..Cs...I>P..*
       00000010: A620B42C 8422C287 8015BC54 7850C435   *. .,."....TxP.5*
       00000020: 540B6864 A4702000 020032C0 A8325005   *T.hd.p ...2..2P.*
       00000030: E9000000 00000000 00000000 00000043   *...............C*
       00000040: 73B10100 00051A00 0500                *s.........      *
```

1. Standard IPCS header line.

2. System SSL header line.
3. The first line of the System SSL detail area. It describes the contents that are dumped in the detail lines. In this example, the SERVER_HELLO message sent to the client is output in the detail lines.
4. The first line of the contents dump. Each dump line consists of offset, 16 bytes of data in hex, and the same 16 bytes of data output in either ASCII or EBCDIC enclosed in asterisks.

Capturing component trace data without an external writer

If there is not an external writer, you can dump the GSKSRVR address space.

To use a dump:

- Dump the GSKSRVR address space with the command:

  ```
  DUMP COMM=(title of dump)
  ```

- Reply with:

  ```
  R x,JOBNAME=(GSKSRVR),SDATA=(RGN,LSQA,ALLNUC,PSA,TRT,CSA,SQA),END
  ```

- Issue

  ```
  TRACE CT,OFF,COMP=GSKSRVR
  ```

  to turn off the trace.

Note: You need to take the dump before you turn off the CTRACE.

Chapter 13. Messages and codes

This topic lists the messages and codes issued by System SSL:

- SSL function return codes ("SSL function return codes" on page 629)
- Deprecated SSL function return codes ("Deprecated SSL function return codes" on page 673)
- ASN.1 status codes (014CExxx) ("ASN.1 status codes (014CExxx)" on page 694)
- CMS status codes (03353xxx) ("CMS status codes (03353xxx)" on page 700)
- SSL started task messages (GSK01nnn) ("SSL started task messages (GSK01nnn)" on page 741)
- Utility messages (GSK00nnn) ("Utility messages (GSK00nnn)" on page 761)

For information about the following stderr messages, see "gsk_strerror()" on page 181:

- Message *xxx* not found in catalog gskmsgs.cat.
- Message identifier *xxx* is not defined.
- Unable to open message catalog gskmsgs.cat.

SSL function return codes

System SSL functions return the value 0 (GSK_OK) if no error is detected. Otherwise, one of the return codes listed in the **gskssl.h** include file is returned.

1 **Handle is not valid.**

Explanation

The environment or SSL handle specified on a System SSL function call is not valid.

User response

Call the **gsk_environment_open()** function to create an environment handle or the **gsk_secure_socket_open()** function to create an SSL handle.

3 **An internal error has occurred.**

Explanation

The System SSL runtime library detected an internal processing error.

User response

Retry the operation. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

4 **Insufficient storage is available**

Explanation

The System SSL runtime library is unable to obtain storage for an internal control block.

User response

Increase the storage available to the application and then retry the failing operation.

5 **Handle is in the incorrect state.**

Explanation

The SSL handle is in the incorrect state for the requested operation.

User response

Correct the application to request SSL functions in the proper sequence.

6 **Key label is not found.**

Explanation

The requested key label is not found in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. When using a PKCS #12 file, this error can also occur when the file is being processed during the establishment of the SSL/TLS environment when a certificate is encountered where there is no friendly name PKCS #12 attribute and the certificate's subject distinguished name is empty.

User response

Specify a label that exists in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. If encountered when establishing an SSL/TLS environment using a PKCS #12 file, verify any certificate that has no subject distinguished name is assigned a PKCS #12 friendly name attribute.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate’s trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate’s profile is generic. A (G) indicates generic.

7 **No certificates available.**

Explanation

The key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token does not contain any certificates, or the SSL client application does not have a certificate available when authentication is requested by the server.

User response

Check for available certificates and add the user certificate and any necessary certification authority certificates to the key database, SAF key ring, or z/OS PKCS #11 token if necessary.

If using a PKCS #12 file, ensure that the file contains the necessary certificates.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate’s trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available. Specify a certificate for the client application to use.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class.

The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate’s profile is generic. A (G) indicates generic.

8 **Certificate validation error.**

Explanation

An error is detected while validating a certificate. This error can occur for the following reasons:

- The root CA certificate is not found in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token.
- The certificate is not marked as a trusted certificate.
- The certificate requires an algorithm or key size that is non-FIPS while executing in FIPS mode.
- The certificate does not validate properly according to the specifications of RFCs 2459, 3280, 5280, or 5759. The validation mode is determined by the GSK_CERT_VALIDATION_MODE setting. RFC 5759 certificate validation is only done when GSK_SUITE_B_PROFILE is set to 128MIN or 192MIN which overrides the GSK_CERT_VALIDATION_MODE setting.
- If TLS V1.3 is negotiated for a secure connection, certificate validation is done according to RFC 5280 unless the GSK_CERT_VALIDATION_MODE setting is explicitly specified.

User response

The following must be verified depending upon the error that is encountered:

- Verify that the root CA certificate is in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS#11 token and is marked as trusted.
- Check all certificates in the certification chain and verify that they are trusted and are not expired.
- If executing in FIPS mode, check that only FIPS algorithms and key sizes are used by the certificate. For more information, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.
- If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate’s trust chain are available.
- If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.
- If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate’s profile is generic. A (G) indicates generic.
- Verify the certificates in the peer certificate chain adhere to the RFC specifications of the certificate validation mode. The RFCs indicate the required and optional characteristics of the certificates. If GSK_SUITE_B_PROFILE is set to 128MIN or 192MIN, see “Suite B cryptography support” on page 49 for more information.

Collect a System SSL trace that contains the error and then contact your service representative if the problem persists.

9 **Cryptographic processing error.**

Explanation

An error is detected by a cryptographic function. This error might also occur while running in FIPS mode when negotiating a secure connection in the following cases:

- Using a non-FIPS key size.
- Using a non-FIPS elliptic curve.

- Using a triple DES cipher and the negotiated triple DES session key does not have three unique key parts.

User response

If the error occurred while executing in FIPS mode, check that only FIPS key sizes or elliptic curves are used. If the error occurred during the establishment of a secure connection in FIPS mode using a triple DES cipher, retry the connection. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

For more information about FIPS key sizes, see Chapter 4, “System SSL and FIPS 140-2,” on page 19 and see Appendix C, “Cipher suite definitions,” on page 795 for information about supported ciphers.

10 **ASN processing error.**

Explanation

An error is detected while processing a certificate field. This error can also occur when a TLS client or server received a message containing a TLS extension that was not correctly formed. The TLS extension data may contain a length field that has an incorrect value.

User response

If using TLS extensions, ensure that the TLS extension data is correct for both the TLS server and client. If the error persists, collect a System SSL trace containing the error and then contact your service representative.

11 **LDAP processing error.**

Explanation

An error is detected while setting up the LDAP environment or retrieving an LDAP directory entry.

User response

Ensure that the LDAP server is running and that there are no network errors. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

12 **An unexpected error has occurred.**

Explanation

An unexpected error is detected by the System SSL run time.

User response

Collect a System SSL trace containing the error and then contact your service representative.

13 **Size specified for supplied structure is too small**

Explanation

The value of the size field in the structure indicates that the size of the structure is insufficient.

User response

Ensure that the size field in the structure that is being used is initialized to the size of structure.

14 **Required gsk_all_cipher_suites structure not supplied**

Explanation

A gsk_all_cipher_suites structure required by the API was not supplied on the function call.

User response

Ensure that all parameters required by the API are specified on the function call

102 **Error detected while reading certificate database**

Explanation

An error is detected while reading the key database, the PKCS #12 file, or retrieving entries from the SAF key ring or z/OS PKCS #11 token.

User response

If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

103 **Incorrect key database record format.**

Explanation

The record format for a key database entry is not correct. This error can occur if the name of a request database is provided instead of the name of a key database.

User response

Ensure that the correct database name is used. Collect a System SSL trace containing a dump of the keyfile entry and then contact your service representative if the error persists.

106 **Incorrect key database password.**

Explanation

The System SSL run time is unable to decrypt a key database entry. Either the supplied database password is incorrect or the database is damaged.

User response

Ensure that the correct key database password is used. Re-create the database if the error persists.

109 **No certification authority certificates.**

Explanation

The key database, SAF key ring, or z/OS PKCS #11 token does not contain any valid certification authority certificates. The SSL run time needs at least one CA or self-signed certificate to perform client authentication.

User response

Add the necessary certificates to the key database, SAF key ring, or z/OS PKCS #11 token and ensure that existing certificates are valid, have not expired, and are marked as trusted certificates.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate's trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate's profile is generic. A (G) indicates generic.

201 **No key database password supplied.**

Explanation

A password stash file is specified but the SSL run time is unable to read the password from the stash file.

User response

Verify that the password stash file exists and both the file and directory path are accessible to the application. Re-create the password stash file if the error persists.

202 **Error detected while opening the certificate database.**

Explanation

An error is detected while opening the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. This error can occur if no name is supplied or the database, PKCS #12 file, key ring, or token does not exist. When using a PKCS #12 file, the file name cannot end with .kdb, .rdb or .sth.

User response

Verify that the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token exists and is accessible by the application. This value is case-sensitive. Ensure that the case is preserved with your request. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

203 **Unable to generate temporary key pair**

Explanation

An error is detected while generating a temporary key pair.

User response

Collect a System SSL trace containing the error and then contact your service representative.

204 **Key database password is expired.**

Explanation

The key database password is expired.

User response

Use the **gskkyman** utility to assign a new password for the key database.

302 **Connection is active.**

Explanation

An SSL secure connection operation cannot be completed because of an active request for the connection.

User response

Retry the failing request when the currently active request completed.

401 **Certificate is expired or is not valid yet.**

Explanation

The current time is either before the certificate start time or after the certificate end time.

User response

Obtain a new certificate if the certificate is expired or wait until the certificate becomes valid if it is not valid yet.

402 **No SSL cipher specifications.**

Explanation

This error can occur if:

- The client and server cipher specifications do not contain at least one value in common. Client and server cipher specifications might be limited depending on which System SSL FMIDs are installed. See Appendix C, "Cipher suite definitions," on page 795 for more information. Server cipher specifications are dependent on the type of algorithms that are used by the server certificate (RSA, DSA, ECDSA, or Diffie-Hellman), which might limit the options available during cipher negotiation.
- No SSL protocols are enabled or if all of the enabled protocols have empty cipher specifications or if the TLS protocol is not enabled while executing in FIPS mode.
- A client supporting only SSL V2 has specified session ID cache reuse by specifying GSK_ENABLE_CLIENT_SET_PEERID set to ON or GSK_REQ_CACHED_SESSION set to ON. Session ID cache reuse is not supported with protocol SSL V2.
- A server supporting only SSL V2 has specified session ID cache reuse through the GSK_SID_VALUE attribute.
- Session ID cache reuse is not supported with protocol SSL V2.
- Attempting to use a certificate with its ECC private key in the ICSF PKDS and only fixed ECDH ciphers are specified.
- Using the TLS V1.1 or higher protocol and only the 40-bit export ciphers are specified.
- Using TLS V1.2 and only 56-bit DES ciphers are specified.
- Using TLS V1.2 and none of the server cipher specifications use key algorithms that are listed in the signature algorithms pairs sent by the client.
- An attempt was made to use a certificate with its DH secure private key in the ICSF PKDS. Only clear private keys are supported.
- An attempt was made to use a certificate with its ECC secure private key in the ICSF PKDS. Only clear private keys are supported.
- Using Suite B mode and no required Suite B ciphers were specified.
- Using protocol TLS V1.0, TLS V1.1, or TLS V1.2, any specified ephemeral elliptic curve (ECDHE) cipher suite is ignored if the client and server do not have a supported elliptic curve in common.
- The server has selected TLS V1.3, but there are no TLS V1.3 cipher specifications specified.
- The remote partner indicated a handshake failure which is a generic error that is encountered during the handshake. This can occur for various reasons such as (but not limited to) the following:
  - The remote partner required the extended master secret extension, but it was not specified on this connection.
  - The remote server partner's certificate chain contains a certificate using a signature algorithm pair that was not specified in either the GSK_TLS_CERT_SIG_ALG_PAIRS or GSK_TLS_SIG_ALG_PAIRS set by the client when attempting a TLS V1.2 or TLS V1.3 handshake.

User response

Ensure that the client and the server have at least one cipher specification and protocol in common. Verify that specified session identifier is correct, not expired, that the cache is large enough to hold the cached session entry, and that maximum connection has not been reached.

If an ephemeral elliptic curve (ECDHE) cipher is expected to be used, ensure that the client and server have a supported elliptic curve or group specification in common. The client provides its list of supported elliptic curves as part of the TLS handshake. If the server does not support at least one elliptic curve supported by the client, TLS_ECDHE cipher suites cannot be used. TLS_ECDHE_ECDSA cipher suites cannot be used if the server's certificate does not match one of the client's supported elliptic curves. A z/OS System SSL client specifies the list of supported elliptic curves through the GSK_CLIENT_ECURVE_LIST setting. An empty client list means all elliptic curves supported can be used. The FIPS setting may further restrict what elliptic curves can be used. A z/OS System SSL server specifies the list of supported elliptic curves through the GSK_SERVER_ALLOWED_KEX_ECURVES setting. See Table 29 on page 804 for supported curves or groups.

Ensure that the GSK_TLS_CERT_SIG_ALG_PAIRS or GSK_TLS_SIG_ALG_PAIRS setting is defined to include all of the signature algorithm pairs that are both supported by the local application and expected from the remote peer.

403 **No certificate received from partner.**

Explanation

The required certificate was not received from the communication partner.

User response

Ensure that the remote application is sending the certificate. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

405 **Certificate format is not supported.**

Explanation

The certificate received from the communication partner is not supported during the negotiated SSL or TLS protocol handshake.

User response

If an ECC certificate is received from the communication partner during a TLS V1.0, TLS V1.1, or TLS V1.2 handshake, the certificate's elliptic curve must be specified in the GSK_CLIENT_ECURVE_LIST environment variable or attribute type. System SSL does not support x25519 and x448 certificates. If the communication partner's certificate is either x25519 or x448, a different certificate must be provided.

If a TLS V1.3 handshake is attempted, DSA, Diffie-Hellman, and elliptic curve x25519 or x448 certificates are not allowed to be received from the communication partner. There are limitations on the key type and sizes that are allowed for TLS V1.3. See "gsk_secure_socket_init()" on page 157's usage section for more information. The communication partner will need to provide a different certificate.

If the problem persists, collect a System SSL trace that contains a dump with the unsupported certificate and then contact your service representative.

406 **Error while reading or writing data.**

Explanation

An I/O error was reported while the System SSL run time was reading or writing data.

User response

Ensure that there are no network errors. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

407 **Key label does not exist.**

Explanation

The supplied label or the default key is not found in the key database or the certificate is not trusted or the certificate uses algorithms or key sizes that are not supported while executing in FIPS mode. If using a PKCS #12 file as the certificate database, the label is either the certificate's friendly name or the subject's distinguished name.

User response

Supply a valid label or define a default key in the key database or specify a label for a certificate that uses FIPS algorithms or key sizes if executing in FIPS mode. If using a PKCS #12 file, use the **gskkyman** command line option **-dc** or **-dcv** to display the contents of the PKCS #12 file. The friendly name or subject distinguished name values is displayed in the label field.

For more information about FIPS, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate’s trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate’s profile is generic. A (G) indicates generic.

408 **Key database password is not correct.**

Explanation

The System SSL run time is unable to decrypt a keyfile entry. Either the supplied keyfile password is incorrect or the keyfile is damaged.

User response

Ensure that the correct keyfile password is used. Re-create the keyfile if the error persists.

410 **SSL message format is incorrect.**

Explanation

An incorrectly formatted SSL message is received from the communication partner.

User response

Collect a System SSL trace containing a dump of the SSL message and then contact your service representative.

411 **Message authentication code is incorrect.**

Explanation

The message authentication code (MAC) for a message is not correct. This indicates that the message was modified during transmission.

User response

Collect a System SSL trace containing a dump of the message and then contact your service representative if the error persists.

412 **SSL protocol or certificate type is not supported.**

Explanation

The SSL handshake is not successful because of an unsupported protocol or certificate type. This error can occur if there is no enabled SSL protocol shared by both the client and the server. When executing in FIPS mode, specifying the SSL V2 or SSL V3 protocol is ignored. When enabled for TLS V1.3, SSL V2 and SSL V3 are not supported and are ignored.

User response

Ensure that the SSL protocol you want is enabled on both the client and the server. Collect a System SSL trace containing a dump of the failing handshake and then contact your service representative if the problem persists.

413 **Certificate signature is incorrect.**

Explanation

The certificate signature is not correct for a certificate received from the communication partner.

User response

Ensure that a valid certificate is being sent by the communication partner. Collect a System SSL trace containing a dump of the incorrect certificate and then contact your service representative if the error persists.

414 **Certificate is not valid.**

Explanation

Either the local certificate or the certificate received from the peer is not valid.

User response

Ensure that a valid certificate is being sent by the communication partner. If a TLS V1.3 handshake is being attempted, the communication partner may have sent a DSA certificate, an RSA certificate with signature algorithms MD2, MD5, or SHA-224, a Brainpool certificate, or an ECC certificate with signature algorithms SHA-1 or SHA-224. These certificates are not supported with TLS V1.3. Collect a System SSL trace containing a dump of the incorrect certificate and then contact your service representative if the error persists.

415 **SSL protocol violation.**

Explanation

The communication partner violated the SSL protocol by sending a message out of sequence or by omitting a required field from a message.

User response

Collect a System SSL trace and then contact your service representative.

416 **Permission denied.**

Explanation

The System SSL run time is unable to access a file or system facility.

User response

Ensure that the application is authorized to access the file or facility.

For key database files, ensure that the application user ID has read access to the file.

For SAF key rings, ensure that the application user ID has appropriate permission to the SAF resource in either the FACILITY or RDATALIB class.

- The user ID must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by the user ID.
- The user ID must have UPDATE access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by another user ID, or
- The user ID must have READ access to the *ringOwner.ringname.*LST resource in the RDATALIB class.

For z/OS PKCS #11 tokens, ensure that the application user ID has READ access to resource USER.tokenname in the CRYPTOZ class.

If the error persists, collect a System SSL trace and then contact your service representative.

417 **Self-signed certificate cannot be validated.**

Explanation

A self-signed certificate cannot be validated because it is not in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token.

User response

Add the self-signed certificate to the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate’s trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate’s profile is generic. A (G) indicates generic.

420 **Socket closed by remote partner.**

Explanation

The remote partner closed the socket. This error is also reported if the remote partner sent a close notification alert.

User response

None.

421 **SSL V2 cipher is not valid.**

Explanation

The SSL V2 cipher is not valid.

User response

Specify a valid cipher. See Table 25 on page 795 for more information about the supported SSL V2 cipher suite definitions.

422 **SSL V3 cipher is not valid.**

Explanation

The SSL V3 cipher is not valid.

User response

Specify a valid cipher. See Table 26 on page 795 for more information about the supported SSL V3 cipher suite definitions.

427 **LDAP is not available.**

Explanation

The System SSL run time is unable to access the LDAP server.

User response

Ensure that the LDAP server is running and that there are no network problems. Collect a System SSL trace and then contact your service representative if the error persists.

428 **Key entry does not contain a private key.**

Explanation

The key entry does not contain a private key or the private key is not usable. This error can also occur if the private key is stored in ICSF and ICSF services are not available, if using a SAF key ring that is owned by another user, if the private key size is greater than the supported configuration limit or the application is executing in FIPS mode. Certificates that are meant to represent a server or client must be connected to a SAF key ring with a USAGE value of PERSONAL and either be owned by the user ID of the application or be SITE certificates. This error can occur when using z/OS PKCS #11 tokens if the user ID of the application does not have appropriate access to the CRYPTOZ class. This error can occur when using private keys associated with user certificates in a SAF key ring that is owned by another user if the user ID of the application does not have appropriate access to the *ringOwner*.*ringName*.LST resource in the RDATALIB class.

User response

Ensure that the ICSF started task is started before the application if the private key is stored in ICSF. When using z/OS PKCS #11 tokens, ensure that the user ID has appropriate access to the CRYPTOZ class.

If executing in FIPS mode, ensure that the certificate that is being used does not have its private key stored in ICSF.

429 **SSL V2 header is not valid.**

Explanation

The received message does not start with a valid SSL V2 header. This error can occur if an SSL V3 client attempts to establish a secure connection with an SSL V2 server.

User response

Enable the SSL V2 protocol on the client and then retry the request.

431 **Certificate is revoked.**

Explanation

The certificate is revoked by the certification authority.

User response

Obtain a new certificate.

432 Session renegotiation is not allowed.

Explanation

An attempt to renegotiate the session parameters for an active connection is rejected. This code occurs if renegotiation is disabled, or if the client or server rejects the renegotiation. If using the TLS protocol, and a no renegotiation alert is sent to the peer or received from the peer, then SSL processing continues using the current session parameters. If using the TLS or the SSL V3 protocol, and a handshake failure alert is sent to the peer or received from the peer, then the SSL connection is closed.

User response

If the session parameters are expected to be successfully reset, then the connection must be closed.

433 Key exceeds allowable export size.

Explanation

The key size that is used for an export cipher suite exceeds the allowable maximum size. For RSA and DSA keys, the maximum export key size is 512 bits. If the certificate key is larger than 512 bits, the SSL run time uses a temporary 512-bit key for the connection.

User response

Collect a System SSL trace and then contact your service representative.

434 Certificate key is not compatible with cipher suite.

Explanation

The certificate key is not compatible with the negotiated cipher suite. The negotiated cipher suite is dependent on the type of algorithms used by the server certificate (RSA, DSA, or Diffie-Hellman) and those available for the client to use. This error can also occur if the client certificate uses an algorithm that is incompatible with the server certificate.

User response

Specify a certificate with the appropriate key type.

435 Certification authority is unknown.

Explanation

The key database does not contain a certificate for the certification authority.

User response

Obtain the certificate for the certification authority and add it to the key database. When using a SAF key ring, the CA certificate must be TRUSTed.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate's trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate’s profile is generic. A (G) indicates generic.

436 **Certificate revocation list cannot be found.**

Explanation

The required certificate revocation list (CRL) cannot be found in the specified LDAP server when the *gsk_crl_security_level* is set to MEDIUM or HIGH or the CRL cannot be found in the HTTP server indicated in the CRL distribution points extension and the GSK_REVOCATION_SECURITY_LEVEL is set to MEDIUM or HIGH.

User response

If contacting an LDAP server to retrieve the CRL, verify that the CRL is present in the LDAP entry being searched and is valid. Verify that the certificate's issuer is the same as the CRL issuer. Contact the certification authority and obtain the required CRL.

If contacting an HTTP server to retrieve the CRL, verify that the CRL is present on the HTTP server. Contact the HTTP server administrator to verify that the CRL is present on the server. If there are crlIssuers present in the CRL distribution point extension, verify that there is at least one match between those and the CRL issuer. If a match cannot be found in the crlIssuers in the CRL distribution point extension or there are no crlIssuers present, verify that the certificate's issuer is the same as the CRL issuer. The HTTP server administrator may need to contact the certification authority to obtain the required CRL.

Collect a System SSL trace containing the error and then contact your service representative if the problem persists.

437 **Connection closed.**

Explanation

For **gsk_secure_socket_read()**, a close notification is received from the peer application. For **gsk_secure_socket_write()**, a close notification is sent to the peer application. A close notification is sent when the **gsk_secure_socket_shutdown()** routine is called or when a close notification is received from the peer application. Additional data may not be sent by the application after the close notification is sent to the peer application.

User response

None

438 **Internal error reported by remote partner.**

Explanation

The peer application detected an internal error while performing an SSL operation and sent an alert to close the secure connection.

User response

Check the error log for the remote application to determine the nature of the processing error.

439 **Unknown alert received from remote partner.**

Explanation

The peer application sent an alert message that is not recognized by the System SSL run time.

User response

Collect a System SSL trace and then contact your service representative.

440 **Incorrect key usage.**

Explanation

The key usage certificate extension does not permit the requested key operation. This error can occur if the key usage extension of a client or server certificate (if any) does not allow the appropriate key usage.

- RSA server certificates using 40-bit export ciphers with a public key size greater than 512 bits must allow digital signature.
- Diffie-Hellman server certificates using fixed Diffie-Hellman key exchange must allow key agreement.
- Other RSA server certificates must allow key encipherment.
- DSA server certificates using ephemeral Diffie-Hellman key exchange must allow digital signature.
- Client certificates using fixed Diffie-Hellman key exchange must allow key agreement.
- ECC client and server certificates using fixed EC Diffie-Hellman (ECDH) key exchange must allow key agreement.
- Otherwise, client certificates must allow digital signature.

User response

Specify a certificate with the appropriate key usage.

If the **gskkyman** utility was used to create either the client (user) or server end-entity certificate, ensure that the appropriate option was selected from the Certificate Usage menu to create a client (user) or server certificate. The Certificate Usage menu consists of options for creating certificate authority and client (user) / server end-entity certificates.

441 **Client certificate not received during TLS handshake**

Explanation

The server requires the client to send its certificate during the TLS handshake, but a certificate was not received.

User response

Verify that the client has access to a valid certificate that can be sent to the server.

442 **Multiple certificates exist for label.**

Explanation

Access of certificate/key from label could not be resolved because multiple certificates/keys exist with the label.

User response

Correct certificate/key store so that label specifies a unique record.

If using a PKCS #12 file, use the **gskkyman** command line option **-dc** or **-dcv** to display the contents of the PKCS #12 file. The friendly name or subject distinguished name value is displayed as the label. Ensure the specified label is unique in the PKCS #12 file.

443 **Multiple keys are marked as the default.**

Explanation

Access of key from default status could not be resolved because multiple keys are marked as the default key.

User response

Correct the certificate/key store so that only one key is marked as the default key.

444 **Error encountered generating random bytes.**

Explanation

The SSL/TLS handshake encountered an error while generating random bytes.

User response

Retry the secure connection. Contact your service representative if the error persists.

445 **Key database is not a FIPS mode database.**

Explanation

While executing in FIPS mode, an attempt was made to open a key database that does not meet FIPS criteria.

User response

Specify a key database that meets FIPS criteria if running in FIPS mode.

446 **TLS extension mismatch has been encountered.**

Explanation

The TLS client received a message from the TLS server containing a TLS extension that was not requested. The TLS server must only respond to an extension that was sent by the TLS client.

User response

Ensure that the TLS server is operating correctly. If the problem persists, collect a System SSL trace and contact your service representative.

447 **Required TLS extension has been rejected.**

Explanation

The TLS server or client encountered a communicating partner that does not support a TLS extension that is defined as required.

User response

Ensure that the TLS extension data is correctly defined, and that both the TLS server and client support the required extension. If the problem persists collect a System SSL trace and contact your service representative.

448 **Requested server name is not recognized.**

Explanation

The TLS server is unable to match the server names that are supplied in a "Server Name Indication" type TLS extension, and either the TLS server or TLS client determined this scenario to be fatal.

User response

Ensure that the TLS extension data is correct for both the TLS server and client.

---

449 **Unsupported fragment length was received.**

Explanation

The TLS server received a Maximum Fragment Length TLS extension request from the TLS client that specifies an unsupported maximum fragment length. Supported maximum fragment lengths are 512 bytes, 1024 bytes, 2048 bytes, and 4096 bytes.

User response

Ensure that the TLS extension data is correct for the TLS server and the communicating partner. If the problem persists collect a System SSL trace and contact your service representative.

---

450 **TLS extension length field is not valid.**

Explanation

The TLS client or server received a message containing a TLS extension that was not correctly formed. The TLS extension data contains a length field that has an incorrect value.

User response

Ensure that the TLS extension data is correct for both the TLS server and client. If the problem persists collect a System SSL trace and contact your service representative.

---

451 **Elliptic Curve is not supported.**

Explanation

The EC domain parameters that are defined for the elliptic curve public or private key are not supported.

User response

Ensure the elliptic curve public/private key pair uses a supported elliptic curve. See Chapter 3, “Using cryptographic features with System SSL,” on page 9 for the list of elliptic curves that are supported by System SSL.

---

452 **EC Parameters not supplied**

Explanation

A gsk_buffer structure containing the EC domain parameters was not supplied on the call.

User response

Supply a gsk_buffer structure containing the EC domain parameters on the function call.

---

453 **Signature not supplied**

Explanation

A gsk_buffer structure containing the signature was not supplied on the call.

User response

Supply a gsk_buffer structure containing the signature on the function call.

454 **Elliptic Curve parameters are not valid**

Explanation

The EC domain parameters that are defined for the elliptic curve public or private key are not valid. Either no parameters could be found or the parameters could not be successfully decoded.

User response

Ensure the elliptic curve public/private key pair uses a valid elliptic curve.

455 **ICSF services are not available**

Explanation

A cryptographic process cannot be completed because of ICSF callable services being unavailable. This error might also occur when attempting to use a cipher suite that uses ICSF to perform a United States only encryption algorithm (such as AES-GCM) when ICSF is only able to use US export restricted encryption algorithms.

User response

Ensure that ICSF is running and operating correctly. If ICSF is running correctly, ensure that ICSF is able to use United States only encryption algorithms.

456 **ICSF callable service returned an error**

Explanation

An ICSF callable service that is employed to facilitate a cryptographic process returned an error condition. This error can occur if the user ID of the application does not have appropriate access to the RACF CSFSERV class resource profiles.

User response

Ensure that ICSF is operating correctly and that the user ID of the application has appropriate access to the RACF CSFSERV class resource profiles. See Table 5 on page 16 or Table 6 on page 16 for information about resource profiles. Collect a System SSL trace and verify the ICSF return code and reason code relating to the error. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about ICSF return and reason codes. If the problem persists contact your service representative.

457 **ICSF PKCS #11 not operating in FIPS mode**

Explanation

While running in FIPS mode, an attempt was made to use ICSF PKCS #11 services, which were not operating in FIPS mode.

User response

Ensure that ICSF is configured to run in FIPS mode.

458 **The SSL V3 expanded cipher is not valid**

Explanation

The SSL V3 4-character cipher is not valid.

User response

Specify a valid 4-character cipher. See Table 27 on page 799 for more information about supported 4-character ciphers.

459 **Elliptic Curve is not supported in FIPS mode.**

Explanation

The EC domain parameters that are defined for the elliptic curve public or private key are not approved in FIPS mode.

User response

Ensure the elliptic curve for the public or private key is valid in FIPS mode. See Chapter 4, "System SSL and FIPS 140-2," on page 19 for a list of elliptic curves that are supported by System SSL when running in FIPS mode.

460 **Required TLS Renegotiation Indication not received**

Explanation

TLS Renegotiation Indication was not received on the initial handshake with peer as required by the GSK_EXTENDED_RENEGOTIATION_INDICATOR environment variable or the gsk_attribute_set_enum enumeration ID GSK_EXTENDED_RENEGOTIATION_INDICATOR. If a server receives this code, then the GSK_EXTENDED_RENEGOTIATION_INDICATOR is set to either SERVER or BOTH and the client did not signal TLS Renegotiation Indication on the initial client hello. If a client receives this code, then the GSK_EXTENDED_RENEGOTIATION_INDICATOR is set to either CLIENT or BOTH and the server did not signal TLS Renegotiation Indication on the initial server hello.

User response

Ensure that the peer is configured to signal TLS Renegotiation Indication. If the peer does not support TLS Renegotiation Indication, and connection is required, then adjust the local setting of the environment variable GSK_EXTENDED_RENEGOTIATION_INDICATOR to "OPTIONAL" or the gsk_attribute_set_enum enumeration ID GSK_EXTENDED_RENEGOTIATION_INDICATOR to GSK_EXTENDED_RENEGOTIATION_INDICATOR_ OPTIONAL.

461 **EC domain parameter format is not supported.**

Explanation

The server key exchange message contains an elliptic curve parameter format or named curve specification that is not supported

User response

For ephemeral ECDH cipher suites, ensure that only the named curve EC domain parameter format is used in the server key exchange message, with a named curve that is supported by System SSL.

462 **Elliptic Curve point format is not supported.**

Explanation

The elliptic curve public value is specified using an EC point format that is not supported.

User response

Ensure the elliptic curve public value is specified using a supported EC point format. System SSL supports only the uncompressed EC points format.

463 **Cryptographic hardware does not support service or algorithm**

Explanation

A call requiring cryptographic hardware was made to ICSF. The current installation hardware does not support the service or algorithm that is being used.

User response

Ensure that the correct protocol is in use for your installation, and that the cryptographic hardware required for this service or algorithm is available to ICSF.

464 **Elliptic curve list is not valid.**

Explanation

The supported elliptic curve list is not formatted correctly or when running in FIPS mode, the removal of unsupported elliptic curves resulted in an empty list.

User response

Ensure the value that is supplied for GSK_CLIENT_ECURVE_LIST or GSK_SERVER_ALLOWED_KEX_ECURVES contains only entries for elliptic curves that are supported by System SSL. See Table 29 on page 804 for a list of supported elliptic curve definitions. Ensure that each entry uses 4 decimal digits.

If enabled for FIPS mode, elliptic curves x25519 and x448 are not supported. The specified list may also be tailored to meet the requirements of the FIPS level being utilized. For information about FIPS mode level support, see Chapter 4, "System SSL and FIPS 140-2," on page 19.

465 **ICSF PKCS #11 services are disabled**

Explanation

An attempt was made to use ICSF PKCS #11 services, which are disabled because of an ICSF FIPS self-test failure.

User response

Stop and restart ICSF. System SSL might need restarting to regain the full hardware benefit from ICSF. Contact your service representative if the error persists.

466 **Signature algorithm pairs list is not valid.**

Explanation

The supported signature algorithm pairs list is not correctly formatted.

User response

Ensure the value supplied for GSK_TLS_SIG_ALG_PAIRS, GSK_TLS_CERT_SIG_ALG_PAIRS, or GSK_OCSP_RESPONSE_SIGALG_PAIRS contains only valid entries for hash and signature algorithm pairs that are supported by System SSL and that each entry is defined using four characters. See Table 30 on page 804 and Table 31 on page 805 for a list of valid 4-character signature algorithm pair definitions.

467 **Signature algorithm not in signature algorithm pairs list.**

Explanation

A signature algorithm that is used to sign a local or peer certificate is not included in the signature algorithm pairs list or the certificate signature algorithm pairs list. The server certificate chain must use signature algorithms included in the signature algorithm pairs or the certificate signature algorithm pairs that are presented by the client during the TLS handshake. The client certificate chain must use signature algorithms included in the signature algorithm pairs or the certificate signature algorithm pairs that are presented by the server during the TLS handshake. If remote partner specifies the signature algorithm pairs and the certificate signature algorithm pairs, the specified certificate signature algorithm pairs take precedence over the signature algorithm pairs. For TLS V1.2 handshakes, the GSK_TLS_CERT_SIG_ALG_PAIRS setting is only used on the client side to indicate the signature algorithms that the client supports in the server's certificate chain.

User response

Ensure that the signatures of the local and peer certificates in the certificate chain use signature algorithms that are present in the signature algorithm pairs list or the certificate signature algorithm pairs list that is presented by the session partner. If the certificate chain is correct, configure the client or server or both to specify all necessary signature algorithms pairs in the GSK_TLS_SIG_ALG_PAIRS or GSK_TLS_CERT_SIG_ALG_PAIRS settings to allow use of the certificate chain. If GSK_TLS_CERT_SIG_ALG_PAIRS is specified, it takes precedence while checking the signature algorithms used in the certificate chain. See Table 30 on page 804 and Table 31 on page 805 for a list of valid 4-character signature algorithm pair definitions.

468 **Certificate key algorithm not in signature algorithm pairs list.**

Explanation

The certificate key algorithm of the local certificate cannot be used to generate digital signatures as it is not included in the signature algorithm pairs list. The server certificate must use a key algorithm included in the signature algorithm pairs list that is presented by the client during the TLS handshake and the server's supported signature algorithm pairs list. The client certificate must use a key algorithm included in the signature algorithm pairs list that is presented by the server during the TLS handshake and the client's supported signature algorithm pairs list.

User response

Ensure that the key algorithm of the certificate is present in the signature algorithm pairs list that is presented by the session partner. If the certificate is correct, configure the client or server or both to specify all necessary signature algorithm pairs in the environment variable GSK_TLS_SIG_ALG_PAIRS that allows the use of the certificate's key for generating digital signatures. See Table 31 on page 805 for a list of valid 4-character signature algorithm pair definitions.

469 **Incorrect key attribute.**

Explanation

One or more PKCS #11 attributes or parameters for a key are missing or incorrect for a requested function that is being performed. For example, a signing operation requires that for the key that is being used, the PKCS #11 sign attribute is to be TRUE. Verify that the correct key is being used for the requested function, and that all required attributes are set for that key. If you are using **gsk_make_enveloped_private_key_msg()**, ensure that a recipient certificate's RSA public key is valid.

User response

Verify that a certificate's PKCS #11 key attributes are correct for the function that is being performed.

470 **Certificate does not meet Suite B requirements.**

Explanation

The certificate in use does not meet the requirements for the Suite B profile that is selected for the environment.

User response

Ensure that the certificate used for the connection satisfies the requirements for the chosen Suite B profile. See "Suite B cryptography support" on page 49 for more information about Suite B certificate requirements.

471	**Secure private key cannot be used with a fixed ECDH key exchange.**

Explanation

A handshake attempted to perform an ECDH key exchange. The certificate's private key is a label that is pointing to a secure key. This is not a supported operation.

User response

Choose a certificate that does not have a secure private key or a cipher that does not perform an ECDH key exchange.

472	**Clear key support not available due to ICSF key policy.**

Explanation

Unable to generate clear keys or PKCS #11 objects because of the caller's RACF access to CRYPTOZ class resource CLEARKEY.SYSTOK-SESSION-ONLY does not allow the generation of non-secure (clear) PKCS #11 keys.

User response

Ensure that the user ID of the application has appropriate access to the RACF CRYPTOZ class resource CLEARKEY.SYSTOK-SESSION-ONLY.

473	**OCSP responder requires a signed request**

Explanation

The OCSP responder contacted for certificate validation requires that all OCSP requests be signed.

User response

Enable OCSP request signing by specifying the label of the signing certificate (GSK_OCSP_REQUEST_SIGKEYLABEL) and the signature algorithm (GSK_OCSP_REQUEST_SIGALG).

474	**HTTP response is not valid**

Explanation

The HTTP response received was not properly formatted or contents are not valid. The HTTP response received must be an HTTP/1.0 or HTTP/1.1 response.

User response

Ensure that the HTTP server is running, that there are no network errors, and the HTTP server sends its responses using HTTP/1.0 or HTTP/1.1 protocols. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

475	**OCSP response is not valid**

Explanation

The OCSP ASN.1 encoded response received from the OCSP responder was not properly formatted or its contents are not valid.

User response

Ensure that the OCSP responder server is properly encoding the OCSP response, that it is running, and that there are no network errors. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

476 **Session ID entry does not exist**

Explanation

For a client application, the session identifier or a session ticket specified by GSK_PEER_ID does not exist or has expired. If a client application has set GSK_REQ_CACHE_SESSION to ON, the required cached session entry could not be located. For a client application, the maximum number of SSL environment connections has been reached by a client application so new GSK_PEER_IDs cannot be stored in the session cache.

For a server application, the session identifier specified by GSK_SID_VALUE does not exist or has expired. If a server application has set GSK_SID_VALUE, the required cached session entry could not be located when the session is using an SSL V3 through TLS V1.2 handshake or the session ticket sent by the client does not contain the correct session identifier while attempting a TLS V1.3 handshake resumption.

User response

Verify that the specified session identifier is correct, is not expired, the cache is large enough to hold the cached session entry, and the maximum client connections has not been reached.

477 **Client SID does not match server SID**

Explanation

For SSL V3 through TLS V1.2 handshakes, the server failed the connection request because the session identifier provided by the client through the client hello does not match the server GSK_SID_VALUE that was specified by **gsk_attribute_set_buffer()**.

For TLS V1.3 handshakes, the server failed the connection request for one of the following reasons:

- The session or ticket identifier in the session ticket through the client hello does not match the server GSK_SID_VALUE that was specified by **gsk_attribute_set_buffer()**.
- A session ticket was not sent from the client.
- If sysplex session ticket caching is not enabled (GSK_SYSPLEX_SESSION_TICKET_CACHE option is set to OFF), the server is unable to decrypt the session ticket because the encryption key is not available.
- The session ticket is expired.

User response

Verify that the specified session identifier is correct.

If attempting to resume a TLS V1.3 connection, ensure that the GSK_SESSION_TICKET_SERVER_TIMEOUT and GSK_SESSION_TICKET_SERVER_KEY_REFRESH settings are appropriate to allow for the session ticket sent from the client to be used. The GSK_SESSION_TICKET_SERVER_KEY_REFRESH setting is only relevant when sysplex session ticket caching is not enabled and GSK_SESSION_TICKET_SERVER_ENABLE is set to ON. If the session ticket is expired or the key used to encrypt the session ticket is not available, session reuse will fail.

478 **Client session cache attributes do not agree**

Explanation

Client application attributes GSK_ENABLE_CLIENT_SET_PEERID, GSK_REQ_CACHED_SESSION, GSK_V3_SIDCACHE_SIZE, GSK_V3_SESSION_TIMEOUT, and GSK_SESSION_TICKET_CLIENT_ENABLE conflict.

User response

Verify that the settings for GSK_ENABLE_CLIENT_SET_PEERID, GSK_REQ_CACHED_SESSION, GSK_V3_SIDCACHE_SIZE, GSK_V3_SESSION_TIMEOUT, and GSK_SESSION_TICKET_CLIENT_ENABLE are in agreement.

Attribute conflicts may cause the application to not behave as desired and may result in handshake failures. SID cache reuse requires that the cache be defined and active.

One or more of the following conflicts may need to be corrected:

- A client application has set GSK_ENABLE_CLIENT_SET_PEERID to OFF and GSK_REQ_CACHED_SESSION to ON.
- A client application has set GSK_ENABLE_CLIENT_SET_PEERID to ON and attributes GSK_V3_SIDCACHE_SIZE or GSK_V3_SESSION_TIMEOUT to zero.
- A client application has set GSK_ENABLE_CLIENT_SET_PEERID to ON and attribute GSK_SESSION_TICKET_CLIENT_ENABLE to OFF when the connection handle is enabled for TLS V1.3.

479 **SID VALUE is not valid**

Explanation

A user-supplied GSK_SID_VALUE was encountered that is not valid in the **gsk_secure_socket_init()** routine.

User response

An application should only set GSK_SID_VALUE for a new connection if cache reuse is desired. The System SSL generated value can only be retrieved from SSL using **gsk_attribute_get_buffer()**. The buffer that is returned belongs to an active System SSL connection and should not be modified or the storage freed by the application. If the connection that provided the GSK_SID_VALUE buffer has closed, the data pointed to by the buffer cannot be determined. An application must copy this data into the application's own storage in order to continue reusing a particular cached session prior to closing the originating connection.

480 **PEER ID is not valid**

Explanation

A user-supplied GSK_PEER_ID was encountered that is not valid in the **gsk_secure_socket_init()** routine.

User response

An application should only set GSK_PEER_ID for a new connection if cache reuse is desired. The System SSL generated value can only be retrieved from SSL using **gsk_attribute_get_buffer()**. The buffer that is returned belongs to an active System SSL connection and should not be modified or the storage freed by the application. If the connection that provided the GSK_PEER_ID buffer has closed, the data pointed to by the buffer cannot be determined. An application must copy this data into the application's own storage in order to continue reusing a particular cached session.

481 **OCSP request failed with internal responder error**

Explanation

The OCSP responder contacted for certificate validation returned an internal error.

User response

Ensure that the OCSP responder server is running and that there are no network errors. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

482 **OCSP response is expired**

Explanation

The current time is after the OCSP response expiration time.

User response

If using the dedicated OCSP responder, ensure that the OCSP responder server is using the most recent revocation information available from the certification authority. If certificate revocation through the AIA extension is enabled, ensure that the OCSP responder servers referenced in the certificate chain are using the most recent revocation information available. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

483 **Error creating OCSP request**

Explanation

An internal error was encountered while creating the OCSP request to send to an OCSP responder.

User response

If OCSP request signing is enabled, verify that the signing certificate resides in the SAF key ring, SSL key database, PKCS#11 token, or PKCS#12 file and the signing certificate is valid (start time is before the current time and is not yet expired) and contains a private key. The key repository is provided through the GSK_KEYRING_FILE environment variable or attribute. The OCSP signing certificate is provided through the GSK_OCSP_REQUEST_SIGKEYLABEL environment variable or attribute.

Collect a System SSL trace containing the error and then contact your service representative if the error persists.

484 **Maximum response size exceeded**

Explanation

When attempting to retrieve revocation information, the HTTP response exceeded the maximum configured response size for either an OCSP response or a HTTP CRL. The response size is provided through either GSK_HTTP_CDP_MAX_RESPONSE_SIZE or GSK_OCSP_MAX_RESPONSE_SIZE environment variables or attributes.

User response

Ensure that the HTTP response maximum size is adequate for the size of the CRLs or OCSP responses that are being retrieved. If necessary, increase the maximum response size until an adequate size is provided to handle the CRLs or OCSP responses that are being retrieved. If unable to determine an adequate size, collect a System SSL trace containing the error and then contact your service representative.

485 **HTTP server communication error**

Explanation

Unable to establish a connection to contact the HTTP server or the OCSP responder to retrieve certificate revocation information.

User response

If enabled for OCSP and a dedicated OCSP responder is enabled, ensure that the responder is running and can be accessed.

If enabled for OCSP responders identified in the certificate AIA extension, ensure that the OCSP responders specified in the extension are running and can be accessed.

If HTTP CRL support is enabled, ensure that the HTTP server specified in the CRL Distribution Point extension is running and can be accessed.

If there is a firewall in place and either an HTTP proxy server or port or an OCSP proxy server or port has been identified, ensure that the servers and ports settings are correct and the servers can be accessed. The HTTP proxy server and port are specified through the GSK_HTTP_CDP_PROXY_SERVER_NAME and GSK_HTTP_CDP_PROXY_SERVER_PORT attributes or environment variables. The OCSP proxy server and port are specified through the GSK_OCSP_PROXY_SERVER_NAME and GSK_OCSP_PROXY_SERVER_PORT attributes or environment variables.

Collect a System SSL trace containing the error and then contact your service representative if the error persists.

486 **Nonce in OCSP response does not match value in OCSP request**

Explanation

When validating the nonce in the OCSP response, the value did not match the value sent in the OCSP request.

User response

If OCSP is enabled for the dedicated OCSP responder (GSK_OCSP_URL), ensure that the OCSP responder server is configured to send a nonce in OCSP responses.

Ensure that nonce checking is required. If not required, set GSK_OCSP_NONCE_CHECK_ENABLE to off.

Collect a System SSL trace containing the error and then contact your service representative if the error persists.

487 **OCSP response not received within configured time limit**

Explanation

The time limit indicated in the value for GSK_OCSP_RESPONSE_TIMEOUT has been exceeded.

User response

Ensure that the HTTP server where the OCSP responder resides is available and able to process OCSP requests. Verify that the value for GSK_OCSP_RESPONSE_TIMEOUT is sufficient to receive a complete response from the HTTP server containing the OCSP responder.

488 **Revocation information is not yet valid**

Explanation

The current time is earlier than the validity period of the revocation information provided though either an OCSP response or CRL.

User response

Ensure that the system time is configured correctly. Collect a System SSL trace containing the error and then contact your service representative if the problem persists.

489 **HTTP server host name is not valid**

Explanation

The URI value in the AIA extension or the CDP extension is not in the correct format or cannot be resolved by the Domain Name Service (DNS). The correct URI format is http://hostname[:portNumber].

User response

If the GSK_OCSP_ENABLE parameter is enabled, verify that the certificate being verified has a URI value in the AIA extension that is properly formatted and can be resolved by the DNS. It may be necessary to obtain a new certificate or to specify the GSK_OCSP_PROXY_SERVER_NAME and GSK_OCSP_PROXY_SERVER_PORT parameters if there is a need to pass through a firewall. If the

GSK_OCSP_URL or GSK_OCSP_PROXY_SERVER_NAME parameters are specified, verify that the host name and the IP address is properly formatted and can be resolved by the DNS.

If the GSK_HTTP_CDP_ENABLE parameter is enabled, verify that the certificate being verified has a URI value in the CDP extension that is properly formatted and can be resolved by the DNS. It may be necessary to obtain a new certificate or to specify the GSK_HTTP_CDP_PROXY_SERVER_NAME and GSK_HTTP_CDP_PROXY_SERVER_PORT parameters if there is a need to pass through a firewall.

490 **PKCS #12 file content not valid**

Explanation

When processing the PKCS #12 file, a format error was detected. This can occur if the file is not properly ASN.1 encoded, been modified if transferred, or the PKCS #12 file is not a Version 3 binary file. PKCS #12 Version 1 files and files in Base64 format are not supported.

User response

If the current file is either a PKCS #12 Version 1 file or a Base64 encoded file, it must be replaced with a PKCS #12 Version 3 file. If transferring the file, be sure to transfer the file in binary format. Correct the PKCS #12 file or obtain a new PKCS #12 file. If the problem persists, collect a System SSL Trace containing the error and then contact your service representative.

491 **Required basic constraints certificate extension is missing**

Explanation

During the establishment of an SSL/TLS secure connection (**gsk_secure_socket_init()** or **gsk_secure_soc_init()** API) with certificate validation processing set to mode ANY or 2459, an intermediate CA certificate was encountered outside of a trusted certificate source (for example, key database file, PKCS #12 file, SAF key ring, or PKCS #11 token), which does not have a basic constraints extension.

User response

Contact the connection partner to determine the certificates being utilized. Once the certificates are identified, either new valid Version 3 certificates can be obtained to replace the intermediate CA certificate or certificates causing the error, or if the usage of the intermediate CA certificates is deemed to be acceptable, a version of the CA certificate or certificates needs to be added to the application's trusted certificate source (for example, key database file, PKCS #12 file, SAF key ring, or PKCS #11 token).

If the error persists after adding the certificates, or if the certificates can not be readily obtained, collect a System SSL trace containing the error and then contact your service representative.

492 **Maximum number of locations allowed to be contacted during certificate validation has been reached**

Explanation

The number of locations allowed by either GSK_MAX_SOURCE_REV_EXT_LOC_VALUES or GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES has been exceeded. The locations for revocation information are specified by the accessLocation in the AIA certificate extension for OCSP and the distributionPoint in the CDP extension for HTTP CRLs.

User response

Use the values in the certificate chain being validated to determine the proper value for GSK_MAX_SOURCE_REV_EXT_LOC_VALUES, GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES, or both. The value for GSK_MAX_SOURCE_REV_EXT_LOC_VALUES must be greater than or equal to the maximum number of location values in a CDP or AIA extension used in the certificate chain being validated. The value for GSK_MAX_VALIDATION_REV_EXT_LOC_VALUES must be greater than or equal to the total number of location values in all CDP and AIA extensions used in the certificate chain being validated. Collect a System SSL trace containing the error and then contact your service representative if the problem persists.

493 **HTTP response not received within configured time limit**

Explanation

The time limit indicated in the value for GSK_HTTP_CDP_RESPONSE_TIMEOUT has been exceeded.

User response

Ensure that the HTTP server is available and able to process HTTP CRL requests. Verify that the value for GSK_HTTP_CDP_RESPONSE_TIMEOUT is sufficient to receive a complete response from the HTTP server

494 **LDAP response not received within configured time limit**

Explanation

The time limit indicated in the value for GSK_LDAP_RESPONSE_TIMEOUT has been exceeded.

User response

Ensure that the LDAP server is available and able to process LDAP CRL requests. Verify that the value for GSK_LDAP_RESPONSE_TIMEOUT is sufficient to receive a complete response from the LDAP server.

495 **OCSP request failed with try later error**

Explanation

The OCSP responder is unable to currently process the OCSP request.

User response

Contact the OCSP responder administrator to verify that the OCSP responder is working properly. Then retry the OCSP request at a later time.

496 **OCSP response signature algorithm not in signature algorithm pairs list.**

Explanation

The OCSP response was signed with an algorithm that was not specified in the OCSP response signature algorithm pairs list.

User response

Verify that the signature algorithms included in the response signature algorithm pairs list (GSK_OCSP_RESPONSE_SIGALG_PAIRS) are supported by the OCSP responder and are valid based on the certificate being validated. For example, the OCSP responder may ignore an signature algorithm of SHA-224 with RSA encryption if the certificate being validated is an ECDSA certificate. Ensure that the OCSP responder supports the OCSP preferred signature algorithms extension. The OCSP response signature algorithm pairs list may need to be updated to include the algorithm that the OCSP responder is using to sign the OCSP response. See Table 31 on page 805 for a list of valid 4-character signature algorithm pair definitions. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

497 **OCSP request signature algorithm pair is not valid.**

Explanation

The OCSP request signature algorithm pair specified in the GSK_OCSP_REQUEST_SIGALG environment variable or attribute type in **gsk_attribute_set_buffer()** is not valid. The valid values for the OCSP request signature algorithm pair definitions can be found in Table 31 on page 805.

User response

Correct the OCSP request signing algorithm that is specified in the GSK_OCSP_REQUEST_SIGALG environment variable or attribute type in **gsk_attribute_set_buffer()**.

498 **OCSP response does not contain requested certificate status.**

Explanation

The OCSP response from the OCSP responder does not contain the requested certificate status.

User response

Contact the OCSP responder administrator to verify that the OCSP responder is operating as expected. If the error persists, collect a System SSL trace containing the error and then contact your service representative.

499 **OCSP response contains duplicate certificate statuses.**

Explanation

The OCSP response from the OCSP responder contains duplicate certificate statuses and it is not possible to determine revocation status of the requested certificate.

User response

Contact the OCSP responder administrator to verify that the OCSP responder is operating as expected. If the error persists, collect a System SSL trace containing the error and then contact your service representative.

500 **Triple DES key parts are not unique**

Explanation

During the negotiation of a secure connection utilizing a triple DES cipher with unique key enforcement enabled, the generated triple DES key did not meet the requirement of each key part being unique. The three key parts are checked for uniqueness in a non-FIPS mode environment when attribute GSK_3DES_KEYCHECK has been set to GSK_3DES_KEYCHECK_ON. Triple DES key part uniqueness checking always takes place in FIPS mode.

User response

Try the connection again.

501 **Buffer size is not valid.**

Explanation

The socket buffer or buffer size is not valid.

User response

Specify a valid buffer and buffer size.

502 **Socket request would block.**

Explanation

The socket is in non-blocking mode and the socket request returned the EWOULDBLOCK error.

User response

Retry the **gsk_secure_socket_read()** or **gsk_secure_socket_write()** request when the socket is ready to send or receive data.

503 **Socket read request would block.**

Explanation

A socket read request that is issued as part of an SSL handshake returned the EWOULDBLOCK error.

User response

Retry the failing request when the socket is ready to receive data.

504 **Socket write request would block.**

Explanation

A socket write request that is issued as part of an SSL handshake return the EWOULDBLOCK error.

User response

Retry the failing request when the socket is ready to send data.

505 **Record overflow.**

Explanation

An SSL protocol record has a plain text record length greater than 16384 or an encrypted text record length greater than 18432.

User response

Ensure that data is not being corrupted during transmission. Obtain a System SSL trace containing a dump of the failing record and contact your service representative if the error persists.

507 **Certificate version does not meet the minimum required version level**

Explanation

An X.509 end-entity certificate being utilized by the peer did not meet the minimum allowed System SSL X.509 certificate version.

User response

Ensure that the peer's certificate version is at least equal to the minimum supported version. The minimum version value can be modified using GSK_PEER_CERT_MIN_VERSION. If a TLS V1.3 connection is being negotiated, the certificate must be version 3.

508 **Key size is smaller than the minimum size allowed.**

Explanation

During the negotiation of a secure connection, the peer's X.509 end-entity certificate contains a RSA, DSA, DH, or ECC key that is smaller than the minimum size allowed.

User response

Ensure that the key sizes are at least equal to the minimum supported key sizes. The minimum key size value can be modified using attributes GSK_PEER_RSA_MIN_KEY_SIZE, GSK_PEER_DSA_MIN_KEY_SIZE, GSK_PEER_DH_MIN_KEY_SIZE, and GSK_PEER_ECC_MIN_KEY_SIZE.

If a TLS V1.3 connection is being negotiated:

- The minimum key size for an RSA peer certificate is the larger of the following two values: 2048 or the value specified in the GSK_PEER_RSA_MIN_KEY_SIZE attribute.
- The minimum key size for an ECC peer certificate is the larger of the following two values: 256 or the value specified in the GSK_PEER_ECC_MIN_KEY_SIZE attribute.

509 **Key label list is not valid**

Explanation

When using the environment variable GSK_SERVER_KEY_LABEL_LIST, one of the following may have occurred:

- More than 8 key labels were specified.
- A specified key label is greater than 127 characters. The backslash (\) characters used as an escape character for a blank space or comma in the label name are not counted as part of that 127 character maximum length.
- List did not consist of at least one key label.

User response

Ensure that the key label list is valid.

510 **No acceptable key labels found**

Explanation

When using the attribute GSK_SERVER_KEYRING_LABEL_LIST, none of the certificates supported the protocol, ciphers (TLS V1.2 and earlier), or the session attributes defined for an SSL connection. Certificates that are either expired or not yet valid are ignored.

User response

Ensure that at least one of the certificates defined by attribute GSK_SERVER_KEYRING_LABEL_LIST supports the requested protocol, cipher (TLS V1.2 and earlier), and session attributes of an SSL connection. Collect a System SSL trace containing the error and contact your service representative if the error persists.

511 **OCSP stapling requires OCSP support to be enabled.**

Explanation

OCSP stapling on the System SSL server application requires either the GSK_OCSP_URL to be specified or the GSK_OCSP_ENABLE option to be set to ON.

User response

Ensure that GSK_OCSP_URL is specified or the GSK_OCSP_ENABLE option is set to ON when OCSP stapling on the server is enabled. Otherwise, turn off OCSP stapling by setting the GSK_SERVER_OCSP_STAPLING environment variable to OFF or by calling the **gsk_attribute_set_enum()** routine with the GSK_SERVER_OCSP_STAPLING attribute type set to GSK_SERVER_OCSP_STAPLING_OFF.

512 **Certificate status response is not valid.**

Explanation

System SSL is configured for server OCSP stapling. If the negotiated protocol is TLS V1.2 and earlier, the TLS server has sent a CERTIFICATE-STATUS response message that the TLS client does not understand. If the negotiated protocol is TLS V1.3, the TLS server has sent a Certificate Status Request extension containing the OCSP response that the TLS client does not understand. The TLS handshake is aborted.

User response

If the negotiated protocol is TLS V1.2 and earlier, the OCSP responder that System SSL has contacted may have provided an improperly formatted OCSP response that System SSL has included in the CERTIFICATE-STATUS message. Verify that the TLS client is able to handle the OCSP responses that reside in the CERTIFICATE-STATUS message.

If the negotiated protocol is TLS V1.3, the OCSP responder that System SSL has contacted may have provided an improperly formatted OCSP response that System SSL has included in the Certificate Status Request extension. Verify that the TLS client is able to handle the OCSP responses that reside within the Certificate Status Request extension in the server's CERTIFICATE message.

If the error persists, collect a System SSL trace containing the error and then contact your service representative.

513 **An inappropriate protocol fallback is detected**

Explanation

The server is enabled for the Signaling Cipher Suite Value (SCSV) support by setting the GSK_SERVER_FALLBACK_SCSV attribute type on **gsk_attribute_set_enum()** to GSK_SERVER_FALLBACK_SCSV_ON or the GSK_SERVER_FALLBACK_SCSV environment variable to ON. When the SCSV is present in the client's supported cipher list during a handshake, it indicates that the client is attempting a fallback handshake with an earlier SSL or TLS protocol. If the server's highest supported TLS or SSL protocol level is greater than the client's SSL or TLS protocol level, the handshake fails with this error.

User response

If the server is configured appropriately with the GSK_SERVER_FALLBACK_SCSV attribute type or environment variable enabled, no action is required as the server is appropriately rejecting a client's attempt to do a handshake at an earlier TLS or SSL protocol level.

Ensure that the client is properly enabled for the highest SSL or TLS protocol level supported by the server and retry the initial handshake without sending the SCSV to the server.

514 **Required TLS extension is missing from remote partner**

Explanation

The remote partner did not send a required TLS extension during the TLS handshake process. The TLS handshake cannot continue.

User response

Ensure that the remote partner is specifying the required TLS extensions that are required for the secure TLS connection. If a TLS V1.3 handshake is attempted, ensure that the remote partner is including the signature algorithms, key share, and supported groups extensions. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

515 **Key share list is not valid.**

Explanation

The specified key share list is not configured or set, not formatted correctly, or contains groups that are not supported by TLS V1.3.

User response

Ensure the values that are supplied for GSK_CLIENT_TLS_KEY_SHARES or GSK_SERVER_TLS_KEY_SHARES only specifies groups that are supported by System SSL for TLS V1.3. Ensure that each entry uses 4 decimal digits and is not empty. See Table 29 on page 804 for a list of supported key share group curve definitions for TLS V1.3.

516 **No key share groups in common with partner**

Explanation

The client and server key share groups do not contain at least one value in common.

User response

Check with the remote partner to determine the key share groups that are to be used and supported by both sides.

If running as a server application, ensure that there is at least one key share group specified in the GSK_SERVER_TLS_KEY_SHARES that is also supported by the remote client partner.

If running as a client application, the remote server partner has selected a key share group that has not been specified in the GSK_CLIENT_TLS_KEY_SHARES list. It may be necessary to update the client's GSK_CLIENT_TLS_KEY_SHARES and GSK_CLIENT_ECURVE_LIST settings to include the server's selected key share group. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

517 **No matches between elliptic curve and key share lists**

Explanation

On the client side when TLS V1.3 is enabled, System SSL uses the GSK_CLIENT_ECURVE_LIST setting to specify the elliptic curves or supported groups that are sent to the server. The GSK_CLIENT_TLS_KEY_SHARES setting specifies the key share groups and it must contain at least one of the groups in the GSK_CLIENT_ECURVE_LIST setting.

On the server side, System SSL has selected TLS V1.3 for the handshake and has determined that the client has sent the elliptic curves or supported groups in a different order than the key share groups.

User response

If running as a client application, update the GSK_CLIENT_ECURVE_LIST or the GSK_CLIENT_TLS_KEY_SHARES to include at least one group that is in common between the two lists. The GSK_CLIENT_TLS_KEY_SHARES list should be a subset of the groups that are specified in the GSK_CLIENT_ECURVE_LIST. Verify that the GSK_CLIENT_ECURVE_LIST is not empty (NULL).

If running as a server application, contact the remote client partner to verify the elliptic curves or supported groups and the key shares are properly configured. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

518 **Alert received from remote partner is allowed by protocol but not expected by System SSL**

Explanation

An alert has been received from the remote partner that is supported by the protocol, but is not expected by System SSL.

User response

Contact the remote partner and verify that it is sending the appropriate alert in this scenario. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

519 **Required ciphers have not been specified**

Explanation

This error can occur for the following reasons:

- The client has enabled the TLS V1.3 protocol and has not specified at least one valid TLS V1.3 cipher specification.
- The client has enabled the TLS V1.3 protocol along with another TLS protocol and has not specified at least one cipher that is supported with the TLS V1.0, TLS V1.1, or TLS V1.2 protocol.

User response

If the client is enabled for the TLS V1.3 protocol:

- Verify that there is at least one valid TLS V1.3 cipher specification has been specified in the GSK_V3_CIPHER_SPECS_EXPANDED setting.
- Verify that there is at least one cipher specification in the GSK_V3_CIPHER_SPECS_EXPANDED setting that is supported with the other TLS protocols that are enabled.

For more information about the supported cipher specifications, see Table 27 on page 799.

520 **4-character cipher specifications are required**

Explanation

If TLS V1.3 is enabled, 4-character cipher specifications must be used.

User response

The application must call **gsk_attribute_set_enum()** and set the enumeration identifier GSK_V3_CIPHERS to have a value of GSK_V3_CIPHERS_CHAR4. Once enabled for 4-character cipher specifications, the application uses the GSK_V3_CIPHER_SPECS_EXPANDED setting for the cipher specifications.

For more information about the supported cipher specifications, see Table 27 on page 799.

521 **Client has detected that the server has attempted an unexpected protocol fallback**

Explanation

An unexpected protocol fallback to an earlier TLS protocol has been detected by the client.

User response

A fallback to an earlier TLS protocol has been detected by the client although the server supports a later TLS protocol. Verify that the application is communicating with the expected remote partner. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

522 **Signature algorithm used by the remote partner for a secure connection is not correct.**

Explanation

The remote partner used an incorrect or an unexpected signature algorithm during the TLS V1.3 handshake. The remote partner signed its TLS V1.3 handshake messages with a signature algorithm that was not included in its local signature algorithm list (GSK_TLS_SIG_ALG_PAIRS) or one that is not allowed to be used with TLS V1.3.

User response

- Contact the remote partner and verify that it is using the correct signature algorithm for the TLS V1.3 handshake. The remote partner may use a signature algorithm that is not supported by the local application to sign TLS handshake messages. If communicating with a z/OS System SSL remote partner, it may use a preferred signature algorithm if there are no signature algorithms in common. See "Signature and hash algorithms" on page 62 for more information. In these cases, the local GSK_TLS_SIG_ALG_PAIRS setting should be updated to support the remote partner's certificate chain.
- If the remote partner is using an RSA certificate, an RSASSA-PSS signature algorithm (0804, 0805, or 0806) must be used to sign the TLS V1.3 handshake messages.
- The remote partner used a signature algorithm that is not supported in TLS V1.3 for the certificate verify message. These signature algorithms are not supported in TLS V1.3: Any DSA signature algorithm, RSASSA-PSS and ECDSA with SHA-1 and SHA-224 signature algorithms, and RSA with MD2, MD5, SHA-1 and SHA-224 signature algorithms.

If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

523 **Key share list cannot contain more groups than the elliptic curve list**

Explanation

There are more groups present in the client's key share group list (GSK_CLIENT_TLS_KEY_SHARES) than there are present in the client's elliptic curve or supported groups list (GSK_CLIENT_ECURVE_LIST). The groups specified in the client's key share list should be a subset of the groups specified in the client's elliptic curve or supported groups list.

User response

Verify that the groups specified for the client's key share group list (GSK_CLIENT_TLS_KEY_SHARES) and the client's elliptic curve or supported groups list (GSK_CLIENT_ECURVE_LIST) are correct. See Table 29 on page 804 for the supported groups and curve specifications.

524 **Remote partner indicates a required TLS extension is missing**

Explanation

The remote partner has indicated that the local partner did not send a required TLS extension during the TLS handshake process. The TLS handshake cannot continue.

User response

Ensure that this application is specifying the required extensions by the communicating partner. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

525 **Key share list received from remote partner is not correct**

Explanation

The remote partner has sent a key share list that is not valid. The key share list contains duplicate key share groups. The TLS handshake cannot continue.

User response

Ensure that the remote partner is sending a valid key share list. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

526 **Missing required certificate request signature algorithms**

Explanation

The remote server partner did not send the required signature algorithm extension in the certificate request.

User response

Ensure that the remote server partner has sent a certificate request and that it contains a required signature algorithm extension. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

527 **Local certificate version does not meet the minimum required version level**

Explanation

An X.509 certificate being utilized locally did not meet the minimum allowed System SSL X.509 certificate version. TLS V1.3 requires certificates to be X.509 version 3 certificates.

User response

Ensure that the version of the locally selected certificate is valid.

528 **Local certificate key size is smaller than the minimum size allowed**

Explanation

The local X.509 certificate contains an RSA or ECC key that is smaller than the minimum size allowed. TLS V1.3 requires that RSA keys have a minimum size of at least 2048 bits, and ECC keys must have a minimum size of at least 256 bits.

User response

Ensure that the key sizes are at least equal to the minimum supported key sizes for the protocol.

529 **Certificate key algorithm is not valid**

Explanation

The certificate key algorithm is not valid for the selected protocol. If the selected protocol is TLS V1.2 and earlier, the server certificate can use either RSA, DSA, Diffie-Hellman, or ECC as the public/private key algorithm. If the selected protocol is TLS V1.3, the server certificate can use either RSA or ECC as the public/private key algorithm.

User response

Specify a certificate with the appropriate key type.

530 **TLS handshake message from the remote partner included an extension that is not allowed or supported**

Explanation

A TLS handshake message from the remote partner contained an extension that is not allowed or supported. The remote partner did not correctly encode the TLS handshake message.

User response

Ensure that the remote partner is correctly encoding the TLS handshake message. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

531 **Legacy compression field must be a single byte set to 0**

Explanation

A TLS handshake message contains an incorrect compression field. The compression field must be a single byte set to 0.

User response

Ensure that the remote partner is correctly encoding the TLS handshake message. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

532 **Session ID received from remote partner is not correct**

Explanation

A TLS handshake message from the remote partner contains a Session ID value that is different from the Session ID that was sent to the remote partner. The Session ID value in the TLS handshake message from the remote partner must be the same. The attempted TLS handshake fails.

User response

Ensure that the remote partner is correctly encoding the TLS handshake message. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

533 **Remote partner indicates unsupported certificate**

Explanation

The remote partner has indicated that it does not support the certificate that has been sent to it or the remote partner's own certificate is not supported.

User response

Ensure that the remote partner is correctly configured to receive the configured certificate and the remote partner's own certificate is supported. The remote partner may need to update its supported signature list to allow the certificate’s signature to be used. If the remote partner is another System SSL application, the GSK_TLS_SIG_ALG_PAIRS or GSK_TLS_CERT_SIG_ALG_PAIRS settings may need to be updated. The remote partner may require a minimum key certificate key size. If the remote partner is another System SSL application, the GSK_PEER_RSA_MIN_KEY_SIZE, GSK_PEER_ECC_MIN_KEY_SIZE, GSK_PEER_DSA_MIN_KEY_SIZE, or GSK_PEER_DH_MIN_KEY_SIZE settings may need to be adjusted on the remote partner. If the remote partner is configured correctly, it may be necessary to update the local certificate to one that is supported by the remote application.

If a TLS V1.3 handshake is attempted, there are limitations on the key type and sizes that are allowed for TLS V1.3. See “gsk_secure_socket_init()” on page 157 's usage section for more information.

If the remote partner supports the certificate sent to it, the remote partner's own certificate may need to be updated. If the remote partner is another System SSL application running in FIPS mode, ensure that the remote partner's certificate is supported. See "Algorithms and key sizes" on page 19 for more information about the supported certificates in FIPS mode.

If the problem persists, contact the remote partner application to collect additional diagnostic data to determine why the sent certificate or the remote partner's own certificate is not supported.

534 **Remote partner indicates an incorrect PSK identity value**

Explanation

The client attempted a TLS V1.3 session resumption with only one session ticket present in its pre-shared key extension. The server responded with an unexpected index value present in its pre-shared key extension response.

User response

Ensure that the remote server partner is operating correctly and supports TLS V1.3 session resumption. If the problem persists, contact the vendor of the remote server partner.

535 **PSK exchange modes extension from the remote partner does not contain supported value**

Explanation

System SSL supports only PSK with (EC)DHE key establishment for TLS V1.3 resumption. The remote partner has not indicated its support of PSK with (EC)DHE.

User response

Ensure that the remote partner supports and is specifying support for PSK with (EC)DHE key establishment for TLS V1.3 resumption. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

If the remote partner is another SSL/TLS provider, contact the vendor of that product.

536 **TLS session has expired**

Explanation

The server can no longer send or accept session tickets because the number of seconds since the initial handshake has exceeded the server timeout value.

User response

The server cannot send or receive session tickets until the remote partner has requested a new connection and a full TLS V1.3 handshake has been negotiated. If the problem persists, the GSK_SESSION_TICKET_SERVER_TIMEOUT setting may need to be increased.

537 **Attempt to send session ticket failed due to incorrect resumption attributes**

Explanation

The server is not configured correctly to send a session ticket. This return code is returned to the caller when GSK_SEND_SESSION_TICKET has been specified in a call to **gsk_secure_socket_misc()**. It can occur because:

- GSK_SESSION_TICKET_SERVER_ENABLE is set to OFF.
- GSK_SESSION_TICKET_SERVER_COUNT is greater than 0.
- A client is attempting to send a session ticket.

User response

Verify the following:

- GSK_SESSION_TICKET_SERVER_ENABLE is set to ON.
- GSK_SESSION_TICKET_SERVER_COUNT is set to 0.
- The call is being made by a server session.

If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

538 **Remote partner used an incorrect cipher on a resumption attempt**

Explanation

The client detected that the server selected an incompatible cipher specification on the TLS V1.3 resumption handshake attempt. Servers supporting TLS V1.3 resumptions may use a different cipher specification on the resumed handshake as long as the hash algorithm associated with the two ciphers are compatible. If the initial handshake used cipher 1301 and the server selected cipher 1303 on the resumed handshake, this is not allowed as cipher 1301 uses a hash algorithm of SHA-256 while cipher 1303 uses a hash algorithm of SHA-384. The resumption handshake attempt fails.

User response

Contact the remote server partner to determine why they allowed a TLS V1.3 resumption to occur with an incompatible cipher specification. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

If the remote partner is another SSL/TLS provider, contact the vendor of that product.

539 **Input buffer contents not valid**

Explanation

The provided buffer does not contain valid data for the function being performed.

User response

Provide valid data in the buffer.

540 **Protocol is not supported in FIPS mode**

Explanation

TLS V1.3 protocol is not supported in FIPS mode.

User response

Currently, the TLS V1.3 protocol is not supported in FIPS mode and is only supported in non-FIPS mode. If the TLS V1.3 protocol is to be enabled, it must be done in non-FIPS mode. Once FIPS mode is disabled in the application, TLS V1.3 can be enabled by setting the GSK_PROTOCOL_TLSV1_3 environment variable or attribute type in **gsk_attribute_set_enum()**.

541 **Remote partner indicates sent certificate is not valid.**

Explanation

The certificate sent to the remote partner during the handshake was not acceptable to the remote partner.

User response

Ensure that the remote partner application is correctly configured to accept the locally configured certificate. The remote partner may be missing a root CA certificate or does not support the locally configured certificate that has been sent to it. If the remote partner is a System SSL application, collect traces of the failure on both sides and then contact your service representative if the error persists. If the remote partner is another SSL/TLS provider, contact the vendor of that product.

542 **Remote partner indicates a handshake failure due to incompatible security parameters.**

Explanation

The remote partner is unable to successfully perform a handshake due to incompatible security parameters. This error can occur if:

- The client and server cipher specifications do not contain at least one value in common. Client and server cipher specifications might be limited depending on which System SSL FMIDs are installed. See Appendix C, "Cipher suite definitions," on page 795 for more information. Server cipher specifications are dependent on the type of algorithms that are used by the server certificate (RSA, DSA, ECDSA, or Diffie-Hellman), which might limit the options available during cipher negotiation.
- If this is a client application attempting a TLS V1.3 handshake, ensure that the remote server partner has support for the groups specified in the GSK_CLIENT_TLS_KEY_SHARES and GSK_CLIENT_ECURVE_LIST settings.
- The remote partner indicated a handshake failure which is a generic error that is encountered during the handshake. This can occur for various reasons such as (but not limited to) the following:
  - The remote server partner's certificate chain contains a certificate using a signature algorithm pair that was not specified in either the GSK_TLS_CERT_SIG_ALG_PAIRS or GSK_TLS_SIG_ALG_PAIRS set by the client when attempting a TLS V1.3 handshake.

User response

Ensure that the client and the server have at least one cipher specification. If the problem persists, collect a System SSL trace and contact the remote partner to obtain the necessary diagnostics or traces to determine why the handshake failed.

Ensure that the GSK_TLS_CERT_SIG_ALG_PAIRS or GSK_TLS_SIG_ALG_PAIRS setting is defined to include all of the signature algorithm pairs that are both supported by the local application and expected from the remote peer.

543 **Remote partner indicates sent TLS handshake message is not valid.**

Explanation

The remote partner has indicated that a TLS message sent from the local application is not valid.

User response

Ensure that the remote partner is correctly parsing the sent TLS message. If the problem persists, collect a System SSL trace and contact the remote partner to obtain the necessary diagnostics or traces to determine why the TLS message is not valid.

544 **Missing required TLS extended master secret extension from remote partner**

Explanation

The TLS server or client encountered a communicating partner that does not support the extended master secret extension. The TLS extended master secret extension is specified as required locally and the communicating partner did not send or provide the extension during the TLS V1.0, TLS V1.1, or TLS V1.2 handshake.

User response

If running as a client application, the GSK_CLIENT_EXTENDED_MASTER_SECRET option has been set to REQUIRED; however, the remote server partner did not include the extended master secret extension during the TLS V1.0, TLS V1.1, or TLS V1.2 handshake. Ensure that the GSK_CLIENT_EXTENDED_MASTER_SECRET option is set appropriately for the remote server partners that are being contacted. If the server partner is a z/OS System SSL application, ensure that the GSK_SERVER_EXTENDED_MASTER_SECRET option is set to ON or REQUIRED and it must be running z/OS V2R3 or later and have PTFs for APAR OA60105 (z/OS V2R3 and V2R4) applied and active. If the remote server partner is another SSL/TLS provider, contact the remote server partner vendor to enable support for the extended master secret extension.

If running as a server application, the GSK_SERVER_EXTENDED_MASTER_SECRET option has been set to REQUIRED; however, the remote client partner did not include the extended master secret extension during the TLS V1.0, TLS V1.1, or TLS V1.2 handshake. Ensure that the GSK_SERVER_EXTENDED_MASTER_SECRET option is set appropriately for the remote client partners that are communicating with this server. If the client partner is a z/OS System SSL application, ensure that the GSK_CLIENT_EXTENDED_MASTER_SECRET option is set to ON or REQUIRED and it must be running z/OS V2R3 or later and have PTFs for APAR OA60105 (z/OS V2R3 and V2R4) applied and active. If the remote client partner is another SSL/TLS provider, contact the remote client partner vendor to enable support for the extended master secret extension.

545 **Extended master secret extension mismatch detected on cached TLS handshake attempt**

Explanation

The TLS client attempted to use a previously negotiated session with the server that did not originally negotiate the extended master secret extension. On the cached handshake attempt, the remote server indicated that it was able to successfully use the cached session; however, it included the extended master secret extension which is not allowed.

User response

Contact the remote server partner to determine why it is sending the extended master secret extension on a cached TLS handshake. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

546 **Incorrect extended key usage.**

Explanation

The extended key usage certificate extension does not permit the requested key operation. This error can occur if the extended key usage extension of a client or server certificate restricts the purposes for which the certificate's key can be used.

- Server certificates used for server authentication in 3280 or 5280 validation mode with an extended key usage extension must allow either the serverAuth or the anyExtendedKeyUsage purpose.
- Client certificates used for client authentication in 3280 or 5280 validation mode with an extended key usage extension must allow either the clientAuth or the anyExtendedKeyUsage purpose.

User response

Specify a certificate with the appropriate extended key usage.

547 **Session ticket information cannot be successfully cached**

Explanation

When trying to cache updated session ticket information prior to sending a TLS V1.3 session ticket, the System SSL started task, GSKSRVR, could not be contacted. The session ticket is not sent by the System SSL server application.

User response

Verify that the System SSL started task, GSKSRVR, is running when GSK_SYSPLEX_SESSION_TICKET_CACHE is enabled. If sysplex ticket caching is not required, disable GSK_SYSPLEX_SESSION_TICKET_CACHE in the server application. If the problem persists, contact the service representative.

548 **Server certificate validation failed with provided reference ID list**

Explanation

The reference list provided was not valid and server domain-based validation failed. This error might occur if:

- The list provided contains more than 16384 characters.
- The list contains an ID with less than three labels.
- There were no matches found between the client's reference ID list and the server certificate.
- Subject DN common name validation requested (GSK_REFERENCE_ID_CN) and the server certificate contains a subject alternative name extension with a DNS name entry. Validation must be performed using the subject alternative name extension.
- The server's certificate is not type x509_name_dn.

User response

Ensure that the reference ID list required is appropriately set to match the server that the client is attempting to connect to.

To validate the server's certificate with a subject alternative name DNS, set GSK_REFERENCE_ID_DNS with a list of DNS name values. A list can be set with GSK_REFERENCE_ID_CN to compare against the server certificate's DN common name when no subject alternative name DNS is present in the server certificate.

601 **Protocol is not SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, or TLS V1.3.**

Explanation

The requested function requires the SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, or TLS V1.3 protocol.

User response

Ensure that the correct protocol is in use before issuing the request.

602 **Function identifier is not valid.**

Explanation

The function identifier that is specified for **gsk_secure_socket_misc()** is not valid.

User response

Specify a valid function identifier.

603 **Specified function enumerator is not valid.**

Explanation

The value that is specified is not a value that is enumerated as a function for the API.

User response

Ensure that the correct function enumerator is coded for the function.

---

604 **Send sequence number is near maximum value**

Explanation

While using TLS V1.1 or higher protocol, the send sequence number is near the maximum value before which it wraps. For TLS V1.1 and higher, an SSL handshake must occur to reset the send sequence number before the sequence number wrapping. System SSL is unable to automatically initiate a handshake on the current function call. This code is not returned again until after a handshake for the connection resets the send sequence number and the send sequence number is again near the maximum value.

User response

The caller should initiate a handshake by calling gsk_secure_socket_misc, specifying GSK_RESET_CIPHER.When the handshake is initiated, the previous function call that returned this code can be called again.

---

605 **Specified function not supported by protocol version**

Explanation

The **gsk_secure_socket_misc()** functional identifier is not supported by the protocol version.

User response

The functional identifier is not supported by the connection protocol. GSK_RESET_WRITE_CIPHER is only supported by TLS V1.3 connections.

---

701 **Attribute identifier is not valid.**

Explanation

The attribute identifier is not valid.

User response

Specify a valid attribute identifier.

---

702 **Attribute length is not valid.**

Explanation

The attribute length is not valid.

User response

Specify a valid attribute length.

---

703 **Enumeration is not valid.**

Explanation

The enumeration value is not valid.

User response

Specify a valid enumeration value.

---

704 **Session identifier cache callback is not valid.**

Explanation

The session identifier cache callback values are not valid. All callback routines must be provided to use an application session identifier cache.

User response

Specify valid session identifier cache callback values.

705 **Numeric value is not valid.**

Explanation

The numeric value is not valid.

User response

Specify a valid numeric value.

706 **Attribute parameter is not valid.**

Explanation

The attribute parameter value is not valid.

User response

Specify a valid attribute parameter value.

707 **TLS extension type is not valid.**

Explanation

The TLS extension type is not valid or not supported.

User response

Specify a valid or supported TLS extension type value.

708 **Supplied TLS extension data is not valid.**

Explanation

TLS extension data that is submitted to the SSL environment or connection is incorrectly defined.

User response

Ensure that the TLS extension data is correctly defined. If the problem persists collect a System SSL trace and contact your service representative.

Deprecated SSL function return codes

The deprecated System SSL functions return the value 0 (GSK_OK) if no error is detected. Otherwise, one of the return codes listed in the gskssl.h include file is returned.

1 **Error detected while reading certificate database**

Explanation

An error is detected while reading the key database, the PKCS #12 file, or retrieving entries from the SAF key ring or z/OS PKCS #11 token.

User response

Collect a System SSL trace containing the error and then contact your service representative.

2 **Error detected while opening the certificate database.**

Explanation

An error is detected while opening the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. This error can occur if no name is supplied or the database, key ring, or token does not exist. When using a PKCS #12 file, the file name cannot end with .kdb, .rdb or .sth.

User response

Verify that the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token exists and is accessible by the application. This value is case-sensitive. Ensure that the case is preserved with your request. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

3 **Incorrect key database record format.**

Explanation

The record format for a key database entry is not correct. This error can occur if the name of a request database is provided instead of the name of a key database.

User response

Ensure that the correct database name is used. Collect a System SSL trace containing a dump of the keyfile entry and then contact your service representative if the error persists.

4 **Key database password is not correct.**

Explanation

The System SSL run time is unable to decrypt a keyfile entry. Either the supplied keyfile password is incorrect or the keyfile is damaged.

User response

Ensure that the correct keyfile password is used and both the file and directory path are accessible to the application

9 **Key label does not exist.**

Explanation

The supplied label or the default key is not found in the key database or the certificate is not trusted. If using a PKCS #12 file as the certificate database, the label is either the certificate's friendly name or the subject's distinguished name. A default key does not exist in a PKCS #12 file.

User response

Supply a valid label or define a default key in the key database. If using a PKCS #12 file, use the **gskkyman** command line option **-dc** or **-dcv** to display the contents of the PKCS #12 file. The friendly name or subject distinguished name values is displayed in the label field.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate's trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate's profile is generic. A (G) indicates generic.

12 **Key label is not found.**

Explanation

The requested key label is not found in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. When using a PKCS #12 file, this error can also occur when the file is being processed during the establishment of the SSL/TLS environment when a certificate is encountered where there is no friendly name PKCS #12 attribute and the certificate's subject distinguished name is empty.

User response

Specify a label that exists in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. If encountered when establishing an SSL/TLS environment using a PKCS #12 file, verify any certificate that has no subject distinguished name is assigned a PKCS #12 friendly name attribute.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate's trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate's profile is generic. A (G) indicates generic.

13 **Duplicate subject names.**

Explanation

The key database, SAF key ring, or z/OS PKCS #11 token contains multiple certificates with the same subject name as the DN specified in the **gsk_secure_soc_init()** initialization data.

User response

Either remove the duplicate certificates or specify a label instead of a DN in the **gsk_secure_soc_init()** initialization data.

16 **Incorrect key database password.**

Explanation

The System SSL run time is unable to decrypt a key database entry. Either the supplied database password is incorrect or the database is damaged.

User response

Ensure that the correct key database password is used. Re-create the database if the error persists.

17 **Key database password is expired.**

Explanation

The key database password is expired.

User response

Use the **gskkyman** utility to assign a new password for the key database.

18 **No certification authority certificates.**

Explanation

The key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token does not contain any valid certification authority certificates. The SSL run time needs at least one CA or self-signed certificate to perform client authentication.

User response

Add the necessary certificates to the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token and ensure that existing certificates are valid and have not expired.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate's trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate's profile is generic. A (G) indicates generic.

19 **No certificates available.**

Explanation

The key database, PKCS #12 file. SAF key ring, or z/OS PKCS #11 token does not contain any certificates, or the SSL client application does not have a certificate available when authentication is requested by the server.

User response

Check for available certificates and add the user certificate and any necessary certification authority certificates to the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token, if necessary. Specify a certificate for the client application to use.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate's trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate's profile is generic. A (G) indicates generic.

70 **Application is not APF-authorized.**

Explanation

The **gsk_srb_initialize()** routine is called but the program is not APF-authorized. SRB mode cannot be used by unauthorized applications.

User response

Contact your system programmer to get your application authorized.

71 **Unable to establish ESTAE environment.**

Explanation

The **gsk_srb_initialize()** routine is unable to establish the ESTAE error recovery environment.

User response

Contact your service representative.

72 **Unable to create service thread.**

Explanation

The **gsk_srb_initialize()** routine is unable to create a thread to handle SRB processing.

User response

Ensure that POSIX thread support is available to the application environment. Contact your service representative if the error persists.

100 **Initialization parameter is not valid**

Explanation

An initialization parameter for **gsk_initialize()** or **gsk_secure_soc_init()** is not valid.

User response

Ensure that all of the parameters are correct. Contact your service representative if the error persists.

102 **Security type is not valid**

Explanation

The security type that is specified in the initialization data for the **gsk_initialize()** routine is not valid.

User response

Specify a valid security type for the *sec_types* parameter.

103 **SSL V2 session timeout is not valid.**

Explanation

The SSL V2 session timeout that is specified in the initialization data for the **gsk_initialize()** routine is not valid.

User response

Specify a valid SSL V2 session timeout value.

104 **SSL V3 session timeout is not valid.**

Explanation

The SSL V3 session timeout that is specified in the initialization data for the **gsk_initialize()** routine is not valid.

User response

Specify a valid SSL V3 session timeout value.

-1 **No SSL cipher specifications.**

Explanation

The client and server cipher specifications do not contain at least one value in common. Client and server cipher specification may be limited depending on which System SSL FMIDs are installed. See Appendix C, "Cipher suite definitions," on page 795 for more information. Server cipher specifications are dependent on the type of algorithms that are used by the server certificate (RSA, DSA, or Diffie-Hellman), which may limit the options available during cipher negotiation. This error can also occur if no SSL protocols are enabled or if all of the enabled protocols have empty cipher specifications.

User response

Ensure that the client and the server have at least one cipher specification in common.

-2 **No certificate received from partner.**

Explanation

The required certificate was not received from the communication partner.

User response

Ensure that the remote application is sending the certificate. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

-3 **Certificate key is not compatible with cipher suite.**

Explanation

The certificate key is not compatible with the negotiated cipher suite. The negotiated cipher suite is dependent on the type of algorithms that are used by the server certificate (RSA, DSA, or Diffie-Hellman) and those available for the client to use. This error can occur if the client certificate uses an algorithm that is incompatible with the server certificate.

User response

Specify a certificate with the appropriate key type.

-5 **SSL V2 header is not valid.**

Explanation

The received message does not start with a valid SSL V2 header. This error can occur if an SSL V3 client attempts to establish a secure connection with an SSL V2 server.

User response

Enable the SSL V2 protocol on the client and then retry the request.

-6 **Certificate format is not supported.**

Explanation

The certificate received from the communication partner is not supported by the current version of the System SSL run time.

User response

Collect a System SSL trace containing a dump that contains the unsupported certificate and then contact your service representative.

-7 **Session renegotiation is not allowed.**

Explanation

An attempt to renegotiate the session parameters for an active connection is rejected. This code occurs if renegotiation is disabled or if the client or server rejects the renegotiation. If using the TLS protocol, and a no renegotiation alert is sent to the peer or received from the peer, then SSL processing continues using the current session parameters. If using the TLS or the SSL V3 protocol, and a handshake failure alert is sent to the peer or received from the peer, then the SSL connection is closed.

User response

If the session parameters are expected to be successfully reset, then the connection must be closed.

-9 **Certificate is revoked.**

Explanation

The certificate is revoked by the certification authority.

User response

Obtain a new certificate.

-10 **Error while reading or writing data.**

Explanation

An I/O error was reported while the System SSL run time was reading or writing data.

User response

Ensure that there are no network errors. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

-11 **SSL message format is incorrect.**

Explanation

An incorrectly formatted SSL message is received from the communication partner.

User response

Collect a System SSL trace containing a dump of the SSL message and then contact your service representative.

-12 **Message authentication code is incorrect.**

Explanation

The message authentication code (MAC) for a message is not correct. This indicates that the message was modified during transmission.

User response

Collect a System SSL trace containing a dump of the message and then contact your service representative if the error persists.

-13 **SSL protocol or certificate type is not supported.**

Explanation

The SSL handshake is not successful because of an unsupported protocol or certificate type. This error can occur if there is no enabled SSL protocol shared by both the client and the server.

User response

Ensure that the SSL protocol you want is enabled on both the client and the server. Collect a System SSL trace containing a dump of the failing handshake and then contact your service representative if the problem persists.

-14 **Certificate signature is incorrect**

Explanation

The certificate signature is not correct for a certificate received from the communication partner.

User response

Ensure that a valid certificate is being sent by the communication partner. Collect a System SSL trace containing a dump of the incorrect certificate and then contact your service representative if the error persists.

-15 **Certificate is not valid**

Explanation

Either the local certificate or the peer certificate is not valid. In order for a certificate to be valid, the complete certificate chain must be present in the key database file, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. Verify that the certificate in the certificate chain is marked trusted.

User response

Ensure that a valid certificate is being sent by the communication partner. Collect a System SSL trace containing a dump of the incorrect certificate and then contact your service representative if the error persists.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate's trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate's profile is generic. A (G) indicates generic.

-16	**SSL protocol violation.**

Explanation

The communication partner violated the SSL protocol by sending a message out of sequence or by omitting a required field from a message.

User response

Collect a System SSL trace and then contact your service representative.

-17	**Permission denied.**

Explanation

The System SSL run time is unable to access a file or system facility.

User response

Ensure that the application is authorized to access the file or facility. Collect a System SSL trace and then contact your service representative if the error persists.

-18	**Self-signed certificate cannot be validated.**

Explanation

A self-signed certificate cannot be validated because it is not in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token.

User response

Add the self-signed certificate to the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate's trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate’s profile is generic. A (G) indicates generic.

-19 **Certification authority is unknown**

Explanation

The key database does not contain a certificate for the certification authority.

User response

Obtain the certificate for the certification authority and add it to the key database. When using a SAF key ring, the CA certificate must be TRUSTed.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate’s trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate’s profile is generic. A (G) indicates generic.

-20 **Insufficient storage is available.**

Explanation

The System SSL runtime library is unable to obtain storage for an internal control block.

User response

Increase the storage available to the application and then retry the failing operation.

-21 **Handle is in the incorrect state.**

Explanation

The SSL connection handle is in the incorrect state for the requested operation.

User response

Correct the application to request SSL functions in the proper sequence.

-22 **Socket closed by remote partner.**

Explanation

The remote partner closed the socket.

User response

None.

-25 **Certificate is expired or is not valid yet.**

Explanation

The current time is either before the certificate start time or after the certificate end time.

User response

Obtain a new certificate if the certificate is expired or wait until the certificate becomes valid if it is not valid yet.

-26 **Key exceeds allowable export size.**

Explanation

The key size used for an export cipher suite exceeds the allowable maximum size. For RSA and DSA keys, the maximum export key size is 512 bits. If the certificate key is larger than 512 bits, the SSL run time uses a temporary 512-bit key for the connection.

User response

Collect a System SSL trace and then contact your service representative.

-27 **Key entry does not contain a private key.**

Explanation

The key entry does not contain a private key or the private key is not usable. This error can also occur if the private key is stored in ICSF and ICSF services are not available, if using a SAF key ring that is owned by another user, if the private key size is greater than the supported configuration limit or the application is executing in FIPS mode. Certificates that are meant to represent a server or client must be connected to a SAF key ring with a USAGE value of PERSONAL and either be owned by the user ID of the application or be SITE certificates. This error can occur when using z/OS PKCS #11 tokens if the user ID of the application does not have appropriate access to the CRYPTOZ class. This error can occur when using private keys associated with user certificates in a SAF key ring owned by another user if the user ID of the application does not have appropriate access to the *ringOwner*.*ringName*.LST resource in the RDATALIB class.

User response

Specify a key entry containing a private key value. Ensure that the ICSF started task is running if the private key is stored in ICSF. When using z/OS PKCS #11 tokens ensure that the user ID has appropriate access to the CRYPTOZ class.

If executing in FIPS mode, ensure that the certificate being used does not have its private key stored in ICSF.

-28 **Function parameter is not valid.**

Explanation

A parameter specified on an SSL function call is not valid.

User response

Ensure that the parameters on the failing function call are correct. Contact your service representative if the error persists.

-29 **An internal error has occurred.**

Explanation

The System SSL runtime library detected an internal processing error.

User response

Retry the operation. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

-30 **Socket request would block.**

Explanation

The socket is in non-blocking mode and the socket request returned the EWOULDBLOCK error.

User response

Retry the **gsk_secure_soc_read()** or **gsk_secure_soc_write()** request when the socket is ready to send or receive data.

-34 **Certificate revocation list cannot be found.**

Explanation

A certificate revocation list (CRL) cannot be found in the specified LDAP server.

User response

Contact the certification authority and obtain the required CRL.

-35 **Certificate validation error.**

Explanation

An error is detected while validating a certificate. This error can occur for the following reasons:

- The root CA certificate is not found in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token.
- The certificate is not marked as a trusted certificate.
- The certificate requires an algorithm or key size that is non-FIPS while executing in FIPS mode.
- The certificate does not validate properly according to the specifications of RFCs 2459, 3280, or 5280. The validation mode is determined by the GSK_CERT_VALIDATION_MODE setting.

User response

The following must be verified depending upon the error that is encountered:

- Verify that the root CA certificate is in the key database, PKCS #12 file, SAF key ring, or z/OS PKCS#11 token and is marked as trusted.
- Check all certificates in the certification chain and verify that they are trusted and are not expired.
- If executing in FIPS mode, check that only FIPS algorithms and key sizes are used by the certificate. For more information, see Chapter 4, “System SSL and FIPS 140-2,” on page 19.
- If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate’s trust chain are available.
- If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.
- If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile

was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate's profile is generic. A (G) indicates generic.

- Verify that the certificates in the peer certificate chain adhere to the RFC specifications of the certificate validation mode. The RFCs indicate the required and optional characteristics of the certificates.

Collect a System SSL trace containing the error and then contact your service representative if the problem persists.

-36 **Cryptographic processing error.**

Explanation

An error is detected by a cryptographic function. This error might also occur while running in FIPS mode when negotiating a secure connection and a non-FIPS key size is used or a triple DES cipher is used and the negotiated triple DES session key does not have three unique key parts.

User response

If the error occurred while executing in FIPS mode, check that only FIPS key sizes are used. If the error occurred during the establishment of a secure connection in FIPS mode using a triple DES cipher, retry the connection.If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

For more information about FIPS key sizes, see Chapter 4, "System SSL and FIPS 140-2".

-37 **ASN processing error.**

Explanation

An error is detected while processing a certificate field.

User response

Collect a System SSL trace containing the error and then contact your service representative.

-38 **LDAP processing error.**

Explanation

An error is detected while setting up the LDAP environment or retrieving an LDAP directory entry.

User response

Ensure that the LDAP server is running and that there are no network errors. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

-39 **LDAP is not available.**

Explanation

The System SSL run time is unable to access the LDAP server.

User response

Ensure that the LDAP server is running and that there are no network problems. Collect a System SSL trace and then contact your service representative if the error persists.

-40 **SSL V2 cipher is not valid.**

Explanation

The SSL V2 cipher is not valid.

User response

Specify a valid cipher.

-41 **SSL V3 cipher is not valid.**

Explanation

The SSL V3 cipher is not valid.

User response

Specify a valid cipher.

-42 **Bad handshake specification.**

Explanation

The handshake specification for the **gsk_secure_soc_init()** routine is not valid.

User response

Specify a valid value for the *hs_type* field in the **gsk_secure_soc_init()** initialization data.

-43 **No read function.**

Explanation

No read function is provided for the **gsk_secure_soc_init()** routine.

User response

Specify a read function for the *skread* field in the **gsk_secure_soc_init()** initialization data.

-44 **No write function.**

Explanation

No write function is provided for the **gsk_secure_soc_init()** routine.

User response

Specify a write function for the *skwrite* field in the **gsk_secure_soc_init()** initialization data.

-46 **Socket write request would block.**

Explanation

A socket write request that is issued as part of an SSL handshake return the EWOULDBLOCK error.

User response

Retry the failing request when the socket is ready to send data.

-47 **Connection is active.**

Explanation

An SSL secure connection operation cannot be completed because of an active request for the connection.

User response

Retry the failing request when the currently active request has completed.

-48 **Connection closed.**

Explanation

For **gsk_secure_soc_read()**, a close notification has been received from the peer application. For **gsk_secure_soc_write()**, a close notification has been sent to the peer application. A close notification is sent when a close notification is received from the peer application. Additional data may not be sent by the application after the close notification has been sent to the peer application.

User response

None.

-51 **Protocol is not SSL V3 or TLS V1.0.**

Explanation

The requested function requires the SSL V3 or TLS V1.0 protocol.

User response

Ensure that the correct protocol is in use before issuing the request.

-53 **Internal error reported by remote partner.**

Explanation

The peer application detected an internal error while performing an SSL operation and sent an alert to close the secure connection.

User response

Check the error log for the remote application to determine the nature of the processing error.

-54 **Unknown alert received from remote partner.**

Explanation

The peer application sent an alert message that is not recognized by the System SSL run time.

User response

Collect a System SSL trace and then contact your service representative.

-55 **Incorrect key usage.**

Explanation

The key usage certificate extension does not permit the requested key operation. This error can occur if the key usage extension of a client or server certificate (if any) does not allow the appropriate key usage.

- RSA server certificates using 40-bit export ciphers with a public key size greater than 512 bits must allow digital signature.

- Diffie-Hellman server certificates using fixed Diffie-Hellman key exchange must allow key agreement.
- Other RSA server certificates must allow key encipherment.
- DSA server certificates using ephemeral Diffie-Hellman key exchange must allow digital signature.
- Client certificates using Diffie-Hellman key exchange must allow key agreement.
- Otherwise client certificates must allow digital signature.

User response

Specify a certificate with the appropriate key usage.

If the **gskkyman** utility was used to create either the client or server end-entity certificate, ensure that the appropriate option was selected from the Certificate Usage menu to create a user or server certificate.

-56 **Multiple certificates exist for label.**

Explanation

Access of certificate/key from label could not be resolved because multiple certificates/keys exist with the label.

User response

Correct certificate/key store so that label specifies a unique record.

If using a PKCS #12 file, use the **gskkyman** command line option **-dc** or **-dcv** to display the contents of the PKCS #12 file. The friendly name or subject distinguished name value is displayed as the label. Ensure that the specified label is unique in the PKCS #12 file.

-57 **Multiple keys are marked as the default.**

Explanation

Access of key from default status could not be resolved because multiple keys are marked as the default key.

User response

Correct certificate/key store so that only one key is marked as the default key.

-70 **SRB processing is not initialized.**

Explanation

The **gsk_srb_initialize()** routine has not been called to initialize the SRB support.

User response

Call **gsk_srb_initialize()** before making any calls to GSKSRBRD or GSKSRBWT.

-71 **SRB lock timeout.**

Explanation

The GSKSRBRD or GSKSRBWT routine is unable to obtain the lock for the SRB control area.

User response

Ensure that the SRB processing threads are not suspended (for example, a synchronous dump suspends thread execution while the dump is processed). Contact your service representative if the error persists.

-72 **SRB suspend failed.**

Explanation

The GSKSRBRD or GSKSRBWT routine is unable to suspend execution while waiting for the completion of the read or write request.

User response

Contact your service representative.

-73 **Unknown SRB service request.**

Explanation

The SRB service task does not recognize the function request.

User response

Contact your service representative.

-99 **An unexpected error has occurred.**

Explanation

An unexpected error is detected by the System SSL run time.

User response

Collect a System SSL trace containing the error and then contact your service representative.

-100 **Buffer size is not valid.**

Explanation

The socket buffer or buffer size is not valid.

User response

Specify a valid buffer and buffer size.

-101 **Handle is not valid.**

Explanation

The SSL connection handle specified on a System SSL function call is not valid.

User response

Call the **gsk_secure_soc_init()** function to create an SSL connection handle.

-104 **Error encountered generating random bytes.**

Explanation

The SSL/TLS handshake encountered an error while generating random bytes.

User response

Retry the secure connection. Contact your service representative if the error persists.

-105 **Key database is not a FIPS mode database.**

Explanation

While executing in FIPS mode, an attempt was made to open a key database that does not meet FIPS criteria.

User response

Specify a key database that meets FIPS criteria if running in FIPS mode.

-106 **Required TLS Renegotiation Indication not received**

Explanation

TLS Renegotiation Indication was not received on the initial handshake with the peer as required by the GSK_EXTENDED_RENEGOTIATION_INDICATOR environment variable. If a server receives this code, then the GSK_EXTENDED_RENEGOTIATION_INDICATOR is set to either SERVER or BOTH and the client did not signal TLS Renegotiation Indication on the initial client hello. If a client receives this code, then the GSK_EXTENDED_RENEGOTIATION_INDICATOR is set to either CLIENT or BOTH and the server did not signal TLS Renegotiation Indication on the initial server hello.

User response

Ensure that the peer is configured to signal TLS Renegotiation Indication. If the peer does not support TLS Renegotiation Indication, and connection is required, then adjust the local setting of the environment variable GSK_EXTENDED_RENEGOTIATION_INDICATOR to "OPTIONAL".

-107 **ICSF services are unavailable**

Explanation

A cryptographic process cannot complete because ICSF callable services are unavailable.

User response

Ensure that ICSF is running and operating correctly.

-108 **ICSF callable service returned an error**

Explanation

An ICSF callable service that is employed to facilitate a cryptographic process returned an error condition. This error can occur if the user ID of the application does not have appropriate access to the RACF CSFSERV class resource profiles.

User response

Ensure that ICSF is operating correctly and that the user ID of the application has appropriate access to the RACF CSFSERV class resource profiles. See Table 5 on page 16 or Table 6 on page 16 for information about resource profiles. Collect a System SSL trace and verify the ICSF return code and reason code relating to the error. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about ICSF return and reason codes. If the problem persists, contact your service representative.

-109 **ICSF PKCS #11 not operating in FIPS mode**

Explanation

While running in FIPS mode, an attempt was made to use ICSF PKCS #11 services that were not operating in FIPS mode.

User response

Ensure that ICSF is configured to run in FIPS mode.

-110 **ICSF PKCS #11 services are disabled**

Explanation

An attempt was made to use ICSF PKCS #11 services, which are disabled because of an ICSF FIPS self test failure.

User response

Stop and restart ICSF. System SSL might need restarting to regain the full hardware benefit from ICSF. Contact your service representative if the error persists.

-111 **ICSF clear key support not available**

Explanation

Unable to generate clear keys or PKCS #11 objects because of the caller's RACF access to CRYPTOZ class resource CLEARKEY.SYSTOK-SESSION-ONLY that does not allow the generation of non-secure (clear) PKCS #11 keys.

User response

Ensure that the user ID of the application has appropriate access to the RACF CRYPTOZ class resource CLEARKEY.SYSTOK-SESSION-ONLY.

-112 **LDAP Response not received within configured time limit**

Explanation

The time limit indicated in the value for GSK_LDAP_RESPONSE_TIMEOUT has been exceeded.

User response

Ensure that the LDAP server is available and able to process LDAP CRL requests. Verify that the value for GSK_LDAP_RESPONSE_TIMEOUT is sufficient to receive a complete response from the LDAP server.

-124 **PKCS #12 file content not valid**

Explanation

When processing the PKCS #12 file, a format error was detected. This can occur if the file is not properly ASN.1 encoded, been modified if transferred, or the PKCS #12 file is not a Version 3 binary file. PKCS #12 Version 1 files and files in Base64 format are not supported.

User response

If the current file is either a PKCS #12 Version 1 file or a Base64 encoded file, it must be replaced with a PKCS #12 Version 3 file. If transferring the file, be sure to transfer the file in binary format. Correct the PKCS #12 file or obtain a new PKCS #12 file. If the problem persists, collect a System SSL Trace containing the error and then contact your service representative.

-125 **Required basic constraints certificate extension is missing**

Explanation

During the establishment of an SSL/TLS secure connection (**gsk_secure_socket_init()** or **gsk_secure_soc_init()** API) with certificate validation processing set to mode ANY or 2459, an intermediate CA certificate was encountered outside of a trusted certificate source (for example, key database file, SAF key ring, or PKCS #11 token), which does not have a basic constraints extension.

User response

Contact the connection partner to determine the certificates being utilized. Once the certificates are identified, either new valid Version 3 certificates can be obtained to replace the intermediate CA certificate or certificates causing the error, or if the usage of the intermediate CA certificates is deemed to be acceptable, a version of the CA certificate or certificates needs to be added to the application's trusted certificate source (for example, key database file, SAF key ring or PKCS #11 token).

If the error persists after adding the certificates, or if the certificates can not be readily obtained, collect a System SSL trace containing the error and then contact your service representative.

-126 **Triple DES key parts are not unique**

Explanation

During the negotiation of a secure connection in FIPS mode, the generated triple DES key did not meet the requirement of having each key part being unique.

User response

Try the connection again.

-127 **Key size is smaller than the minimum size allowed.**

Explanation

During the negotiation of a secure connection, the peer's X.509 end-entity certificate contain a RSA, DSA, DH, or ECC key that is smaller than the minimum size allowed.

User response

Ensure that the key sizes are at least equal to the minimum supported key sizes. The minimum key size value can be modified using attributes GSK_PEER_RSA_MIN_KEY_SIZE, GSK_PEER_DSA_MIN_KEY_SIZE, GSK_PEER_DH_KEY_SIZE, and GSK_PEER_ECC_MIN_KEY_SIZE.

-128 **An inappropriate protocol fallback is detected**

Explanation

The server is enabled for the Signaling Cipher Suite Value (SCSV) support by setting the GSK_SERVER_FALLBACK_SCSV to ON. When the SCSV is present in the client's supported cipher list during a handshake, it indicates that the client is attempting a fallback handshake with an earlier SSL or TLS protocol. If the server's highest supported TLS or SSL protocol level is greater than the client's SSL or TLS protocol level, the handshake fails with this error.

User response

If the server is configured appropriately with the GSK_SERVER_FALLBACK_SCSV enabled, no action is required as the server is appropriately rejecting a client's attempt to do a handshake at an earlier TLS or SSL protocol level.

Ensure that the client is properly enabled for the highest SSL or TLS protocol level that is supported by the server and retry the initial handshake without sending the SCSV to the server.

-129 **Missing required TLS extended master secret extension from remote partner**

Explanation

The TLS server or client encountered a communicating partner that does not support the extended master secret extension. The TLS extended master secret extension is specified as required locally and the communicating partner did not send or provide the extension during the TLS V1.0 handshake.

User response

If running as a client application, the GSK_CLIENT_EXTENDED_MASTER_SECRET option has been set to REQUIRED; however, the remote server partner did not include the extended master secret extension during the TLS V1.0 handshake. Ensure that the GSK_CLIENT_EXTENDED_MASTER_SECRET option is set appropriately for the remote server partners that are being contacted. If the server partner is a z/OS System SSL application, ensure that the GSK_SERVER_EXTENDED_MASTER_SECRET option is set to ON or REQUIRED and it must be running z/OS V2R3 or later and have PTFs for APAR OA60105 (z/OS V2R3 and V2R4) applied and active. If the remote server partner is another SSL/TLS provider, contact the remote server partner vendor to enable support for the extended master secret extension.

If running as a server application, the GSK_SERVER_EXTENDED_MASTER_SECRET option has been set to REQUIRED; however, the remote client partner did not include the extended master secret extension during the TLS V1.0 handshake. Ensure that the GSK_SERVER_EXTENDED_MASTER_SECRET option is set appropriately for the remote client partners that are communicating with this server. If the client partner is a z/OS System SSL application, ensure that the GSK_CLIENT_EXTENDED_MASTER_SECRET option is set to ON or REQUIRED and it must be running z/OS V2R3 or later and have PTFs for APAR OA60105 (z/OS V2R3 and V2R4) applied and active. If the remote client partner is another SSL/TLS provider, contact the remote client partner vendor to enable support for the extended master secret extension.

-130 **Extended master secret extension mismatch detected on cached TLS handshake attempt**

Explanation

The TLS client attempted to use a previously negotiated session with the server that did not originally negotiate the extended master secret extension. On the cached handshake attempt, the remote server indicated that it was able to successfully use the cached session; however, it included the extended master secret extension which is not allowed.

User response

Contact the remote server partner to determine why it is sending the extended master secret extension on a cached TLS handshake. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

-131 **Incorrect extended key usage.**

Explanation

The extended key usage certificate extension does not permit the requested key operation. This error can occur if the extended key usage extension of a client or server certificate restricts the purposes for which the certificate's key can be used.

- Server certificates used for server authentication in 3280 or 5280 validation mode with an extended key usage extension must allow either the serverAuth or the anyExtendedKeyUsage purpose.
- Client certificates used for client authentication in 3280 or 5280 validation mode with an extended key usage extension must allow either the clientAuth or the anyExtendedKeyUsage purpose.

User response

Specify a certificate with the appropriate extended key usage.

ASN.1 status codes (014CExxx)

ASN.1 status codes have the prefix "014CE". These status codes identify ASN.1 encoding and decoding errors.

014CE001 **No more data.**

Explanation

The end of an ASN.1 encoded stream is reached prematurely. This error can occur if an encoded stream is truncated.

User response

Verify that the encoded certificate is not modified. Contact your service representative if the error persists.

014CE002 **Data value overflow.**

Explanation

A decoded data value is too large to be represented as the specified data type.

User response

Contact your service representative.

014CE003 **Length value is not valid.**

Explanation

The length of an encoded item is not valid. This error can occur if an encoded stream is truncated.

User response

Verify that the encoded certificate is not modified. Contact your service representative if the error persists.

014CE004 **Data encoding is not valid.**

Explanation

The encoded data violates the ASN.1 encoding rules.

User response

Contact your service representative.

014CE005 **Parameter is not valid**

Explanation

An application parameter is not valid.

User response

Correct the application to specify valid parameters for the failing function call. Contact your service representative if the error persists.

014CE006 **Insufficient memory is available.**

Explanation

There is not enough memory available to allocate a required control block or data element.

User response

Increase the memory available to the application and then retry the request. Contact your service representative if the error persists.

014CE007 **Indefinite-length encoding is not allowed**

Explanation

An indefinite-length encoding is encountered for a data element that requires a length value.

User response

Contact your service representative.

014CE008 **Data element must be an ASN.1 primitive.**

Explanation

A constructed element is encountered instead of an ASN.1 primitive.

User response

Contact your service representative.

014CE009 **Data element must be constructed.**

Explanation

An ASN.1 primitive is encountered instead of a constructed element.

User response

Contact your service representative.

014CE00A **Data value is not present**

Explanation

An ASN.1 element has no value and does not have a default value.

User response

Contact your service representative.

014CE00B **Indefinite-length encoding is not supported.**

Explanation

Indefinite-length encoding is not supported for the current structure. An X.509 certificate is encoded using ASN.1 DER (Distinguished Encoding Rules) which does not allow the use of indefinite-length encodings.

User response

Contact your service representative.

014CE00C **Unused bit count is not valid**

Explanation

The unused bit count for a bit string must be between 0 and 7.

User response

Contact your service representative if this error occurs while decoding a bit string. Correct the application if this error occurs while encoding a bit string.

014CE00D **Unused bit count is not valid for a segmented bit string.**

Explanation

The unused bit count must be zero for each segment other than the final segment of a bit string.

User response

Contact your service representative.

014CE00E **Data type is not correct.**

Explanation

An unexpected data type is encountered while decoding a data element.

User response

Contact your service representative.

014CE00F **Excess data found at end of data element**

Explanation

There is unprocessed encoded data after decoding a data element.

User response

Contact your service representative.

014CE010 **Required data element is missing.**

Explanation

A required data element is not found when decoding an encoded structure.

User response

Contact your service representative.

014CE011 **Selection is not within the valid range.**

Explanation

The selection for an ASN.1 element is not within the valid range for that element.

User response

Contact your service representative.

014CE012 **No selection found**

Explanation

No selection found for an ASN.1 element.

User response

Contact your service representative.

014CE013 **Syntax already set.**

Explanation

The decoding syntax is already set for an ASN.1 element.

User response

Contact your service representative.

014CE014 **Character string cannot be converted.**

Explanation

A character string cannot be converted to the target code page. This error can occur when a character string contains characters which cannot be represented in the target code page.

User response

Ensure that the character string uses characters which are valid for the target code page. Contact your service representative if the error persists.

014CE015 **Codeset is not allowed**

Explanation

The requested code set is not valid for the current data element.

User response

Contact your service representative.

014CE016 **Attribute value is not valid.**

Explanation

An attribute value is not valid.

User response

Contact your service representative.

014CE017 **Attribute value separator is missing.**

Explanation

An X.500 attribute value separator is missing.

User response

Ensure that the name string is correctly formed. Each attribute consists of an attribute type and an attribute value separated by an equal sign. Contact your service representative if the error persists.

014CE018 **Attribute value is missing**

Explanation

An X.500 attribute value is missing.

User response

Correct the application to specify an attribute for each relative distinguished name.

014CE019 **Object identifier syntax error**

Explanation

The syntax of an object identifier is not valid. The object identifier consists of one or more decimal numbers separated by periods.

User response

Correct the application to specify a valid object identifier.

014CE01A **PKCS #12 version is not correct.**

Explanation

The PKCS #12 version is not correct.

User response

Contact your service representative.

014CE01B **Interval is not valid.**

Explanation

The certificate interval is not valid.

User response

Contact your service representative.

014CE01C **Object identifier element count is not valid**

Explanation

An object identifier must have at least three elements.

User response

Correct the application to provide a valid object identifier.

014CE01D **Incorrect value for the first object identifier element.**

Explanation

The first element of an object identifier must be 0, 1, or 2.

User response

Correct the application to provide a valid object identifier.

014CE01E **Incorrect value for the second object identifier element.**

Explanation

The second element of an object identifier must be between 0 and 39 if the first element is 0 or 1.

User response

Correct the application to provide a valid object identifier.

014CE01F **Unknown attribute identifier.**

Explanation

An unrecognized attribute identifier is encountered while decoding a certificate extension or an X.509 name. As a result, the attribute value cannot be decoded.

User response

Ensure that the name string is correctly formed. Each attribute consists of an attribute type and an attribute value separated by an equal sign. Contact your service representative if the error persists.

014CE020 **Unknown critical certificate extension.**

Explanation

The X.509 certificate contains a critical extension that is not recognized by the System SSL run time. The certificate cannot be processed.

User response

Obtain a new certificate without the unknown critical certificate extension.

014CE021 **X.500 name syntax error.**

Explanation

The syntax of an X.500 distinguished name is not valid. See RFC 2253 (tools.ietf.org/html/rfc2253) for more information about the format of a distinguished name.

User response

Correct the application to specify a valid distinguished name.

014CE022 **Version is not supported.**

Explanation

The version number in a certificate, certificate request, or certificate revocation list is not supported by the current level of System SSL.

User response

Obtain a new certificate, certificate request, or certificate revocation list with a supported version number.

CMS status codes (03353xxx)

Certificate Management Services (CMS) status codes have the prefix "03353". These status codes include informational messages, including errors that require a user response.

03353001 **Insufficient memory is available.**

Explanation

There is not enough memory available to allocate a required control block or data element.

User response

Increase the memory available to the application and then retry the request. Contact your service representative if the error persists.

03353002 **Certificate extension is not supported.**

Explanation

An X.509 certificate extension is either not supported by the current level of the System SSL run time or is not supported by the certificate version. The certificate extension is not processed. If the extension is marked as a critical extension, the X.509 certificate cannot be processed.

User response

Upgrade the System SSL run time if a later software level supports the certificate extension.

03353003 **Cryptographic algorithm is not supported.**

Explanation

An X.509 cryptographic algorithm is not supported by the current level of the System SSL run time. This error can also occur if the current operation does not support the specified cryptographic algorithm. When running in FIPS mode, this error may occur if an attempt is made to use an algorithm that is not supported in FIPS mode.

User response

Ensure that the cryptographic algorithm is supported for the requested operation or that it is supported if executing in FIPS mode. Upgrade the System SSL run time if a later software level supports the cryptographic algorithm.

03353004 **Signature is not correct**

Explanation

The signature is incorrect for an X.509 certificate or certificate revocation list. This usually means that the certificate has been modified since it was signed by the issuing Certificate Authority.

User response

Verify that the certificate has not been modified. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

03353005 **Cryptographic request failed.**

Explanation

A cryptographic request failed with an unexpected error.

User response

Collect a System SSL trace containing the error and then contact your service representative if the error persists.

03353006 **Input/Output request canceled.**

Explanation

An input/output operation is canceled by the user. This can occur if the user cancels a terminal read request by pressing an attention key or by pressing the enter key without entering any data.

User response

None

03353007 **Input/Output request failed.**

Explanation

An input/output operation fails.

User response

Verify that the file or key ring can be accessed and is not damaged. If creating or updating a file, verify that the file system containing the file is not full. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

03353008 **Verification password does not match.**

Explanation

The user is prompted to verify the password by entering it a second time. The user did not enter the same password both times.

User response

Enter the same password when prompted.

03353009 **File or keyring not found**

Explanation

A file or key ring cannot be opened because it is not found.

User response

Verify that the correct name is specified. This value is case-sensitive. Ensure that the case is preserved with your request. Contact your service representative if the error persists.

0335300A **Database is not valid.**

Explanation

The key database or the request database is not valid. This error can occur if the wrong database password is used when opening the database or if the database format is not supported by the current level of the System SSL run time.

User response

Verify that the database has not been modified or truncated. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

0335300B	**Message not found.**

Explanation

The System SSL run time is unable to locate a message in the message catalog.

User response

Verify that the message catalog can be accessed by the application and can be located using the NLSPATH environment variable. Contact your service representative if the error persists.

0335300C	**Handle is not valid.**

Explanation

The handle that is passed to the System SSL run time is not valid. This error can occur if the handle is closed or is not the proper type for the requested function.

User response

Pass a valid handle to the System SSL routine.

0335300D	**Record deleted.**

Explanation

The requested record is deleted.

User response

None

0335300E	**Record not found.**

Explanation

The requested record is not found.

User response

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate's trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate's profile is generic. A (G) indicates generic.

0335300F	**Incorrect database type**

Explanation

The database does not support the requested operation. This error can occur if the database type is not valid. It can also occur if an attempt is made to add a request record to a key database or a key record to a request database.

User response

Specify an operation supported by the database.

03353010 **Database is not open for update.**

Explanation

A request to modify the key or request database cannot be completed because update mode was not requested when the database was opened or an update was requested on a FIPS mode database while in non-FIPS mode.

User response

Request update mode when opening a database for modification.

03353011 **Mutex request failed.**

Explanation

A mutex operation failed.

User response

Contact your service representative.

03353012 **Backup file already exists.**

Explanation

Before updating a database file, the System SSL run time creates a backup file with the same name with ".new" appended to the name. This file is then deleted after the database file has been rewritten. The file is not deleted if an error occurs while rewriting the database file.

User response

Correct the problem that caused the database update to fail. Then copy the backup file to the database file and delete the backup file.

03353013 **Database already exists.**

Explanation

A request to create a new database cannot be completed because the database file already exists.

User response

Choose a different name for the new database or delete the existing database.

03353014 **Record is too big.**

Explanation

A new record cannot be added to the database because it is larger than the database record length.

User response

If using the **gskkyman** utility, use option 4 from the Database Menu to enlarge the database record length. Applications using the System SSL APIs can use the **gsk_change_database_record_length** API to enlarge the database record length.

03353015 **Database password is expired.**

Explanation

The database password is expired.

User response

Change the database password.

03353016 **The password is not correct.**

Explanation

The wrong password is specified for a key database, an encrypted private key, or an import file. This error can also occur if the file has been modified. Also, this error can occur if the key that is being exported is a secure private key in the TKDS and the specified password length is greater than 63 bytes.

User response

Specify the correct password.

03353017 **Access denied.**

Explanation

The database or key ring cannot be opened because the permissions do not allow access by the current user.

User response

Ensure that the user has read/write access to the database if opening the database for update mode; otherwise ensure that the user has read access to the database or key ring.

03353018 **Database is locked for update.**

Explanation

Another process has opened the database in update mode. Only one process may have the database open for update at a time.

User response

Wait until the database has been closed by the other process and then retry the request.

03353019 **Record length is too small.**

Explanation

The database record length is less than the minimum value of 2500.

User response

Specify a record length of 2500 or greater.

0335301A **No private key.**

Explanation

The key entry does not contain a private key or the private key is not usable. This error might also occur if:

- The private key is stored in ICSF, and ICSF services are not available.
- If the private key size is greater than the supported configuration limit or the application is executing in FIPS mode.
- This error can occur when using a SAF key ring if:
  - The key ring is owned by another user.
  - Using a private key that is associated with a user certificate in a SAF key ring that is owned by another user, and if the user ID of the application does not have appropriate access to the ringOwner.ringName.LST resource in the RDATALIB class.
  - Certificates meant to represent a server or client must be connected to a SAF key ring with a USAGE value of PERSONAL, and either owned by the user ID of the application or SITE certificates.
- This error can occur when using z/OS PKCS #11 tokens if:
  - The user ID of the application does not have appropriate access to the CRYPTOZ class.
  - The label name is not valid for a certificate's PKCS #11 TKDS secure key.
  - The PKCS #11 key object does not exist.
  - The certificate's PKCS #11 TKDS secure key algorithm is not supported.
  - Using **gsk_make_enveloped_private_key_msg()** and the PKCS #11 secure key object that is used as input exists in the PKDS instead of the TKDS.

User response

Verify that the ICSF started task is running if the private key is stored in ICSF. Otherwise, repeat the failing request by using a database entry containing a private key. If using z/OS PKCS #11 tokens, ensure that the user ID has appropriate access to the CRYPTOZ class.

If executing in FIPS mode, ensure that the certificate that is being used does not have its private key stored in ICSF.

If using PKCS # 11 tokens:

- Verify that the certificate's PKCS #11 secure key label name is valid within the TKDS.
- Verify that the PKCS #11 TKDS secure key algorithm is supported.
- If you are using **gsk_make_enveloped_private_key_msg()**, verify that the input PKCS #11 key object exists in the TKDS.

0335301B **Record label is not valid.**

Explanation

The record label is not valid. A label may contain letters, numbers, and punctuation. A record label may not be an empty string.

User response

Provide a valid record label.

0335301C **Record label is not unique.**

Explanation

A record label must be unique within a key database file and its associated request database.

User response

Verify the labels already in use by a key database file and its associated request database and provide a unique record label.

0335301D **Record type is not valid.**

Explanation

The database record type is not valid.

User response

Provide a valid database record type.

0335301E **Duplicate certificate.**

Explanation

An attempt is made to add a certificate to a key database but the database already contains the certificate. A certificate is a duplicate if the issuer name and certificate serial number are the same.

User response

Delete the existing certificate before adding the new certificate.

0335301F **Incorrect Base64 encoding.**

Explanation

An encoded stream cannot be decoded because it contains an incorrect Base64 encoding. A Base64 encoding consists of a header line (for example, -----BEGIN CERTIFICATE-----), encoded text, and a footer line (for example, -----END CERTIFICATE-----). The encoded text is encoded using a 64-character subset in groups of 4 characters.

User response

Ensure that the encoded stream has not been truncated or modified. Base64 encoding uses text data and must be in the local code page. Contact your service representative if the error persists.

03353020 **Unrecognized file or message encoding.**

Explanation

A file or message cannot be imported because the format is not recognized.

System SSL supports X.509 DER-encoded certificates, PKCS #7 signed data messages, and PKCS #12 personal information exchange messages for certificate import files. The import file data may be the binary data or the Base64-encoding of the binary data.

System SSL supports PKCS #7 data, encrypted data, signed data, and enveloped data for messages. This error can also occur if the message is not constructed properly.

User response

Ensure that the import file or message has not been modified. A Base64-encoded import file must be converted to the local code page when it is moved to another system while a binary import file must not be modified when it is moved to another system.

If importing a certificate from a Base64 file, the first and last lines contain readable data. The first line in the file contains '-----BEGIN CERTIFICATE-----' and the last line in the file contains '----END CERTIFICATE-----'. If data is not correct, ensure that the file was transferred successfully.

03353021 **Certificate is not yet valid.**

Explanation

The current time is earlier than the beginning of the certificate validity.

User response

Either wait until the certificate is valid or request a new certificate with an earlier starting date from the certification authority.

03353022 **Certificate is expired**

Explanation

The current time is after the end of the certificate validity.

User response

Request a new certificate from the certification authority.

03353023 **Name format is not supported.**

Explanation

An unsupported name format is encountered while validating a certificate.

User response

Contact your service representative.

03353024 **Issuer certificate not found.**

Explanation

An issuer certificate is not found while validating a certificate. This error can occur if the issuer certificate required for a new certificate is not in the key database or if the required issuer certificate is not trusted or has expired.

User response

Ensure that the key database contains the required issuer certificate and that the certificate is marked as trusted. See "Database menu" on page 547 for information about displaying the contents of an external certificate file to verify which issuer certificate is required. Contact your service representative if the error persists.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate's trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class.

The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate's profile is generic. A (G) indicates generic.

03353025 **Certification path is too long.**

Explanation

The certification path length exceeds the maximum that is specified in the certification authority certificate.

User response

Report the problem to the certification authority.

03353026 **Incorrect key usage.**

Explanation

The key usage certificate extension does not permit the requested key operation.

User response

Obtain a certificate, which allows the requested key operation.

03353027 **Issuer is not a certification authority.**

Explanation

The issuer of an X.509 certificate or the certificate used to sign a certificate revocation list is not a certification authority. This indicates that the basic constraints certificate extension in the issuer certificate or the certificate used to sign a certificate revocation list does not contain the certification authority indicator.

User response

Report the problem to the issuer of the certificate. If creating a signed certificate revocation list, ensure that a valid certificate authority is specified in the **gsk_construct_signed_crl()**, **gsk_create_signed_crl()**, or **gsk_create_signed_crl_record()** routines.

03353028 **Export file format is not supported.**

Explanation

The requested export file format is not supported for the specified database record. Certificates can be exported using the DER and PKCS #7 formats. Certificates and keys can be exported using the PKCS #12 formats.

User response

Select an appropriate export file format.

03353029 **Cryptographic algorithm is not available.**

Explanation

An X.509 cryptographic algorithm is not available. Because of government export regulations, strong encryption is not available on the local system.

User response

Select an algorithm that is available.

0335302A **Record type cannot be changed.**

Explanation

The record type cannot be changed when replacing a database record.

User response

Create a new database entry for the record.

0335302B **Subject name cannot be changed.**

Explanation

The subject name cannot be changed when replacing a database record where the database record has no private key or is used as a signing certificate for other user or server certificates.

User response

Create a new database entry for the record.

0335302C **Public key cannot be changed.**

Explanation

The subject public key cannot be changed when replacing a database record.

User response

Create a new database entry for the record.

0335302D **Default key cannot be changed**

Explanation

The default key for the database cannot be changed using the **gsk_replace_record()**routine.

User response

Use the **gsk_set_default_key()** routine to change the default key for the database.

0335302E **Database contains certificates signed by the certificate.**

Explanation

A CA certificate cannot be deleted because the database still contains certificates that were signed using that certificate. A certificate renewal for a signing certificate fails with this error code if the certificates subject name has changed.

User response

Delete all certificates that are signed by the CA certificate before deleting the certificate. To renew a signing certificate with a changed subject name all dependent certificates must be resigned with the new certificate:

- Create certificate renewal requests for each dependent certificate and delete the dependent certificates and keys.
- Receive the new signing certificate.
- Sign any dependent certificate requests with the new signing certificate.
- Receive the signed dependent certificate renewals.

0335302F **Certificate chain is not trusted.**

Explanation

A certification authority (CA) certificate in the certification chain is not trusted.

User response

Set the trust status for the CA certificate if the certificate can be used for authentication purposes.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate's trust chain are available and have not expired.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate’s profile is generic. A (G) indicates generic.

03353030 **Key not supported by encryption or signature algorithm.**

Explanation

The supplied key is not supported by the requested encryption or signature algorithm.

User response

Provide the appropriate key for the encryption or signature algorithm. For example, an RSA key cannot be used to verify that a DSA signature and a DSA key cannot be used to encrypt data. RSASSA-PSS signatures require an RSA key to perform the operation successfully.

03353031 **Signer certificate not found.**

Explanation

A signer certificate is not found while creating or processing a signed message,

User response

Provide a certificate for each signer, including signers of authenticated attributes.

03353032 **Content type is not supported.**

Explanation

An unsupported PKCS #7 content type is encountered.

User response

See the Programming Reference for the failing routine to determine the supported content types.

03353033 **Recipient certificate not found.**

Explanation

A recipient certificate is not found while creating or processing an enveloped message.

User response

Provide at least one recipient certificate.

03353034 **Encryption key size is not supported.**

Explanation

The encryption key size is not supported by the System SSL run time.

User response

See the System SSL information to determine which key sizes are supported. In general, when executing in non-FIPS mode, 40-bit keys and 128-bit keys are supported for RC2 and RC4, 56-bit keys are supported for DES, 168-bit keys are supported for Triple DES, and 128-bit keys and 256-bit keys are supported for AES. RSA keys must be between 512 and 4096 bits, DSS keys must be between 512 and 2048 bits, and Diffie-Hellman keys must be between 512 and 2048 bits.

When executing in FIPS mode with GSK_FIPS_STATE_ON or GSK_FIPS_STATE_LEVEL1 set, 168-bit keys are supported for Triple DES, and 128-bit keys and 256-bit keys are supported for AES. RSA keys must be between 1024 and 4096 bits, DSS keys must be between 1024 and 2048 bits, and Diffie-Hellman keys must be 2048 bits.

When executing in FIPS mode with GSK_FIPS_STATE_LEVEL2 set, 112-bit security is enforced when creating new keys or performing digital signature generation and encryption type operations. Digital signature verification, decryption using Triple DES and RSA decryption with 80-bit key lengths is allowed when processing already protected information. For key generation, DSS keys must be between 1024 and 2048 bits, Diffie-Hellman keys must be 2048 bits, ECC keys must 192 or greater, and RSA keys must be between 2048 and 4096 bits. For verification, DSS keys must be 1024 or 2048 bits, ECC keys must 192 or greater, and RSA keys must be between 1024 and 4096 bits.

When executing in FIPS mode with GSK_FIPS_STATE_LEVEL3 set, 112 bit or higher security strength is enforced as defined in NIST SP800-131Ar1. For key generation, DSS keys must be 2048 bits, Diffie-Hellman keys must be 2048 bits, ECC keys must 224 or greater, and RSA keys must be between 2048 and 4096 bits. For verification, DSS keys must be 2048 bits, ECC keys must 224 or greater, and RSA keys must be between 2048 and 4096 bits.

When using RSASSA-PSS signature algorithm to perform sign or verify operations, only RSA key sizes 2048 through 4096 inclusive are supported.

This error can also occur if the requested key size is not compatible with the supplied key generation parameters. See the System SSL information to determine which key sizes are supported. See Chapter 4, “System SSL and FIPS 140-2,” on page 19 for information about operating in FIPS mode.

03353035 **Encryption key parity is not correct.**

Explanation

DES and Triple DES encryption keys must have odd parity for each key byte.

User response

Verify that the key is generated correctly. Contact your service representative if the error persists.

03353036 **Encryption key is weak.**

Explanation

A small subset of the possible DES and Triple DES encryption keys are weak and can be broken more easily than the rest of the keys. For this reason, the weak keys should be avoided when generating a DES or Triple DES key. The error can also occur while running in FIPS mode with a user supplied Triple DES session key when the key does not contain three unique key parts.

User response

A user supplied Triple DES key was found to be weak or a Triple DES key was specified that did not have three unique key parts. Ensure the key being used is valid and retry the operation. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

03353037 **Initial vector size is not correct.**

Explanation

The initial vector that is used by the encryption routine is not the correct length.

User response

Contact your service representative.

03353038 **Encryption data size is not correct.**

Explanation

The length of the encryption data is not correct. For symmetric key algorithms using cipher block chaining, the encryption data must be a multiple of the cipher block size. For asymmetric key algorithms, the encryption data must be the same length as the cipher key modulus.

User response

Verify that the encryption data has not been truncated. Contact your service representative if the error persists.

03353039 **Encryption block format is not correct.**

Explanation

The encryption block format is not correct following decryption. This error can occur if the wrong key is used to decrypt the block.

User response

Verify that the correct key is being used to decrypt the data. Contact your service representative if the error persists.

0335303A **Number does not have a modular inverse.**

Explanation

The cryptographic support is unable to find an inverse for a number.

User response

Contact your support representative.

0335303B **LDAP processing error.**

Explanation

An error is detected while setting up the LDAP environment or retrieving an LDAP directory entry.

User response

Ensure that the LDAP server is running and that there are no network errors. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

0335303C **LDAP is not available.**

Explanation

The System SSL run time is unable to access the LDAP server.

User response

Ensure that the LDAP server is running and that there are no network problems. Collect a System SSL trace and then contact your service representative if the error persists.

0335303D **Digest data size is not correct.**

Explanation

The length of the digest data is not correct. Digest data size by algorithm is:

- MD2 – 16 bytes
- MD5 – 16 bytes
- SHA-1 – 20 bytes
- SHA-224 – 28 bytes
- SHA-256 – 32 bytes
- SHA-384 – 48 bytes
- SHA-512 – 64 bytes

User response

Verify that the data has not been truncated. Contact your service representative if the error persists.

0335303E **Database name is not valid.**

Explanation

The database file name or SAF key ring name is not valid. The length of the fully-qualified database file name must be between 1 and 251 while the length of the SAF key ring must be between 1 and 237.

User response

Provide a valid database name.

0335303F **Database open failed.**

Explanation

The System SSL run time is unable to open the database file, SAF key ring or z/OS PKCS #11 token.

User response

Verify that the database file, SAF key ring, or z/OS PKCS #11 token exists and is accessible by the application. Collect a System SSL trace and then contact your service representative if the error persists.

03353040 **Self-signed certificate not in database.**

Explanation

A self-signed certificate cannot be validated because it is not in the key database, SAF key ring or z/OS PKCS #11 token.

User response

Add the self-signed certificate to the key database, SAF key ring or z/OS PKCS #11 token.

This code may also occur if the intermediate certificate on the key ring was not marked Trusted.

If using RACF key rings, certificates that are marked as not trusted in the RACF database are not retrieved from the key ring. Ensure that the certificates needed to build the certificate’s trust chain are available.

If using RACF key rings and the DIGTCERT and DIGTRING classes are RACLIST'ed, issue the SETROPTS RACLIST (DIGTCERT, DIGTRING) REFRESH command to refresh the profiles to ensure that the latest changes are available.

If generic profiling checking was enabled for the DIGTCERT class when the certificate was created or added and its issuer's distinguished name contains any generic characters (*, & and %), a generic certificate profile was created. This generic profile processing may cause the certificate not to be read from the key ring. This certificate will need to be removed and added back after turning off generic profile checking for DIGTCERT class. The SEARCH CLASS(DIGTCERT) command can be used to determine if the certificate’s profile is generic. A (G) indicates generic.

03353041 **Certificate is revoked.**

Explanation

A certificate is revoked and cannot be used.

User response

Obtain a new certificate from the certification authority.

03353042 **Issuer name is not valid.**

Explanation

The certificate issuer name must be a non-empty X.509 distinguished name.

User response

Obtain a new certificate with a valid issuer name.

03353043 **Subject name is not valid.**

Explanation

The certificate subject name must be either a non-empty distinguished name or an empty distinguished name with a SubjectAltName certificate extension.

User response

Obtain a new certificate with a valid subject name.

03353044 **Name constraints violated.**

Explanation

The certificate name is not allowed by the certification path name constraints.

User response

Report the problem to the certification authority.

03353045 **No content data.**

Explanation

The PKCS #7 content information does not contain any content data.

User response

Change the application to provide content data for the content information.

03353046 **Version is not supported.**

Explanation

An unsupported version is encountered.

User response

See the Programming Reference for the failing routine to determine the supported versions.

03353047 **Subject name is same as signer name.**

Explanation

A request to create a new certificate cannot be processed because the requested subject name is the same as the subject name of the signing certificate.

User response

Choose a different subject name for the new certificate.

03353048 **Diffie-Hellman group parameters are not valid.**

Explanation

The Diffie-Hellman group parameters are not valid. The subprime Q must be greater than 1 and less than the prime P. The base G must be greater than 1 and less than the prime P. See RFC 2631 (tools.ietf.org/html/rfc2631) for more information about how the Diffie-Hellman parameters are generated.

User response

Verify that the correct parameters are supplied when calling the failing routine. Contact the certification authority if the Diffie-Hellman group parameters are obtained from an X.509 certificate. Otherwise, collect a System SSL trace and then contact your service representative.

03353049 **Diffie-Hellman values are not valid.**

Explanation

The Diffie-Hellman values are not valid. The private value X must be greater than 1 and less than the prime P. The public value Y must be greater than 1 and less than the prime P. In addition, the result of raising the public value Y to the power of the subprime Q modulo the prime P must be equal to 1. See RFC 2631 (tools.ietf.org/html/rfc2631) for more information about how the Diffie-Hellman values are generated.

User response

Contact the certification authority if the Diffie-Hellman values are obtained from an X.509 certificate. Otherwise, collect a System SSL trace and then contact your service representative.

0335304A **Digital Signature Standard parameters are not valid.**

Explanation

The Digital Signature Standard parameters are not valid. The subprime Q must be greater than 1 and less than the prime P. The base G must be greater than 1 and less than the prime P. See FIPS 186-2: Digital Signature Standard (DSS) (csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf) for more information about how the parameters are generated.

User response

Verify that the correct parameters are supplied when calling the failing routine. Contact the certification authority if the DSS parameters are obtained from an X.509 certificate. Otherwise, collect a System SSL trace and then contact your service representative.

0335304B **Certificate not valid for host.**

Explanation

A server certificate does not contain the current host name as either the common name (CN) element of the subject name or as a DNS entry for the subject alternate name.

User response

Obtain a new certificate containing the host name you want.

0335304C **No certificate in import file.**

Explanation

The import file does not contain an X.509 certificate.

User response

Specify a valid certificate import file.

0335304D **The content-type authenticated attribute is not allowed.**

Explanation

The set of authenticated attributes that are supplied within the *attributes_signers* parameter must NOT include the content-type authenticated attribute as this is automatically provided by **gsk_make_signed_data_content_extended()** and **gsk_make_signed_data_msg_extended()**.

User response

Do not include content-type or message-digest in the set of authenticated attributes that are supplied to **gsk_make_signed_data_content_extended()** or **gsk_make_signed_data_msg_extended()**.

0335304E **The message-digest authenticated attribute is not allowed.**

Explanation

The set of authenticated attributes that are supplied from the *attributes_signers* parameter must NOT include the message-digest authenticated attribute as this is automatically provided by **gsk_make_signed_data_content_extended()** and **gsk_make_signed_data_msg_extended()**.

User response

Do not include content-type or message-digest in the set of authenticated attributes that are supplied to **gsk_make_signed_data_content_extended()** or **gsk_make_signed_data_msg_extended()**.

0335304F **Attribute identifier is not valid.**

Explanation

The attribute identifier is not valid.

User response

Specify a valid attribute identifier.

03353050 **Enumeration is not valid.**

Explanation

The enumeration value is not valid.

User response

Specify a valid enumeration value.

03353051 **CA certificate not supplied**

Explanation

A signing CA certificate was not supplied on the call.

User response

Supply a CA certificate on the function call.

03353052 **Validation option is not valid.**

Explanation

The specified validation option is not valid.

User response

Specify a valid validation option.

03353053 **Certificate request not supplied.**

Explanation

A certificate request structure was not supplied on the call.

User response

Supply a certificate request structure on the function call.

03353054 **Public key info not supplied.**

Explanation

A pkcs_public_key_info structure was not supplied on the call.

User response

Supply a pkcs_public_key_info structure on the function call.

03353055 **Modulus bits not supplied.**

Explanation

The number of modulus bits was not supplied on the call.

User response

Supply the number of modulus bits on the function call.

03353056 **Exponent not supplied.**

Explanation

A gsk_buffer structure containing the exponent was not supplied on the call.

User response

Supply a gsk_buffer structure containing the exponent on the function call.

03353057 **Private key info not supplied.**

Explanation

A pkcs_private_key_info structure was not supplied on the call.

User response

Supply a pkcs_private_key_info on the function call.

03353058 **Modulus not supplied.**

Explanation

A gsk_buffer structure containing the modulus for the RSA key was either not supplied on the call or not supplied in the gsk_private_key or gsk_public_key structure.

User response

Ensure that a gsk_buffer structure containing the modulus for the RSA key is supplied on the function call, or is defined in the gsk_private_key or gsk_public_key structure.

03353059 **Public exponent not supplied.**

Explanation

A gsk_buffer structure containing the public exponent for the RSA key was not supplied on the call or not supplied in the gsk_private_key or gsk_public_key structure.

User response

Ensure that a gsk_buffer structure containing the public exponent for the RSA key is supplied on the function call, or is defined in the gsk_private_key or gsk_public_key structure.

0335305A **Private exponent not supplied.**

Explanation

A gsk_buffer structure containing the private exponent for the RSA key was not supplied on the call, or not supplied in the gsk_private_key structure.

User response

Ensure that a gsk_buffer structure containing the private exponent for the RSA key is supplied on the function call, or is defined in the gsk_private_key structure.

0335305B **First prime not supplied.**

Explanation

A gsk_buffer structure containing the first prime for the RSA key was not supplied on the call, or not supplied in the gsk_private_key structure.

User response

Ensure that a gsk_buffer structure containing the first prime exponent for the RSA key is supplied on the function call, or is defined in the gsk_private_key structure.

0335305C **Second prime not supplied.**

Explanation

A gsk_buffer structure containing the second prime for the RSA key was not supplied on the call, or not supplied in the gsk_private_key structure.

User response

Ensure that a gsk_buffer structure containing the second prime for the RSA key is supplied on the function call, or is defined in the gsk_private_key structure.

0335305D **First prime exponent not supplied.**

Explanation

A gsk_buffer structure containing the first prime exponent for the RSA key was not supplied on the call, or not supplied in the gsk_private_key structure.

User response

Ensure that a gsk_buffer structure containing the prime exponent for the RSA key is supplied on the function call, or is defined in the gsk_private_key structure.

0335305E **Second prime exponent not supplied.**

Explanation

A gsk_buffer structure containing the second prime exponent for the RSA key was not supplied on the call, or not supplied in the gsk_private_key structure.

User response

Ensure that a gsk_buffer structure containing the second prime exponent for the RSA key is supplied on the function call, or is defined in the gsk_private_key structure.

0335305F **CRT coefficient not supplied.**

Explanation

A gsk_buffer structure containing the CRT coefficient for the RSA key was not supplied on the call, or not supplied in the gsk_private_key structure.

User response

Ensure that a gsk_buffer structure containing the CRT coefficient for the RSA key is supplied on the function call, or is defined in the gsk_private_key structure.

03353060 **Certificate revocation list cannot be found.**

Explanation

The required certificate revocation list (CRL) cannot be found in the specified LDAP server when the *gsk_crl_security_level* is set to HIGH or the CRL cannot be found in the HTTP server indicated in the CRL distribution points extension and the *gskcms_revocation_security_level* is set to MEDIUM or HIGH.

User response

If contacting an LDAP server to retrieve the CRL, verify that the CRL is present in the LDAP entry being searched and is valid. Verify that the certificate's issuer is the same as the CRL issuer. Contact the certification authority and obtain the required CRL.

If contacting an HTTP server to retrieve the CRL, verify that the CRL is present on the HTTP server. Contact the HTTP server administrator to verify that the CRL is present on the server. If there are crlIssuers present in the CRL distribution point extension, verify that there is at least one match between those and the CRL issuer. If a match cannot be found in the crlIssuers in the CRL distribution point extension or there are no crlIssuers present, verify that the certificate's issuer is the same as the CRL issuer. The HTTP server administrator may need to contact the certification authority to obtain the required CRL.

Collect a System SSL trace containing the error and then contact your service representative if the problem persists.

03353061 **Multiple certificates exist for label.**

Explanation

Access of certificate/key from label could not be resolved because multiple certificates/keys exist with the label.

User response

Correct certificate/key store so that label specifies a unique record.

03353062 **Multiple keys are marked as the default.**

Explanation

Access of key from default status could not be resolved because multiple keys are marked as the default key.

User response

Correct the certificate/key store so that only one key is marked as the default key.

03353064 **Digest type and key type are incompatible.**

Explanation

The specified digest algorithm and the key algorithm are incompatible.

User response

Specify a digest algorithm that is compatible with the signing key algorithm.

03353065 **Generate random bytes input buffer not valid.**

Explanation

The input buffer to gsk_generate_random_bytes is not valid.

User response

Ensure a valid gsk_buffer structure has been supplied to the gsk_generate_random_bytes API. Contact your service representative if the error persists.

03353066 **Generate random bytes produced duplicate output.**

Explanation

The Random Number Generator produced identical consecutive blocks of output data. If in FIPS mode, any further attempts to use System SSL continues to fail until the application is restarted or the executing process is reinitialized.

User response

Restart the SSL application or process to reinitialize the SSL DLLs. If the problem persists, collect a System SSL trace containing the error and contact your service representative.

03353067 **Known Answer Test has failed.**

Explanation

A Known Answer Test failed to match the expected results. Any further attempts to use System SSL continues to fail until the application is restarted or the executing process is reinitialized.

User response

Restart the SSL application or process to reinitialize the SSL DLLs. If the problem persists, collect a System SSL trace containing the error and contact your service representative.

03353068 **API is not supported.**

Explanation

The API is not supported. An attempt was made to use an API that is not supported in the current mode of operation (FIPS or non-FIPS).

User response

Ensure that the API being used is supported in the mode in which the application is executing. If you are invoking a FIPSonly API, you must restart your application in FIPS mode.

03353069 **Key database is not a FIPS mode database.**

Explanation

While executing in FIPS mode, an attempt was made to open a key database that is non-FIPS.

User response

Specify a key database that meets FIPS 140-2 criteria, if running in FIPS mode.

0335306A **Key database can only be opened for update if running in FIPS mode.**

Explanation

While executing in non-FIPS mode, an attempt was made to open a FIPS key database for update.

User response

To open a FIPS key database for update, you must be executing in FIPS mode.

0335306B **Cannot switch from non-FIPS mode to FIPS mode.**

Explanation

While executing in non-FIPS mode, an attempt was made to switch to FIPS mode.

User response

Once executing in non-FIPS mode it is not possible to switch to FIPS mode.

0335306C **Attempt to execute in FIPS mode failed.**

Explanation

A request to execute in FIPS mode failed because the required System SSL DLLs could not be loaded.

User response

Ensure that the Cryptographic Services Security Level 3 FMID is installed and that module verification has been setup correctly. See "System SSL module verification setup" on page 23 for module verification information. Module verification failures may also result in RACF messages (for example, ICH440I) being written to the console with information about the failure. If the module verification problem persists, collect a System SSL trace containing the error and then contact your service representative.

0335306D **Acceptable policy intersection cannot be found.**

Explanation

The Certificate Policies extension of the certificate does not contain an acceptable policy as required by the application or an issuing certificate.

User response

Ensure that the certificate chain is valid and the user certificate is intended to be used for the required purpose.

0335306E **Variable argument count is not valid.**

Explanation

The specified variable argument count is not valid.

User response

Specify a valid variable argument count.

0335306F **Required certificate extension is missing.**

Explanation

A certificate extension that is mandatory for the certificate to be used for the required purpose has not been found.

User response

Ensure that the certificate chain is correct and complies with the validation mode defined for the connection. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

If performing a PKCS #7 operation to encode a signer identifier, ensure that the certificate has a subject key identifier extension.

03353070 **Certificate extension data is incorrect.**

Explanation

A certificate extension has incorrect data or has a necessary field missing.

User response

Ensure that the certificate chain is correct and complies with the validation mode defined for the connection. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

03353071 **Certificate extension data has an incorrect critical indicator.**

Explanation

A critical indicator for a certificate extension is incorrect. Either the extension is required to be marked critical and is marked non-critical or is required to be marked non-critical and is marked critical.

User response

Ensure that the certificate chain is correct and complies with the validation mode defined for the connection. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

03353072 **Certificate contains a duplicate extension.**

Explanation

The certificate or CRL undergoing validation contains multiple certificates or CRL extensions of the same type.

User response

Ensure that the certificate chain is correct and complies with the validation mode defined for the connection. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

03353073 **Cannot match CRL distribution points.**

Explanation

The DN in the Issuing Distribution Point extension of the CRL does not match a suitable DN in the certificate undergoing validation. The DN in the Issuing Distribution Point extension must match either:

- A DN of type fullName in the Distribution Point of the CRL Distribution Points extension of the certificate undergoing validation
- The CRLIssuer field in the Distribution Point of the CRL Distribution Points extension of the certificate undergoing validation
- The Certificate Issuer name, if no CRL Distribution Point extension exists in the certificate undergoing validation

User response

Ensure that the certificate chain is correct and complies with the validation mode defined for the connection. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

03353074 **FIPS mode key generation failed pair-wise consistency check.**

Explanation

While executing in FIPS mode, a key pair was generated that failed a pair-wise consistency check. Any further attempts to use System SSL continues to fail until the application is restarted or the executing process is reinitialized.

User response

Restart the SSL application or process to reinitialize the SSL DLLs. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

03353076 **Prime not supplied.**

Explanation

A gsk_buffer structure containing the prime for the DSA or Diffie-Hellman key was not supplied in the gsk_private_key or gsk_public_key structure.

User response

Ensure that the prime value for the DSA or Diffie-Hellman key is defined in the gsk_private_key or gsk_public_key structure.

03353077 **Subprime not supplied.**

Explanation

A gsk_buffer structure containing the sub_prime for the DSA key was not supplied in the gsk_private_key or gsk_public_key structure.

User response

Ensure that the sub prime value for the DSA key is defined in the gsk_private_key or gsk_public_key structure.

03353078 **Base not supplied.**

Explanation

A gsk_buffer structure containing the base for the DSA or Diffie-Hellman key was not supplied in the gsk_private_key or gsk_public_key structure.

User response

Ensure that the base value for the DSA or Diffie-Hellman key is defined in the gsk_private_key or gsk_public_key structure.

03353079 **Private value not supplied.**

Explanation

A gsk_buffer structure containing the private value for the DSA or Diffie-Hellman key was not supplied in the gsk_private_key structure.

User response

Ensure that the private value for the DSA or Diffie-Hellman key is defined in the gsk_private_key structure.

0335307A **Public value not supplied.**

Explanation

A gsk_buffer structure containing the public value for the DSA or Diffie-Hellman key was not supplied in the gsk_public_key structure.

User response

Ensure that the public value for the DSA or Diffie-Hellman key is defined in the gsk_public_key structure.

0335307B **Private key structure not supplied.**

Explanation

The structure containing the private key components was not supplied on the call.

User response

Supply the structure containing the private key components on the function call.

0335307C **Public key structure not supplied.**

Explanation

The structure containing the public key components was not supplied on the call.

User response

Supply the structure containing the public key components on the function call.

0335307D **Size specified for supplied structure is too small.**

Explanation

The value of the size field in the structure indicates that the size of the structure is insufficient.

User response

Ensure that the size field in the structure being used is initialized to the size of structure.

0335307E **Elliptic Curve is not supported.**

Explanation

The elliptic curve domain parameters that are defined for the elliptic curve public or private key are not supported.

User response

Ensure the elliptic curve public/private key pair uses a supported elliptic curve. See Chapter 3, "Using cryptographic features with System SSL," on page 9 for the list of elliptic curves that are supported by System SSL.

0335307F **EC Parameters not supplied.**

Explanation

A gsk_buffer structure containing the EC domain parameters was not supplied on the call.

User response

Supply a gsk_buffer structure containing the EC domain parameters on the function call.

03353080 **Signature not supplied.**

Explanation

A gsk_buffer structure containing the signature was not supplied on the call.

User response

Supply a gsk_buffer structure containing the signature on the function call.

03353081 **Elliptic Curve parameters are not valid.**

Explanation

The EC domain parameters that are defined for the elliptic curve public or private key are not valid. Either no parameters could be found or the parameters could not be successfully decoded.

User response

Ensure the elliptic curve public/private key pair uses a valid elliptic curve.

03353082 **Elliptic Curve not supported in FIPS mode.**

Explanation

The EC domain parameters that are defined for the elliptic curve public or private key are not approved in FIPS mode.

User response

Ensure the elliptic curve for the public or private key is valid in FIPS mode. See Chapter 4, "System SSL and FIPS 140-2," on page 19 for a list of elliptic curves that are supported by System SSL when running in FIPS mode.

03353083 **ICSF services are unavailable.**

Explanation

A cryptographic process cannot be completed because of ICSF callable services being unavailable.

User response

Ensure that ICSF is running and operating correctly.

03353084 **ICSF callable service returned an error.**

Explanation

Ensure that ICSF is operating correctly and if access to the ICSF callable services are protected with CSFSERV class profiles that the user ID of the application has READ access to the profiles protecting the ICSF callable services. See Table 5 on page 16or Table 6 on page 16 for information about the required resource profile access. If the problem persists, collect a System SSL trace and contact your service representative.

User response

Ensure that ICSF is operating correctly and that the user ID of the application has appropriate access to the CSFSERV class RACF resource profiles. See Table 5 on page 16 or Table 6 on page 16 for information about required resource profile access. Collect a System SSL trace and verify the ICSF return code and reason code relating to the error. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about ICSF return and reason codes. If the problem persists contact your service representative.

03353085 **ICSF PKCS #11 not operating in FIPS mode.**

Explanation

While running in FIPS mode, an attempt was made to use ICSF PKCS #11 services, which were not operating in FIPS mode.

User response

Ensure that ICSF is configured to run in FIPS mode.

03353086 **Incorrect key algorithm.**

Explanation

A supplied key uses an algorithm type that is not suitable for the requested function. This error can occur if a non-ECC key has been supplied to an ECC related function, or if incompatible keys are supplied for certificate creation, such as a certificate containing a Diffie-Hellman key to be signed with an ECDSA key.

User response

Ensure the key supplied uses a suitable key algorithm type. Collect a System SSL trace containing the error to verify the key algorithms. Contact your service representative if the error persists.

03353087 **Certificate revocation list is expired.**

Explanation

The current time is after the nextUpdate time specified in the CRL.

User response

Obtain the latest copy of the CRL from the certification authority.

03353088 **Cryptographic hardware does not support service or algorithm.**

Explanation

A call requiring cryptographic hardware was made to ICSF. The current installation hardware does not support the service or algorithm being used. The cryptographic hardware may not have the required level of support to perform an RSASSA-PSS signing. For more information about the cryptographic hardware support for RSASSA-PSS, see “RSASSA-PSS signature support” on page 14.

User response

Ensure that the correct protocol is in use for your installation or that cryptographic hardware that is required for this service or algorithm is available to ICSF.

03353089 **ICSF PKCS #11 services are disabled.**

Explanation

An attempt was made to use ICSF PKCS #11 services, which are disabled because of an ICSF FIPS self-test failure.

User response

Stop and restart ICSF. System SSL may need restarting to regain the full hardware benefit from ICSF. Contact your service representative if the error persists.

0335308A **Known Answer Test has failed when attempting to use ICSF.**

Explanation

A Known Answer Test failed because of ICSF returning an error. Any further attempts to use System SSL continues to fail until the application is restarted or the executing process is reinitialized.

User response

Ensure that ICSF is running and operating correctly and that the user ID of the application has appropriate access to the CSFSERV class RACF resource profiles. See Table 5 on page 16 for information about required resource profile access. Collect a System SSL trace and verify the ICSF return code and reason code relating to the error. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for more information about ICSF return and reason codes. If the problem persists contact your service representative.

0335308B **Variable argument validate root is not valid.**

Explanation

The specified variable argument validate root is not valid.

User response

Specify a valid variable argument validate root.

0335308C **PKCS #11 label name not valid.**

Explanation

The PKCS #11 secure key label name is not valid. This might be because the label is NULL, an empty string, or has only an equal sign (=).

User response

Verify that input label is correct.

0335308D **Incorrect key attribute.**

Explanation

One or more PKCS #11 attributes or parameters for a key are missing or incorrect for a requested function that is being performed. For example, a signing operation requires that for the key that is being used, the PKCS #11 sign attribute is to be TRUE. Verify that the correct key is being used for the requested function, and that all required attributes are set for that key. If using **gsk_make_enveloped_private_key_msg()**, ensure that a recipient certificate's RSA public key is valid.

User response

Verify that a certificate's PKCS #11 key attributes are correct for the function that is being performed.

0335308E **PKCS #11 object was not found.**

Explanation

PKCS #11 token, token object, or session object are not found.

User response

Verify that a PKCS #11 token or token object is in the TKDS data set. Also, verify that the session object is not lost because of ICSF restarting after the object is created.

0335308F **An algorithm or key size is not FIPS approved for an ICSF operation.**

Explanation

ICSF is in FIPS mode. A call to ICSF for cryptographic or signing support failed because the input key algorithm or size is not supported in FIPS mode. For example, an RSA key size of 512 is not supported in FIPS mode.

User response

Verify that the certificate key that is being used is a supported algorithm and size when ICSF is in FIPS mode. See Algorithm support: FIPS and non-FIPS for more information about supported algorithms and key sizes.

03353090 **PKCS #11 key cannot be extracted.**

Explanation

An attempt to export a PKCS #11 secure key failed because PKCS #11 attribute CKA_EXTRACTABLE is set to CK_FALSE.

User response

Verify that input label is correct. If it is correct, then the key cannot be exported.

03353093 **Clear key support not available due to ICSF key policy.**

Explanation

Unable to generate clear keys or PKCS #11 objects because of the caller's RACF access to CRYPTOZ class resource CLEARKEY.SYSTOK-SESSION-ONLY or CLEARKEY.token_name not permitting the generation of non-secure (clear) PKCS #11 keys.

User response

Ensure that the user ID of the application has appropriate access to the RACF CRYPTOZ class resource CLEARKEY.SYSTOK-SESSION-ONLY. If using **gskkyman**, ensure issuer also has access to resource CLEARKEY.token_name. token_name is the name of the PKCS #11 token that is being managed by **gskkyman**.

03353094 **OCSP responder requires a signed request**

Explanation

The OCSP responder contacted for certificate validation requires that all OCSP requests be signed.

User response

Enable OCSP request signing by specifying a valid database handle (ocspDbHandle), label of signing certificate (ocspReqlabel), and the signature algorithm (ocspReqSignatureAlgorithm) within the OCSP data source structure.

03353095 **HTTP response is not valid**

Explanation

The HTTP response received was not properly formatted or contents are not valid. The HTTP response received must be an HTTP/1.0 or HTTP/1.1 response.

User response

Ensure that the HTTP server is running, that there are no network errors, and the HTTP server sends its responses using HTTP/1.0 or HTTP/1.1 protocols. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

03353096 **OCSP response is not valid**

Explanation

The OCSP ASN.1 encoded response received from the OCSP responder was not properly formatted or its contents are not valid.

User response

Ensure that the OCSP responder server is properly encoding the OCSP response, that it is running, and that there are no network errors. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

03353097 **OCSP request failed with internal responder error**

Explanation

The OCSP responder contacted for certificate validation returned an internal error.

User response

Ensure that the OCSP responder server is running and that there are no network errors. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

03353098 **OCSP response is expired**

Explanation

The current time is after the OCSP response expiration time.

User response

If using the dedicated OCSP responder, ensure that the OCSP responder server is using the most recent revocation information available from the certification authority. If certificate revocation through the AIA extension is enabled, ensure that the OCSP responder servers referenced in the certificate chain are using the most recent revocation information available. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

03353099 **Numeric value is not valid**

Explanation

A numeric value specified as either a parameter to or as a field within the data structure is not valid. If using the **gsk_construct_signed_crl()** routine, the *this_update* time must be less than the CRL expiration time.

User response

Verify that all numeric values specified contain numeric values that are within acceptable ranges. If using the **gsk_construct_signed_crl()** routine, specify a *this_update* time that is less than the CRL expiration time. The CRL expiration time is the earlier of the *next_update* time or the signing certificate's expiration time.

0335309A **A PKCS #7 CMS Version is not supported**

Explanation

The PKCS #7 CMS version is not supported.

User response

Verify that the PKCS #7 CMS version is supported by System SSL.

0335309B **Input certificate not supplied**

Explanation

The required input field, certificate, is missing.

User response

Verify that the input certificate is not NULL.

0335309C **Error creating OCSP request**

Explanation

An internal error was encountered while creating the OCSP request to send to an OCSP responder.

User response

If OCSP request signing is enabled, verify that the signing certificate resides in the handle returned from **gsk_open_keyring()** or **gsk_open_database()** and the signing certificate is valid (start time is before the current time and is not yet expired) and contains a private key. The handle is provided through the *ocspDbHandle* field within the *gskdb_ocsp_source* structure provided on the **gsk_create_revocation_source()** routine. The signing key label is provided through the *ocspReqLabel* field within the *gskdb_ocsp_source* structure provided on the **gsk_create_revocation_source()** routine.

Collect a System SSL trace containing the error and then contact your service representative if the error persists.

0335309D **Maximum response size exceeded**

Explanation

When attempting to retrieve revocation information, the HTTP response exceeded the maximum configured response size for either an OCSP response or a HTTP CRL. The response size is provided through either the *ocspMaxResponseSize* or *httpCdpMaxResponseSize* fields within the *gskdb_ocsp_source* or *gskdb_cdp_source* structures provided on the **gsk_create_revocation_source()** routine.

User response

Ensure that the HTTP response maximum size is adequate for the size of the CRLs or OCSP responses that are being retrieved. If necessary, increase the maximum response size until an adequate size is provided to handle the CRLs or OCSP responses that are being retrieved. If unable to determine an adequate size, collect a System SSL trace containing the error and then contact your service representative.

0335309E **HTTP server communication error**

Explanation

Unable to establish a connection to contact the HTTP server or the OCSP responder to retrieve certificate revocation information.

User response

If enabled for OCSP and a dedicated OCSP responder is enabled, ensure that the responder is running and can be accessed.

If enabled for OCSP responders identified in the certificate AIA extension, ensure that the OCSP responders specified in the extension are running and can be accessed.

If HTTP CRL support is enabled, ensure that the HTTP server specified in the CRL Distribution Point extension is running and can be accessed.

If there is a firewall in place and either an HTTP proxy server or port or an OCSP proxy server or port has been identified, ensure that the servers and ports settings are correct and the servers can be accessed. The proxy server and ports are specified through the *gskdb_ocsp_source* and *gskdb_cdp_source* structures provided on the call to the **gsk_create_revocation_source()** routine.

Collect a System SSL trace containing the error and then contact your service representative if the error persists.

0335309F **Variable argument security level is not valid**

Explanation

The specified variable argument security level is not valid.

User response

Specify a valid variable argument security level.

033530A0 **Extended key usage input count is not valid**

Explanation

The extended key usage input count must be at least 1.

User response

Verify that the extended key usage count is equal to the number of extended key usages provided. The count may not be less than 1.

033530A1 **Extended key usage input is not supplied**

Explanation

The extended key usage structure x509_key_usages is NULL or the x509_key_purpose structure within x509_key_usages is NULL.

User response

Verify that x509_key_usages is not NULL and that x509_key_purpose is not NULL within x509_key_usages.

033530A2 **Extended key usage comparison failed**

Explanation

The extended key usages comparison failed. The user-supplied validation parameter and extended key purpose list did not compare favorably to the certificate's extended key usage extension values.

User response

The extended key usage comparison failed. Verify that the desired validation option is correct for your use.

033530A3 **Extended key usage type is not supported for this operation**

Explanation

The extended key usage type should be a value defined within x509_purpose_type. Purpose types *x509_purpose_unknown* and *x509_purpose_maximum* are invalid. Use special case **GSKCMS_VALIDATE_EXTENDED_KEY_USAGE_CHECK_OID** only if OID comparison is required.

User response

Verify that the input x509_purpose_type is defined within the x509_purpose_type and is not *x509_purpose_unknown*, *x509_purpose_maximum*, or special case **GSKCMS_VALIDATE_EXTENDED_KEY_USAGE_CHECK_OID**.

033530A4 **Certificate does not have an extended key usage extension**

Explanation

The specified certificate does not have an extended key usage extension.

User response

Verify that the input x509_certificate has an extended key usage extension defined.

033530A5 **Nonce in OCSP response does not match value in OCSP request**

Explanation

When validating the nonce in the OCSP response, the value did not match the value sent in the OCSP request.

User response

If OCSP is enabled for the dedicated OCSP responder, ensure that the OCSP responder server is configured to send a nonce in OCSP responses.

Ensure that nonce checking is required. If not required, set *ocspCheckNonce* to FALSE within the *gskdb_ocsp_source* structure provided on the **gsk_create_revocation_source()** routine.

Collect a System SSL trace containing the error and then contact your service representative if the error persists.

033530A6 **OCSP response not received within configured time limit**

Explanation

The time limit indicated in the value for *ocspResponseTimeout* in the OCSP data source has been exceeded.

User response

Ensure that the HTTP server where the OCSP responder resides is available and able to process OCSP requests. Verify that the value for *ocspResponseTimeout* in the OCSP data source is sufficient to receive a complete response from the HTTP server containing the OCSP responder.

033530A7 **Revocation information is not yet valid**

Explanation

The current time is earlier than the validity period of the revocation information provided though either an OCSP response or CRL.

User response

Ensure that the system time is configured correctly. Collect a System SSL trace containing the error and then contact your service representative if the problem persists.

033530A8 **HTTP server host name is not valid**

Explanation

The URI value in the AIA extension or the CDP extension is not in the correct format or cannot be resolved by the Domain Name Service (DNS). The correct URI format is http://hostname[:portNumber].

User response

If an OCSP data source has been provided and the *ocspEnable* parameter is enabled, verify that the certificate being verified has a URI value in the AIA extension that is properly formatted and can be resolved by the DNS. It may be necessary to obtain a new certificate or to specify the *ocspProxyServerName* and *ocspProxyServerPort* parameters if there is a need to pass through a firewall. If the *ocspURL* or *ocspProxyServerName* parameters are specified, verify that the host name and the IP address is properly formatted and can be resolved by the DNS.

If an CDP data source has been provided and the *cdpEnableFlags* parameter is enabled for HTTP URIs, verify that the certificate being verified has a URI value in the CDP extension that is properly formatted and can be resolved by the DNS. It may be necessary to obtain a new certificate or to specify the *httpCdpProxyServerName* and *httpCdpProxyServerPort* parameters if there is a need to pass through a firewall.

033530A9 **An internal error has occurred.**

Explanation

The System SSL runtime library detected an internal processing error.

User response

Retry the operation. If the problem persists, collect a System SSL trace containing the error and then contact your service representative.

033530AA **Required basic constraints certificate extension is missing**

Explanation

During the certificate validation processing for mode RFC 2459 or ANY, an intermediate CA certificate was encountered outside of a trusted certificate source which does not have a basic constraints extension.

User response

The specified untrusted certificate sources that was passed on either the **gsk_validate_certificate_mode()** or **gsk_validate_certificate()** API needs to be examined to determine which intermediate CA certificate is being disallowed. Either the intermediate CA certificate should be replaced with a valid Version 3 certificate or if the usage of the CA certificate is acceptable, the certificate needs to be moved to a trusted certificate source and removed from the untrusted certificate source.

If the error persists after adding the certificates, or if the certificates can not be readily obtained, collect a System SSL trace containing the error and then contact your service representative.

033530AB **PKCS #12 input certificate has no subject DN or friendly name**

Explanation

When reading the certificates of the specified PKCS #12 file, a certificate was encountered that had no subject distinguished name or PKCS #12 friendly name.

User response

Verify that all certificates within the provided PKCS #12 file have either a subject distinguished name or a friendly name attribute. The friendly name attribute or the subject distinguished name is used to create the certificate's label.

033530AC **PKCS #12 file name may not end with .kdb, .rdb or .sth**

Explanation

A PKCS #12 file name cannot end with .kdb, .rdb or .sth.

User response

Verify that the PKCS #12 file name does not end with .kdb, .rdb or .sth. If it does, it needs to be renamed.

033530AD **Required parameter is not set**

Explanation

One or more required parameters in the *gskdb_ocsp_source*, *gskdb_cdp_source*, or *gskdb_extended_directory_source* structures within the *gskdb_source* structure are not correctly set.

User response

If using the **gsk_create_revocation_source()** routine to create an OCSP data source, the *ocspEnable* parameter must be set to TRUE or the *ocspURL* parameter must be specified. If *ocspReqLabel* is set, the *ocspDbHandle* must be specified. The *ocspDbHandle* is the database handle returned by the **gsk_open_database()** routine or the **gsk_open_keyring()** routine that contains the name of the certificate specified in the *ocspReqLabel* parameter.

If using the **gsk_create_revocation_source()** routine to create an CDP data source, the *httpCdpEnableFlags* must be set to either GSKCMS_CDP_ENABLE_HTTP or GSKCMS_CDP_ENABLE_ALL.

If using the **gsk_create_revocation_source()** routine to create an LDAP extended directory source, the *ldapServerName* parameter must be specified.

033530AE **Maximum number of locations allowed to be contacted during certificate validation has been reached**

Explanation

The number of locations allowed by either the *max_source_rev_ext_loc_values* or *max_validation_rev_ext_loc_values* parameters to **gsk_validate_certificate_mode()** has been exceeded. The locations for revocation information are specified by the accessLocation in the AIA certificate extension for OCSP and the distributionPoint in the CDP extension for HTTP CRLs.

User response

Use the values in the certificate chain being validated to determine the proper value for the *max_source_rev_ext_loc_values* parameter, the *max_validation_rev_ext_loc_values* parameter, or both. The value for *max_source_rev_ext_loc_values* must be greater than or equal to the maximum number of location values in a certificate CDP or AIA extensions. The value for *max_validation_rev_ext_loc_values* must be greater than or equal to the total number of location values in all CDP and AIA extensions used in the certificate chain being validated. Collect a System SSL trace containing the error and then contact your service representative if the problem persists.

033530AF **HTTP response not received within configured time limit**

Explanation

The time limit indicated in the value for *httpCdpResponseTimeout* in the CDP data source has been exceeded.

User response

Ensure that the HTTP server is available and able to process HTTP CRL requests. Verify that the value for *httpCdpResponseTimeout* in the CDP data source is sufficient to receive a complete response from the HTTP server.

033530B0 **LDAP response not received within configured time limit**

Explanation

The time limit indicated in the value for *ldapResponseTimeout* in the extended directory data source or the time out value specified for the basic directory data source has been exceeded. The basic directory data source time out is specified on the **gsk_set_directory_numeric_value()** routine

User response

Ensure that the LDAP server is available and able to process LDAP CRL requests. Verify that the time out value for the basic or extended directory data source is sufficient to receive a complete response from the LDAP server.

033530B1 **An unknown error has occurred**

Explanation

System SSL has detected an unknown processing error.

User response

Collect a System SSL trace containing the error and then contact your service representative.

033530B2 **OCSP request failed with try later error**

Explanation

The OCSP responder is unable to currently process the OCSP request.

User response

Contact the OCSP responder administrator to verify that the OCSP responder is working properly. Then retry the OCSP request at a later time.

033530B3 **Signature algorithm pairs list is not valid.**

Explanation

The supported signature algorithm pairs list is not correctly formatted.

User response

Ensure the value that is supplied for the OCSP response signature algorithm pairs in the *ocspResponseSigAlgPairs* parameter within the *gskdb_ocsp_source* structure contains only valid entries for hash and signature algorithm pairs that are supported by System SSL and that each entry is defined using 4 characters. See Table 31 on page 805 for a list of valid 4-character signature algorithm pair definitions.

033530B4 **OCSP response signature algorithm not in signature algorithm pairs list.**

Explanation

The OCSP response was signed with an algorithm that was not specified in the OCSP response signature algorithm pairs list.

User response

Verify that the signature algorithms included in the response signature algorithm pairs list (the *ocspResponseSigAlgPairs* parameter within the *gskdb_ocsp_source* structure) are supported by the OCSP responder and are valid based on the certificate being validated. For example, the OCSP responder may ignore an signature algorithm of SHA-224 with RSA encryption if the certificate being validated is an ECDSA certificate. Ensure that the OCSP responder supports the OCSP preferred signature algorithms extension. The OCSP response signature algorithm pairs list may need to be updated to include the algorithm that the OCSP responder is using to sign the OCSP response. See Table 31 on page 805 for a list of valid 4-character signature algorithm pair definitions. Collect a System SSL trace containing the error and then contact your service representative if the error persists.

033530B5 **Cannot switch from one FIPS mode level to another FIPS mode level**

Explanation

While executing in FIPS mode (GSK_FIPS_STATE_ON, GSK_FIPS_STATE_LEVEL1, GSK_FIPS_STATE_LEVEL2, or GSK_FIPS_STATE_LEVEL3), an attempt was made to switch to another FIPS mode LEVEL. This is not supported. Only the switch to GSK_FIPS_STATE_OFF is supported.

User response

A FIPS mode LEVEL has already been defined and the only FIPS mode setting change that is allowed is to GSK_FIPS_STATE_OFF. See Chapter 4, "System SSL and FIPS 140-2," on page 19 for additional information about operating in FIPS mode.

033530B6 **OCSP request signature algorithm pair is not valid.**

Explanation

The OCSP request signature algorithm pair specified in the *ocspReqSignatureAlgorithm* parameter of the *gskdb_ocsp_source* structure is not valid. The valid values for the OCSP request signature algorithm pair definitions can be found in Table 19 on page 252.

User response

Correct the OCSP request signing algorithm that is specified in the *ocspReqSignatureAlgorithm* parameter of the *gskdb_ocsp_source* structure to a valid one.

033530B7 **OCSP response does not contain requested certificate status.**

Explanation

The OCSP response from the OCSP responder does not contain the requested certificate status.

User response

Contact the OCSP responder administrator to verify that the OCSP responder is operating as expected. If the error persists, collect a System SSL trace containing the error and then contact your service representative.

033530B8 **OCSP response contains duplicate certificate statuses.**

Explanation

The OCSP response from the OCSP responder contains duplicate certificate statuses and it is not possible to determine the revocation status of the requested certificate.

User response

Contact the OCSP responder administrator to verify that the OCSP responder is operating as expected. If the error persists, collect a System SSL trace containing the error and then contact your service representative.

033530B9 **Triple DES key parts are not unique**

Explanation

While executing in FIPS mode, a triple DES key was found to not have three unique key parts. This can occur with a user supplied session key or when generating a triple DES key.

User response

A user supplied triple DES key was found to not have three unique parts. Ensure the supplied key is valid.

033530BA **Certificate does not meet Suite B requirements.**

Explanation

The certificate in use does not meet the requirements for the Suite B profile that is selected for the environment.

User response

Ensure that the certificate used satisfies the requirements for the chosen Suite B profile. See "Suite B cryptography support" on page 49 for more information about Suite B certificate requirements.

033530BB **Certificate database is not supported**

Explanation

Verify that the key database being used is supported. If using **gskkyman** to display a GSKIT CMS V4 key database, the **gskkyman** command line mode must be used.

User response

Access to a GSKIT CMS V4 certificate key database is only allowed by the **gskkyman** utility in command line mode or when creating an SSL/TLS connection.

033530BC **Time value is not valid**

Explanation

The specified *timeVal* structure contains a sub field value that is not valid. The year of the *timeVal* that is passed in must be 1970 - 2106 and is the actual year minus 1900. Therefore:

- *tm_year* must be 70 - 206.
- *tm_mon* must be 0 - 11.
- *tm_day* must be 1 - 31.
- *tm_hour* must be 0 - 23.
- *tm_min* must be 0 - 59.
- *tm_sec* must be 0 - 59.

User response

Specify a valid time value.

033530BD **Output parameter is not valid**

Explanation

An output parameter specified is null or has an invalid value.

User response

Specify a valid output parameter.

033530BE **Format is not valid**

Explanation

The requested certificate revocation list output format is not valid for encoding a certificate revocation list. Certificate revocation lists can be encoded using base64 encoding or DER encoding formats.

User response

Specify an appropriate certificate revocation list encoding format.

033530BF **Expiration date exceeds February 6, 2106 at 23:59:59 UTC**

Explanation

System SSL does not support expiration dates that exceed February 6, 2106, at 23:59:59 UTC.

User response

Ensure that the expiration date does not extend past February 6, 2106, at 23:59:59 UTC.

033530C0 **RSASSA-PSS digest algorithm is not supported**

Explanation

The RSASSA-PSS cryptographic digest algorithm is not supported by the operation or the current level of the System SSL run time. When running in FIPS mode, this error may occur if an attempt is made to use a digest algorithm that is not supported at a specific FIPS mode level. The digest algorithm is specified as part of the RSASSA-PSS parameters. Supported digest types are SHA-256, SHA-384, and SHA-512.

User response

Ensure that the digest algorithm is supported for the requested operation or that it is supported if executing in FIPS mode.

033530C1 **RSASSA-PSS mask generation algorithm is not supported**

Explanation

The RSASSA-PSS cryptographic mask generation algorithm specified is not supported by the current level of the System SSL run time. Supported mask type is Mgf1.

User response

Ensure that the mask generation algorithm is supported for the requested operation.

033530C2 **PKDS RSA private key type not valid for RSASSA-PSS signature generation**

Explanation

A secure RSA private key residing in the PKDS is an incorrect type to perform RSASSA-PSS digital signing.

User response

Ensure that the secure private has been generated correctly into the PKDS.

033530C3 **Input parameter is not valid.**

Explanation

The value specified for an input parameter is not a valid value for the requested operation.

User response

Ensure that a valid non-NULL value is specified for the input parameter of the requested operation.

033530C4 **Incorrect Suite B certificate key usage**

Explanation

TLS Suite B certificate contains extraneous key usage bits.

For TLS Suite B CA certificates, the certificate sign, CRL sign, digital signature and nonRepudiation bits can be set. All other bits are not allowed to be set.

For TLS Suite B end-entity certificates, the digital signature and nonRepudiation bits can be set. All other bits are not allowed to be set.

User response

Obtain a certificate, which contains the allowed key usage settings.

SSL started task messages (GSK01nnn)

Messages from the SSL started task (GSKSRVR) have the prefix "GSK01". These status codes include informational messages, including errors that require a user response.

GSK01001I **System SSL version *version.release* Service level *level*.**

Explanation

This message displays the System SSL version, release, and service level.

User response

None

GSK01002E **Insufficient storage available.**

Explanation

The SSL server is unable to obtain storage for an internal control block.

User response

Increase the storage available to the GSKSRVR started task and then try the request again.

GSK01003I **SSL server initialization complete.**

Explanation

The server initialization is complete.

User response

None

GSK01004I **SSL server shutdown requested.**

Explanation

The system operator entered a STOP command for the SSL server.

User response

None

GSK01005E **Unrecognized SSL server command: Specify DISPLAY, TRACE, RESET, or STOP.**

Explanation

An unrecognized command name is specified on a MODIFY operator command. The valid SSL server commands are DISPLAY, TRACE, RESET, and STOP.

User response

Specify a valid SSL server command.

GSK01006E **Incorrect command option specified.**

Explanation

An incorrect SSL server command option is specified.

The valid DISPLAY command options are:

- CRYPTO - Display the available encryption algorithms.
- LEVEL - Display the System SSL version, release, and service level.
- SIDCACHE - Display the sysplex session cache status.
- STATS - The valid STATS command options are:
  - SIDCACHE - Display the sysplex session cache statistics.
  - TICKETCACHE – Display the sysplex session ticket cache statistics.
- TICKETCACHE – Display the sysplex session ticket cache status.
- XCF - Display SSL sysplex status.

The valid TRACE command options are:

- OFF - Turn off SSL tracing
- ON,level - Enable SSL tracing using the specified trace level.

The valid RESET command option is:

- STATS – The valid STATS command options are:
  - SIDCACHE – Resets the sysplex session cache statistics.
  - TICKETCACHE – Resets the sysplex session ticket cache statistics.

User response

Specify a valid command option.

GSK01007E **Missing command option.**

Explanation

An SSL server command is entered which requires a command option but no command option is entered.

User response

Enter a complete SSL server command.

GSK01008I **Sysplex status.**

Explanation

This message is displayed in response to the SSL server DISPLAY XCF command. The remaining lines in this multi-line message display the status of each SSL server in the sysplex. A server is ACTIVE if the GSKSRVR started task is running. A security server is INACTIVE if the GSKSRVR started task has been stopped. No entry is displayed for a system where the GSKSRVR started task has not been started.

User response

None

GSK01009I **Cryptographic status.**

Explanation

This message is displayed in response to the SSL server DISPLAY CRYPTO command. The remaining lines in this multi-line message display the available encryption algorithms.

User response

None

GSK01010A **The SSL server is already running.**

Explanation

The GSKSRVR started task is already running. Only one instance of the SSL server may be active in the same system.

User response

Stop the GSKSRVR started task before starting a new instance of the SSL server.

GSK01011A **The SSL server is not APF-authorized.**

Explanation

The GSKSRVR started task is not running with APF authorization.

User response

Add the pdsename.SIEALNKE data set to the list of APF-authorized data sets and then restart the GSKSRVR started task. If you are using a STEPLIB or JOBLIB for the GSKSRVR started task, verify that all data sets in the concatenation are APF-authorized.

GSK01012A **Unable to make address space non-swappable: Error *error-code*.**

Explanation

The SSL server is unable to make its address space non-swappable. The error code is the value that is returned by the SYSEVENT system service.

User response

Verify that the GSKSRVR started task is APF-authorized. See the SYSEVENT description in *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO* for more information. Contact your service representative if the error persists.

GSK01013I **SSL server restart registration complete on system.**

Explanation

The GSKSRVR started task successfully registered with ARM (Automatic Restart Management) on the indicated system. The GSKSRVR started task is automatically restarted if it fails unexpectedly (it does not restart if it detects an error and stops). The ARM element type is SYSSSL and the ARM element name is GSKSRVR_system-name. The ARM policy can be used to override the default registration values if needed.

User response

None

GSK01014I **SSL server restarting on system.**

Explanation

The GSKSRVR started task is being restarted following an unexpected failure. The RESTART_ATTEMPTS value in the ARM policy determines the number of restarts that are attempted.

User response

None

GSK01015E **Unable to register for restart: Error *error-code*, Reason *reason-code*.**

Explanation

The GSKSRVR started task is unable to register with ARM (Automatic Restart Management). The IXCARM request failed with the indicated error and reason codes.

User response

See the IXCARM description in *z/OS MVS Programming: Sysplex Services Reference* for more information. Contact your service representative if the error persists.

GSK01016E **Unable to unregister for restart: Error *error-code*, Reason *reason-code*.**

Explanation

The GSKSRVR started task is unable to unregister with ARM (Automatic Restart Management). The IXCARM request failed with the indicated error and reason codes.

User response

See the IXCARM description in *z/OS MVS Programming: Sysplex Services Reference* for more information. Contact your service representative if the error persists.

GSK01017I **SSL server restart deregistration complete on system.**

Explanation

The GSKSRVR started task successfully deregistered with ARM (Automatic Restart Management) on the indicated system. The SSL server is no longer automatically restarted if it fails unexpectedly.

User response

None

GSK01018I **Trace option processed: *trace-option*.**

Explanation

The indicated trace request has been processed by the SSL server.

User response

None

GSK01019E **Unable to create mutex: *error-text*.**

Explanation

The GSKSRVR started task is unable to create a mutex for the indicated reason.

User response

See the **pthread_mutex_init()** description in *z/OS XL C/C++ Runtime Library Reference* for more information. Contact your service representative if the error persists.

GSK01020E **Unable to lock mutex: *error-text*.**

Explanation

The GSKSRVR started task is unable to lock a mutex for the indicated reason.

User response

See the **pthread_mutex_lock()** description in *z/OS XL C/C++ Runtime Library Reference* for more information. Contact your service representative if the error persists.

GSK01021E **Unable to create thread: *error-text*.**

Explanation

The GSKSRVR started task is unable to create a thread for the indicated reason.

User response

See the **pthread_create()** description in *z/OS XL C/C++ Runtime Library Reference* for more information. Contact your service representative if the error persists.

GSK01022E **Unable to initialize local services: Error *error-code*, Reason *reason-code*.**

Explanation

The GSKSRVR started task is unable to initialize the local services support. The error code indicates that the failing system function and the reason code are the error code that is returned by the system function.

These error codes are defined:

- 1 = The job step is not APF-authorized.
- 2 = The security server is already running.
- 3 = The ESTAEX request failed.
- 5 = The LXRES request failed.
- 6 = The ETCRE request failed.
- 7 = The ETCON request failed.
- 8 = The IEANTCR request failed.
- 9 = The CTRACE DEFINE request failed.

User response

Verify that the GSKSRVR started task is APF-authorized. See the system function description in *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for more information. Contact your service representative if the error persists.

GSK01023E **Unable to create session cache data space: Error *error-code*, Reason reason-code.**

Explanation

The GSKSRVR started task is unable to create the session cache data space.

These error codes are defined:

1 = DSPSERV CREATE failed.
The reason code contains the DSPSERV return code in the upper halfword and bits 8-23 of the DSPSERV reason code in the lower halfword.

2 = ALESERV ADD failed.
The reason code is the ALESERV return code.

User response

See the DSPSERV or ALESERV description in *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN* for more information. Contact your service representative if the error persists.

GSK01024E **Unable to initialize cross-system services: Error *error-code*, Reason *reason-code*.**

Explanation

The GSKSRVR started task is unable to initialize cross-system services.

These error codes are defined:

1 = The job step is not APF-authorized.

3 = IXCJOIN failed.
The reason code contains the IXCJOIN return code in the upper halfword and the IXCJOIN reason code in the lower halfword.

4 = IXCQUERY failed.
The reason code contains the IXCQUERY return code in the upper halfword and the IXCQUERY reason code in the lower halfword.

User response

See the IXCJOIN or IXCQUERY description in *z/OS MVS Programming: Sysplex Services Reference* for more information. Contact your service representative if the error persists.

GSK01025I **System *name* has joined the GSKSRVR group.**

Explanation

The GSKSRVR started task completed initialization on the indicated system and is now a member of the GSKSRVGP cross-system group.

User response

None

GSK01026I **System *name* has left the GSKSRVR group.**

Explanation

The GSKSRVR started task is stopping on the indicated system and has left the GSKSRVGP cross-system group.

User response

None

GSK01027I **Cross-system services ended due to sysplex partitioning.**

Explanation

The local system is leaving the sysplex. As a result, GSKSRVR cross-system services are no longer available.

User response

None

GSK01028E **Local program call request failed: Error *error-code*.**

Explanation

The GSKSRVR started task is unable to process a local program call request.

These error codes are defined:

- 8 = Parameter buffer overflow.
- 12 = Unable to allocate storage.
- 16 = Local service support is not enabled.
- 20 = Program call task abended.
- 24 = Unable to obtain control lock.
- 28 = Requested function is not supported.

User response

Contact your service representative.

GSK01029I **Cross-system services are not available.**

Explanation

The DISPLAY XCF command cannot be processed because cross-system services are not available.

User response

None

GSK01030I **Maximum number of lines displayed.**

Explanation

The maximum number of lines that are allowed for a multi-line write-to-operator message is reached.

User response

None

GSK01031I **No session cache users.**

Explanation

The DISPLAY SIDCACHE command was issued but there are no session cache users to display.

User response

None

GSK01032I **Session cache status**

Explanation

This message is displayed in response to the SSL server DISPLAY SIDCACHE command. The remaining lines in this multi-line message display the cache users.

User response

None

GSK01033E **Unable to extend the session cache data space: Error *error-code*, Reason *reason-code*.**

Explanation

The GSKSRVR started task is unable to increase the size of the session cache data space.

The error codes have these values:

1 = DSPSERV EXTEND failed.
The reason code contains the DSPSERV return code in the upper halfword and bits 8-23 of the DSPSERV reason code in the lower halfword.

User response

The new session cache entry is not stored in the session cache data space. See the DSPSERV description in *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN* for more information. Contact your service representative if the error persists.

GSK01034E **Unable to send cross-system message: Error *error-code*, Reason *reason-code*.**

Explanation

The GSKSRVR started task is unable to send a message to another system member of the GSKSRVGP group or another system member in the GSKSRVGP group is unable to successfully fulfill the requested operation. The GSKSRVGP group identifies each GSKSRVR system member participating in the cross-system group.

The error codes have these values:

- 1 = XCF message has been successfully sent; however, the target system member timed out while processing the received System SSL request.
- 2 = Cross-system services are not available. The local GSKSRVR system member is unable to communicate with other system members in the GSKSRVGP cross-system group.
- 3 = No longer used.
- 4 = No longer used.
- 5 = Unable to allocate storage on the target system member while processing the System SSL request.
- 6 = No longer used.
- 7 = No longer used.
- 8 = IXCMSGO failed. The reason code contains the IXCMSGO return code in the upper halfword and the IXCMSGO reason code in the lower halfword.
- 9 = IXCMSGI failed on the target system member. The reason code contains the IXCMSGI return code in the upper halfword and the IXCMSGI reason code in the lower halfword.
- 10 = Request function code is not supported.
- 11 = Request canceled.
- 12 = Unknown notification message.
- 13 = No longer used.
- 14 = Unable to allocate storage on the local system member.
- 15 = IXCMSGI failed on the local system member. The reason code contains the IXCMSGI return code in the upper halfword and the IXCMSGI reason code in the lower halfword.

User response

The requested operation is not processed.

- For error code 1, the requested operation was sent; however, the target system member timed out while processing the received System SSL request. If this error code occurs often during a period of one minute or more, contact your service representative.

- For error code 2, the requested operation could not be sent to the target system member because the local system member was unable to join the GSKSRVGP cross-system group. The local GSKSRVR should be restarted to correct this issue.
- For error code 5, the requested operation could not be completed because an out-of-storage error occurred on the target system member when receiving the XCF message. If this error code occurs often, increase the region size of the other members of the GSKSRVGP.
- For error codes 8, 9, and 15, see the IXCMSGI or IXCMSGO description in *z/OS MVS Programming: Sysplex Services Reference* for more information.
- For error code 12, an unknown notification message has been received from the target system member.
- For error code 14, the local system member was unable to allocate storage to receive the response message from the target system member. Ensure that the local system member has enough space to handle the incoming XCF response message. If this error code occurs often, increase the region size of this GSKSRVR system member.

Contact your service representative if the error persists.

GSK01035E **SSL server is not available.**

Explanation

The SSL server task is not available. This error occurs if the GSKSRVR started task is not running, has not completed initialization, or is stopping.

User response

Wait until the GSKSRVR started task is available and then try the failing request again.

GSK01036E **No job name specified.**

Explanation

No job name was specified on the TRACE CT command when starting a component trace.

User response

Specify at least one job name when starting a component trace.

GSK01037E **Unable to call SSL server: Error *errorcode*, Reason *reasoncode*.**

Explanation

The command processor for the TRACE CT command is unable to call the GSKSRVR started task.

These error codes are defined:

- 8 = Parameter buffer overflow
- 12 = Unable to allocate storage
- 16 = Local service support is not enabled
- 20 = Program call task abended (the reason is the abend code)
- 24 = Unable to obtain control lock
- 28 = Requested function is not supported

User response

Verify that the GSKSRVR started task is running on the local system. Contact your service representative if the error persists.

GSK01038E **Incorrect trace option specified.**

Explanation

The OPTIONS parameter on the TRACE CT command does not specify a valid SSL trace option. The only valid option is LEVEL=n where n is the requested SSL trace level. See Appendix A, “Environment variables,” on page 763 for the description of the GSK_TRACE environment variable for more information about setting the SSL trace level.

User response

Specify a valid SSL trace option.

GSK01039E **The trace buffer size must be between 64K and 512K.**

Explanation

The trace buffer size that is specified on the TRACE CT command must be between 64K and 512K.

User response

Specify a valid trace buffer size.

GSK01040I **SSL component trace started.**

Explanation

The SSL component trace has been started. The jobs that are specified on the TRACE CT command may be already running or may be started after the TRACE CT command is processed. However, any jobs that are already running must have been started after the GSKSRVR started task was started.

User response

None

GSK01041I **SSL component trace ended.**

Explanation

The SSL component trace has ended.

User response

None

GSK01042E **Incorrect OPTIONS syntax**

Explanation

The OPTIONS parameter syntax on the IPCS CTRACE command is not correct for an SSL component trace. SSL supports three options: JOB, PID, and TID. The CTRACE OPTIONS parameter is specified as CTRACE COMP(GSKSRVR) OPTIONS((JOB(name),PID(hexid),TID(hexid))).

User response

Specify a valid OPTIONS parameter.

GSK01043E **Incorrect trace option.**

Explanation

An incorrect trace option was specified on the IPCS CTRACE command for an SSL component trace. SSL supports three options: JOB, PID, and TID. The CTRACE OPTIONS parameter is specified as CTRACE COMP(GSKSRVR) OPTIONS((JOB(name),PID(hexid),TID(hexid))). The job name must be 1-8 characters. The hexadecimal identifier for PID and TID must be 1-8 hexadecimal digits.

User response

Specify a valid OPTIONS parameter.

GSK01044E **Duplicate trace option.**

Explanation

An SSL trace option is specified more than once on the IPCS CTRACE command.

User response

Do not specify the same trace option more than once.

GSK01045E **Incorrect hexadecimal value.**

Explanation

The value for the PID and TID trace options for the IPCS CTRACE command must be a hexadecimal value consisting of 1-8 hexadecimal digits.

User response

Specify a valid hexadecimal value.

GSK01046I **Trace filter options: *option list***

Explanation

The IPCS CTRACE command specifies one or more trace entry filter options.

User response

None

GSK01047I **SSL component trace started for *jobname/JobID*.**

Explanation

The SSL component trace has started for the indicated job. This message is displayed once for each job that matches the jobnames that are specified in the TRACE CT command. Tracing is started and the message is displayed when SSL component trace has been started and activation has been detected by the System SSL APIs.

User response

None

GSK01048W **Component trace buffer overflow.**

Explanation

Both of the SSL component trace buffers are full and additional trace entries cannot be added until the trace writer has written the current data to the trace data set. Trace entries are discarded until the trace writer emptied one of the trace buffers.

User response

Increase the trace buffer size that is specified on the TRACE command and restart the component trace.

GSK01049A **The SSL server must be started as a started task.**

Explanation

The GSKSRVR was not started as a started task. The user ID of the GSKSRVR started task must be defined to the started procedure. See "Configuring the SSL started task" on page 612 for more information.

User response

Start GSKSRVR as a started task.

GSK01050I **SSL Component trace started for *Jobname/JobID/ProcessID***

Explanation

The SSL component trace started for the indicated process. This message is displayed each time component trace is started for each SSL process whose job name matches one of the job names that are specified in the TRACE CT command. Tracing is started and the message is displayed when SSL component trace has been started and activation has been detected by the System SSL APIs. This message is written to the system log only.

User response

None.

GSK01051E ***Jobname/ASID* Hardware encryption error. ICSF hardware encryption processing is unavailable**

Explanation

The specified job encountered a severe hardware encryption error during ICSF hardware processing. Encryption functions are processed in software. See message GSK01052W in the system log for algorithm-specific detail.

This message can also be issued when the job is running in FIPS mode and if during System SSL runtime initialization, it was determined that a cryptographic card that is configured as an accelerator was available and a later call to ICSF for an RSA operation resulted in a severe error.

User response

Ensure that ICSF hardware encryption services are installed and functioning correctly. When ICSF hardware encryption services have been restored and functioning correctly, processing should return to using hardware.

GSK01052W ***Jobname/ASID* Hardware encryption error. *Algorithm* encryption processing switched to software**

Explanation

The specified job encountered a severe ICSF or hardware encryption error. ICSF or hardware processing for the specified algorithm has been disabled. Any future encryption or decryption using this algorithm is performed in software for the particular SSL application or process.

User response

Ensure that ICSF hardware encryption services are installed and functioning correctly. When ICSF hardware encryption services have been restored and functioning correctly, processing should return to using hardware.

GSK01053E **Known Answer Tests failed with status *status-code***

Explanation

The FIPS power-on known answer tests failed with the reported CMS status code. System SSL is unable to execute in FIPS mode.

User response

See "CMS status codes (03353xxx)" on page 700 for information about the reported status code. Collect a System SSL trace of the failing application and contact your service representative if the error persists.

GSK01054E **SSL server starting in non-FIPS mode. Status *status-code***

Explanation

The environment variable GSK_FIPS_STATE was specified in the envar file in the GSKSRVR home directory, yet the started task was unable to execute in FIPS mode. The started task is started in non-FIPS mode.

If the indicated CMS status code is zero, then the value that is specified for the environment variable was not GSK_FIPS_STATE_ON, so FIPS mode was not attempted. If the indicated CMS status code is non-zero, an attempt was made to set FIPS mode but failed.

The System SSL started task continues to execute in non-FIPS mode. In non-FIPS mode, GSKSRVR does not provide sysplex session ID caching for FIPS mode application servers. Sysplex session ID caching is provided only for non-FIPS mode application servers.

User response

If the indicated status is zero, correct the environment variable GSK_FIPS_STATE so that it either specifies the value 'GSK_FIPS_STATE_ON' or remove the environment variable if FIPS mode is not required for the started task. If the indicated status is non-zero, see "CMS status codes (03353xxx)" on page 700 for information about the reported status code.

Collect a System SSL trace of the failing application and contact your service representative if the error persists.

GSK01057I **SSL server starting in FIPS mode.**

Explanation

GSK_FIPS_STATE=GSK_FIPS_STATE_ON was specified in the envar file in the GSKSRVR home directory.

The System SSL started task has initialized successfully and executes in FIPS mode. In FIPS mode, GSKSRVR provides sysplex session ID caching for both FIPS mode and non-FIPS mode application servers.

User response

None

GSK01064I **GSK_FIPS_ICSF_TRACKING environment variable is no longer supported.**

Explanation

The GSK_FIPS_ICSF_TRACKING environment variable is not supported after z/OS V1R13, and the setting that is specified in the SSL started task environment variable file is ignored.

User response

Remove the GSK_FIPS_ICSF_TRACKING environment variable from the SSL started task environment variable file.

GSK01065E **Unable to create session ticket cache data space: Error *error-code*, Reason *reason-code*.**

Explanation

These error codes are defined:

1 = DSPSERV CREATE failed.
The reason code contains the DSPSERV return code in the upper halfword and bits 8-23 of the DSPSERV reason code in the lower halfword.

2 = ALESERV ADD failed.
The reason code is the ALESERV return code.

User response

See the DSPSERV or ALESERV description in *z/OS MVS Programming: Sysplex Services Reference* for more information. Contact your service representative if the error persists.

GSK01066E **Unable to extend the session ticket cache data space: Error *error-code*, Reason *reason-code*.**

Explanation

The GSKSRVR started task is unable to increase the size of the session ticket cache data space.

The error codes have these values:

1 = DSPSERV EXTEND failed.
The reason code contains the DSPSERV return code in the upper halfword and bits 8-23 of the DSPSERV reason code in the lower halfword.

User response

The session ticket cache entry is not stored in the session ticket cache data space. See the DSPSERV description in *z/OS MVS Programming: Sysplex Services Reference* for more information. Contact your service representative if the error persists.

GSK01067E **Unable to create session ticket cache mutex: *error-text*.**

Explanation

The GSKSRVR started task is unable to create a session ticket cache mutex for the indicated reason.

User response

See the **pthread_mutex_init()** description in *z/OS XL C/C++ Runtime Library Reference* for more information. Contact your service representative if the error persists.

GSK01068E **Unable to lock session cache mutex for *operation*: *error-text*.**

Explanation

The GSKSRVR started task is unable to lock the session ticket cache mutex for the indicated operation and reason.

User response

See the **pthread_mutex_lock()** description in *z/OS XL C/C++ Runtime Library Reference* for more information. Contact your service representative if the error persists.

GSK01069E **Unable to create component trace mutex: *error-text*.**

Explanation

The GSKSRVR started task is unable to create a component trace mutex for the indicated reason.

User response

See the **pthread_mutex_init()** description in *z/OS XL C/C++ Runtime Library Reference* for more information. Contact your service representative if the error persists.

GSK01070E **Unable to reply to cross-system message function *function-code*: ID 0*xid-value*, System *system*, Error *error-code*, Reason *reason-code*.**

Explanation

The GSKSRVR started task is unable to send a reply to another member of the GSKSRVGP group, which is specified in this message.

The function codes have these values:

1
: Session cache entry retrieval.

2
: Session cache entry deletion.

3
: Session ticket cache entry retrieval.

4
: Session ticket cache entry update.

5
: Session ticket cache entry deletion.

The ID value indicates the identifier associated with the session cache or session ticket cache entry that is being retrieved, deleted, or updated.

The error codes have these values:

2
: Cross-system services are not available.

4
: User not authorized to access token data.

8
: IXCMSGOX failed. The reason code contains the IXCMSGOX return code in the upper halfword and the IXCMSGOX reason code in the lower halfword.

11
: Request canceled.

12
: Unknown notification message.

User response

The XCF response is unable to be sent to the specified system. See the IXCMSGOX description in *z/OS MVS Programming: Sysplex Services Reference* for more information. Contact your service representative if the error persists.

GSK01071E **Unable to create session cache read/write lock: *error-text*.**

Explanation

The GSKSRVR started task is unable to create a session cache read/write lock for the indicated reason. Session cache statistics are not gathered or updated.

User response

See the **pthread_rwlock_init()** description in *z/OS XL C/C++ Runtime Library Reference* for more information. Contact your service representative if the error persists.

GSK01072E **Unable to obtain session cache read lock: *error-text*.**

Explanation

The GSKSRVR started task is unable to obtain session cache read lock for the indicated reason. Session cache statistics are not displayed in response to a DISPLAY STATS,SIDCACHE operator modify command or session cache statistics are not updated during a session cache entry add or retrieval operation.

User response

See the **pthread_rwlock_rdlock()** description in *z/OS XL C/C++ Runtime Library Reference* for more information. Contact your service representative if the error persists.

GSK01073E **Unable to obtain session cache write lock: *error-text*.**

Explanation

The GSKSRVR started task is unable to obtain session cache write lock for the indicated reason. Session cache statistics are not displayed in response to a RESET STATS,SIDCACHE operator modify command and are not reset to 0.

User response

See the **pthread_rwlock_wrlock()** description in *z/OS XL C/C++ Runtime Library Reference* for more information. Contact your service representative if the error persists.

GSK01074E **Unable to create session ticket cache read/write lock: *error-text*.**

Explanation

The GSKSRVR started task is unable to create a session ticket cache read/write lock for the indicated reason. Session ticket cache statistics are not gathered or updated.

User response

See the **pthread_rwlock_init()** description in *z/OS XL C/C++ Runtime Library Reference* for more information. Contact your service representative if the error persists.

GSK01075E **Unable to obtain session ticket cache read lock: *error-text*.**

Explanation

The GSKSRVR started task is unable to obtain session ticket cache read lock for the indicated reason. Session ticket cache statistics are not displayed in response to a DISPLAY STATS,TICKETCACHE operator modify command or session cache statistics are not updated during a session ticket cache entry add, update, or retrieval operation.

User response

See the **pthread_rwlock_rdlock()** description in *z/OS XL C/C++ Runtime Library Reference* for more information. Contact your service representative if the error persists.

GSK01076E **Unable to obtain session ticket cache write lock: *error-text*.**

Explanation

The GSKSRVR started task is unable to obtain session ticket cache write lock for the indicated reason. Session ticket cache statistics are not displayed in response to a RESET STATS,TICKETCACHE operator modify command and are not reset to 0.

User response

See the **pthread_rwlock_wrlock()** description in *z/OS XL C/C++ Runtime Library Reference* for more information. Contact your service representative if the error persists.

GSK01077I **No session ticket cache users.**

Explanation

The DISPLAY TICKETCACHE command was issued, but there are no session ticket cache users to display.

User response

None.

GSK01078I **Session ticket cache status**

Explanation

This message is displayed in response to the SSL server DISPLAY TICKETCACHE command. The remaining lines in this multi-line message display the session ticket cache users. See "Interpreting the session ticket cache statistics" on page 619 for more information.

User response

None.

GSK01079I **Session ticket cache statistics**

Explanation

This message is displayed in response to the SSL server DISPLAY STATS,TICKETCACHE command. The remaining lines in this multi-line display the session ticket cache statistics, which include add, update, and retrieval statistics for the local GSKSRVR instance and requests from other GSKSRVR instances in the sysplex. See "Interpreting the session ticket cache statistics" on page 619 for more information.

User response

None.

GSK01080I **Session ticket cache statistics before reset**

Explanation

This message is displayed in response to the SSL server RESET STATS,TICKETCACHE command. The remaining lines in this multi-line display the session ticket cache statistics, which include add, update, and retrieval statistics for the local GSKSRVR instance and requests from other GSKSRVR instances in the sysplex. The session ticket cache statistics are displayed prior to the statistics reset being performed. See "Interpreting the session ticket cache statistics" on page 619 for more information.

User response

None.

GSK01081I **Session cache statistics**

Explanation

This message is displayed in response to the SSL server DISPLAY STATS,SIDCACHE command. The remaining lines in this multi-line display the session cache statistics, which include add and retrieval statistics for the local GSKSRVR instance and requests from other GSKSRVR instances in the sysplex. See "Interpreting the session cache statistics" on page 615 for more information.

User response

None.

GSK01082I **Session cache statistics before reset**

Explanation

This message is displayed in response to the SSL server RESET STATS,SIDCACHE command. The remaining lines in this multi-line display the session cache statistics, which include add and retrieval statistics for the local GSKSRVR instance and requests from other GSKSRVR instances in the sysplex. The session cache statistics are displayed prior to the statistics reset being performed. See "Interpreting the session cache statistics" on page 615 for more information.

User response

None.

GSK01083E **Unable to send to cross-system message function *function-code*: ID *id-value*, System *system*, Error *error-code*, Reason *reason-code*.**

Explanation

The GSKSRVR started task is unable to send or receive a response from the indicated system in the message. The indicated system is another member of the GSKSRVGP group or another system member in the GSKSRVGP group is unable to successfully fulfill the requested operation. The GSKSRVGP group identifies each GSKSRVR system member participating in the cross-system group.

The function codes have these values:

- 1 = Session cache entry retrieval.
- 2 = Session cache entry deletion.
- 3 = Session ticket cache entry retrieval.
- 4 = Session ticket cache entry update.
- 5 = Session ticket cache entry deletion.

The ID value indicates the identifier associated with the session cache or session ticket cache entry that is being retrieved, deleted, or updated.

The error codes have these values:

- 1 = XCF message has been successfully sent; however, the target system member timed out while processing the received System SSL request.
- 2 = Cross-system services are not available. The local GSKSRVR system member is unable to communicate with other system members in the GSKSRVGP cross-system group.
- 5 = Unable to allocate storage on the target system member while processing the System SSL request.
- 8 = IXCMSGOX failed. The reason code contains the IXCMSGOX return code in the upper halfword and the IXCMSGOX reason code in the lower halfword.
- 9 = IXCMSGIX failed on the target system member. The reason code contains the IXCMSGIX return code in the upper halfword and the IXCMSGIX reason code in the lower halfword.
- 10 = Request function code is not supported.
- 11 = Request canceled.
- 12 = Unknown notification message.
- 14 = Unable to allocate storage on the local system member.
- 15 = IXCMSGIX failed on the local system member. The reason code contains the IXCMSGIX return code in the upper halfword and the IXCMSGIX reason code in the lower halfword.
- 16 = Request function contains invalid data.
- 17 = XCF message has been successfully sent; however, the target system member was unable to obtain the session ticket cache mutex lock during a retrieval operation.
- 18 = XCF message has been successfully sent; however, the target system member was unable to obtain the session ticket cache mutex lock during an update operation.

For error codes 1, 10, 16, 17, and 18, there may be a corresponding GSK01084E message issued on the system indicated in this message.

For error code 11, there may be a corresponding GSK01070E message issued on the system indicated in this message.

User response

The requested operation is not processed.

- For error code 1, the requested operation was sent; however, the target system member timed out while processing the received System SSL request. If this error code occurs often during a period of one minute or more, contact your service representative.
- For error code 2, the requested operation could not be sent to the target system member because the local system member was unable to join the GSKSRVGP cross-system group. The local GSKSRVR should be restarted to correct this issue.
- For error code 5, the requested operation could not be completed because an out-of-storage error occurred on the target system member when receiving the XCF message. If this error code occurs often, increase the region size of the other members of the GSKSRVGP.
- For error codes 8, 9, and 15, see the IXCMSGIX or IXCMSGOX description in *z/OS MVS Programming: Sysplex Services Reference* for more information.
- For error code 12, an unknown notification message has been received from the target system member.
- For error code 14, the local system member was unable to allocate storage to receive the response message from the target system member. Ensure that the local system member has enough space to handle the incoming XCF response message. If this error code occurs often, increase the region size of this GSKSRVR system member.
- For error code 17, the system indicated in this message will display message GSK01068E indicating the reason why the mutex lock could not be obtained during the retrieval operation.

- For error code 18, the system indicated in this message will display message GSK01068E indicating the reason why the mutex lock could not be obtained during the update operation.

GSK01084E **Unable to process incoming cross-system message function *function-code*: ID *id-value*, System *system*, Error *error-code*, Reason *reason-code*.**

Explanation

The GSKSRVR started task is processing an incoming message from the indicated system in the message. The indicated system is another member of the GSKSRVGP group.

The function codes have these values:

- 1 = Session cache entry retrieval.
- 2 = Session cache entry deletion.
- 3 = Session ticket cache entry retrieval.
- 4 = Session ticket cache entry update.
- 5 = Session ticket cache entry deletion.

The ID value indicates the identifier associated with the session cache or session ticket cache entry that is being retrieved, deleted, or updated.

The error codes have these values:

- 1 = The system timed out while processing the received System SSL request.
- 10 = Request function code is not supported.
- 16 = Request function contains invalid data.
- 17 = The system was unable to obtain the session ticket cache mutex lock during a retrieval operation. The GSK01068E message should indicate the reason why the mutex lock could not be obtained.
- 18 = The system was unable to obtain the session ticket cache mutex lock during an update operation. The GSK01068E message should indicate the reason why the mutex lock could not be obtained.

User response

The requested operation is not processed.

- For error code 1, the requested operation was received; however, there was an issue obtaining an internal lock. If this error code occurs often during a period of one minute or more, contact your service representative.
- For error codes 10 and 16, these are internal processing errors. If these errors occur, contact your service representative.
- For error code 17, the GSK01068E message is issued indicating the reason why the mutex lock could not be obtained during the retrieval operation. Investigate the GSK01068E message and provide it to your service representative if the error persists.
- For error code 18, the GSK01068E message is issued indicating the reason why the mutex lock could not be obtained during the update operation. Investigate the GSK01068E message and provide it to your service representative if the error persists.

Utility messages (GSK00nnn)

System SSL utility messages have the prefix "GSK00". These messages identify conditions from utilities (such as **gsktrace** and System SSL run time) that require a system operator response.

GSK00001E **Unable to open trace file *name*: *error-message***

Explanation

The **gsktrace** command is unable to open the trace file.

User response

Verify that the trace file exists and can be accessed by the user issuing the **gsktrace** command. Contact your service representative if the error persists.

GSK00002E **Unable to read trace file *name*: *error-message***

Explanation

The **gsktrace** command is unable to read the trace file.

User response

Verify that there are no file system errors and that the trace file has not been modified. Contact your service representative if the error persists.

GSK00003E **Trace record length *size* exceeds the maximum length.**

Explanation

A record in the trace file is longer than the maximum length for a trace record. This probably means that the trace file has been modified.

User response

Verify that the trace file has not been modified and was created by a compatible level of the System SSL run time.

GSK00004R **Enter password:**

Explanation

The System SSL run time needs a database or certificate password.

User response

Enter the requested password.

GSK00005R **Re-enter password:**

Explanation

The System SSL run time is verifying the password.

User response

Enter the same password you entered for the first password prompt.

GSK00006E **File *name* is not a valid SSL trace file.**

Explanation

The **gsktrace** command is unable to process the file because it is not in the proper format. This error can occur if the trace file was created by an earlier level of the System SSL run time.

User response

Process the trace file using the **gsktrace** command that is at the same level as the System SSL run time which created the trace file.

GSK00007R **Enter new password:**

Explanation

The System SSL run time is needs a new database password.

User response

Enter the requested password.

GSK00008E **z/OS PKCS #11 function *function-name* failed with return code *return-code***

Explanation

The indicated z/OS PKCS #11 function failed with the reported return code. The return code is displayed in hexadecimal with its decimal value in parentheses. See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information about the function and return code value.

User response

See *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications* for more information about the reported function and return code. If the problem cannot be resolved, contact your service representative.

GSK00009E **Problem encountered with the installation of the gskkyman utility.**

Explanation

During installation, the sticky bit is set on for the **gskkyman** utility. If the sticky bit is turned off, attempts to invoke **gskkyman** fails.

User response

Verify that the sticky bit is set. If not set, set the sticky bit.

To check the sticky bit setting, issue:

```
ls -l /usr/lpp/gskssl/bin/gskkyman
```

The first part of the output should be:

```
-rwxr-xr-t
```

The t indicates that the sticky bit is on.

To set the sticky bit on, issue the following command from an authorized ID:

```
chmod +t /usr/lpp/gskssl/bin/gskkyman
```

Appendix A. Environment variables

These tables contain all the environment variables used by the System SSL application and read during the startup of the application.

Table 23. SSL-Specific environment variables

Environment variables	Usage	Valid values
GSK_3DES_KEYCHECK	Specifies that each part of a Triple DES key is checked to be unique when in non-FIPS mode. Uniqueness check is always performed when in FIPS mode.	A value of 0 or OFF specifies no key uniqueness check is performed. A value of 1 or ON specifies a key uniqueness check is performed. The default value is OFF.
GSK_AIA_CDP_PRIORITY	Specifies the priority order that the AIA and the CDP extensions are checked for certificate revocation information.	A value of 1 or ON indicates that the AIA extension is queried before examining the CDP extension. This means that any OCSP responders specified in the AIA extension or the OCSP responder specified in GSK_OCSP_URL is contacted before attempting to contact the HTTP servers specified in the URI values of the CDP extension. A value of 0 or OFF indicates that the CDP extension is queried before examining the AIA extension. This means that the HTTP servers specified in the URI values of the CDP extension is contacted before attempting to contact the OCSP responders in the AIA extension or the OCSP responder specified in GSK_OCSP_URL. The default value is ON.
GSK_CERT_DIAG_INFO	Specifies the circumstances in which the gsk_cert_diagnostic_callback routine should be called.	A value of FAILURE specifies that the gsk_cert_diagnostic_callback routine will only be called if the certificate validation fails for the peer. This is the default. A value of SUCCESS specifies that the gsk_cert_diagnostic_callback routine will only be called if the certificate validation is successful for the peer. A value of BOTH specifies that the gsk_cert_diagnostic_callback routine will be called for both peer certificate validation successes and failures.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_CERT_VALIDATE_KEYRING_ROOT	Specifies how certificates in a SAF key ring are validated.	A value of ON or 1 specifies that SAF key ring certificates must be validated to the root CA certificate. Specify OFF or 0 if SAF key ring certificates are only validated to the trust anchor certificate. If a sole intermediate certificate is found in a SAF key ring and the next issuer is not found in the same SAF key ring, the intermediate certificate acts as a trust anchor and the certificate chain is considered complete. By default, SAF key ring certificates are only validated to the trust anchor certificate. This setting does not affect the validation of SSL key database file, PKCS #12 file, or PKCS #11 token certificates because these certificates are always validated to the root CA certificate. The default value is OFF.
GSK_CERT_VALIDATION_MODE	Specifies which Internet standard is to be used for certificate validation.	A value of 2459 specifies certificate validation against RFC 2459 only. A value of 3280 specifies certificate validation against RFC 3280 only. A value of 5280 specifies certificate validation against RFC 5280 only. A value of ANY specifies certificate validation against RFC 2459 initially - if that fails, validate against RFC 3280 - if that fails, validate against RFC 5280. The default value is ANY. If TLS V1.3 is negotiated for a secure connection, certificate validation is done according to RFC 5280 unless explicitly specified.
GSK_CLIENT_AUTH_NOCERT_ALERT	Specifies whether the SSL server application accepts a connection from a client where client authentication is requested and the client fails to supply an X.509 certificate.	A value of OFF or 0 allows connections with clients where client authentication is requested and the client fails to supply an X.509 certificate. A value of ON or 1 terminates connections with clients where client authentication is requested and the client fails to supply an X.509 certificate. The default value is OFF.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_CLIENT_ECURVE_LIST	Specifies the list of elliptic curves or supported groups that are supported by the client as a string consisting of 1 or more 4-character values in order of preference for use. For TLS V1.0, TLS V1.1, and TLS V1.2 protocols, this list is used by the client to guide the server as to which elliptic curves are preferred when using ECC-based cipher suites. For the TLS V1.3 protocol, this list is used by the client to guide the server as to which elliptic curves are preferred and guide group selection for encryption and decryption of handshake messages. Only NIST recommended curves along with x25519 and x448 can be specified. If x25519 or x448 is specified along with TLS V1.0, TLS V1.1, or TLS V1.2 and the partner server is using an ECDSA certificate, the elliptic curve used in the server's certificate must appear in the list. This is because System SSL does not support x25519 or x448 certificates. To use Brainpool standard certificates for an SSL environment or connection, set GSK_CLIENT_ECURVE_LIST to "" or use gsk_attribute_set_buffer() to re-initialize the GSK_CLIENT_ECURVE_LIST buffer to NULL. See Table 29 on page 804 for a list of valid 4-character elliptic curve and supported groups specifications.	The default specification is 00210023002400250019. If TLS V1.3 is enabled, 0029 (x25519) is appended to the end of the default list.
GSK_CLIENT_EPHEMERAL_DH_GROUP _SIZE	Specifies the minimum Diffie-Hellman group size required by the client to be used by the server for an ephemeral Diffie-Hellman key exchange.	A value of LEGACY specifies the Diffie-Hellman group size to be 1024 in non-FIPS mode and 2048 in FIPS mode. A value of 2048 specifies the Diffie-Hellman group size to be 2048. The default value is LEGACY.

Table 23. SSL-Specific environment variables (continued)

Environment variables	**Usage**	**Valid values**
GSK_CLIENT_EXTENDED_MASTER_SECRET	Specifies if the TLS client sends the extended master secret extension to the server. This option is only applicable for TLS V1.0, TLS V1.1, and TLS V1.2 handshakes.	A value of 0, OFF, or DISABLED specifies that the TLS client does not send the extended master secret extension to the server. A value of 1, ON, or ENABLED specifies that the TLS client sends the extended master secret extension to the server but does not require the server to support the extension. A value of REQUIRED specifies that the TLS client sends the extended master secret extension to the server and requires the server to support the extension. If a server does not send the extended master secret extension, the handshake fails. Before setting this option to REQUIRED, ensure that the server being communicated with supports the extended master secret extension. If the remote server partner is a z/OS System SSL application, it must be running z/OS V2R3 or later and have PTFs for APAR OA60105 (z/OS V2R3 and V2R4) applied and active before setting this option to REQUIRED. The default value is ON.
GSK_CLIENT_TLS_KEY_SHARES	Specifies the list of the key share groups that are supported by the client during a TLS V1.3 handshake. During a TLS V1.3 handshake, the client sends the key share groups that are in common and in the same order as the supported groups list (GSK_CLIENT_ECURVE_LIST). The server selects a group from the client's preferred order and the ones that it supports. The client and server use the selected group to encrypt and decrypt TLS V1.3 handshake messages. See Table 29 on page 804 for a list of valid 4-character key share specifications.	There is no default value. This setting must be specified when enabled for TLS V1.3.
GSK_CRL_CACHE_ENTRY_MAXSIZE	Specifies the maximum size in bytes of a CRL to be kept in the LDAP CRL cache.	The valid cache entry sizes are 0 through 2147483647. The default value is 0, which means there is no limit on the size of a CRL that is allowed to be stored in the LDAP CRL cache. The size must be greater than or equal to 0.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_CRL_CACHE_EXTENDED	Specifies that LDAP extended CRL cache support is enabled. Enabling extended support: • LDAP CRLs are only cached when there is an expiration time present and it is greater than the current time. • Limits the number of CRLs that can be stored in the LDAP cache to 32. This can be overridden by specifying GSK_CRL_CACHE_SIZE. • Disables caching of temporary CRLs. This can be enabled by specifying GSK_CRL_CACHE_TEMP_CRL. • Ignores GSK_CRL_CACHE_TIMEOUT. When disabled, LDAP basic CRL caching can be used and retrieved LDAP CRLs are only cached when GSK_CRL_CACHE_TIMEOUT is greater than 0 and GSK_CRL_CACHE_SIZE is set to a non-zero number.	A value of ON or 1 enables LDAP extended CRL caching. A value of OFF or 0 disables LDAP extended CRL caching. The default value is OFF.
GSK_CRL_CACHE_SIZE	Specifies the maximum number of CRLs that are allowed to be stored in the LDAP CRL cache.	The valid cache sizes are -1 through 32000. A value of -1 means unlimited while a value of 0 means caching is not enabled. If LDAP extended CRL cache support is enabled, the default is 32 and caching only occurs if the CRL contains an expiration time that is later than the current time. If LDAP basic CRL cache support is enabled, the default is unlimited or -1 and caching only occurs when GSK_CRL_CACHE_TIMEOUT is greater than 0.
GSK_CRL_CACHE_TEMP_CRL	Specifies if a temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not reside on the LDAP server.	A value of ON or 1 indicates that a temporary LDAP CRL cache entry is added to the LDAP CRL cache. A value of OFF or 0 indicates that a temporary LDAP CRL cache entry is not to be added to the LDAP CRL cache. If LDAP extended CRL cache support is enabled, the default value is OFF. If LDAP basic CRL cache support is enabled, the default value is ON.
GSK_CRL_CACHE_TEMP_CRL_TIMEOUT	Specifies the time in hours that a temporary CRL cache entry resides in the LDAP extended CRL cache when caching of temporary CRLs is enabled. A temporary LDAP CRL cache entry is added to the LDAP CRL cache when the CRL does not reside on the LDAP server.	The range is 1 through 720 hours and defaults to 24 hours.
GSK_CRL_CACHE_TIMEOUT	Specifies the number of hours that a cached LDAP CRL remains valid.	The valid timeout values are 0 through 720 and defaults to 24. A value of 0 disables the LDAP CRL cache.

Table 23. SSL-Specific environment variables (continued)

Environment variables	**Usage**	**Valid values**
GSK_CRL_SECURITY_LEVEL	Specifies the level of security to be used when contacting LDAP servers to check CRLs for revoked certificates during certificate validation. An attempt to contact the LDAP server is performed when the CRL is not found in the LDAP cache. To enforce contact with the LDAP server for each CRL being checked, CRL caching must be disabled. For LDAP basic CRL caching, see the GSK_CRL_CACHE_TIMEOUT or GSK_CRL_CACHE_SIZE settings. For LDAP extended CRL caching, see the GSK_CRL_CACHE_SIZE setting.	LOW - Certificate validation does not fail if the LDAP server cannot be contacted. MEDIUM - Certificate validation requires the LDAP server to be contactable, but does not require a CRL to be defined. This is the default. HIGH - Certificate validation requires revocation information to be provided by the LDAP server.
GSK_EXC_ABEND_DUMP	Specifies whether the SSL condition handler should call the cdump() service to dump the current thread before resuming the failing routine. The dump is placed in the current directory unless LE is instructed to use a different directory by the _CEE_DMPTARG environment variable. See *z/OS Language Environment Programming Guide* for more information about LE callable services.	A value of 1 enables SSL dumps and a value of 0 disables SSL dumps. The default is 0. The export file contains just the requested certificate when the DER format is selected.
GSK_EXTENDED_RENEGOTIATION_ INDICATOR	Specifies the level of enforcement of renegotiation indication as specified by RFC 5746 during the initial handshake.	A value of OPTIONAL does not require the renegotiation indicator during initial handshake. This is the default. A value of CLIENT allows the client initial handshake to proceed only if the server indicates support for RFC 5746 Renegotiation. A value of SERVER allows the server initial handshake to proceed only if the client indicates support for RFC 5746 Renegotiation. A value of BOTH will allow the server and client initial handshakes to proceed only if partner indicates support for RFC 5746 Renegotiation.
GSK_HTTP_CDP_CACHE_ENTRY_MAXSIZE	Specifies the maximum size in bytes of a CRL that is allowed to be stored in the HTTP CDP CRL cache. Any CRLs larger than this size are not cached.	The valid sizes are 0 through 2147483647. The default value is 0, which means there is no limit on the size of the CRL stored in the HTTP CDP CRL cache.
GSK_HTTP_CDP_CACHE_SIZE	Specifies the maximum number of CRLs that are allowed to be stored in the HTTP CDP CRL cache.	The valid sizes are 0 through 32000. The default value is 32. If set to 0, HTTP CDP CRL caching is disabled.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_HTTP_CDP_ENABLE	Specifies if certificate revocation checking with the HTTP URI values in the CDP extension is enabled.	A value of 0, OFF, or DISABLED indicates that certificate revocation checking with the HTTP URI values in the CDP extension is not enabled. A value of 1, ON, or ENABLED indicates certificate revocation checking with the HTTP URI values in the CDP extension is enabled. The default value is OFF.
GSK_HTTP_CDP_MAX_RESPONSE_SIZE	Specifies the maximum size in bytes accepted as a response from an HTTP server when retrieving a CRL. Setting the maximum response size too small could implicitly disable HTTP CRL support.	The valid sizes are 0 through 2147483647. The default value is 204800 (200K). A value of 0 disables checking the size and allows a CRL of any size.
GSK_HTTP_CDP_PROXY_SERVER_NAME	Specifies the DNS name or IP address of the HTTP proxy server for HTTP CDP CRL retrieval.	The default value is NULL.
GSK_HTTP_CDP_PROXY_SERVER_PORT	Specifies the HTTP proxy server port for HTTP CDP CRL retrieval.	Port must be between 1 and 65535. The default port value is 80.
GSK_HTTP_CDP_RESPONSE_TIMEOUT	Specifies the time in seconds to wait for a response from the HTTP server.	The valid time limits are 0 through 43200 seconds (12 hours). The default value is 15 seconds and a value of 0 indicates that there is no time limit.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_HW_CRYPTO	Specifies whether the hardware cryptographic support is used. Note that ICSF (Integrated Cryptographic Service Facility) must be configured and running in order for System SSL to use the hardware cryptographic support that is available in the cryptographic cards. SHA-1, SHA-2, DES, Triple DES, and AES hardware functions can be used without ICSF if the zArchitecture message-security assist is installed. For more information about hardware cryptographic support, see Chapter 3, "Using cryptographic features with System SSL," on page 9. Selected hardware cryptographic functions can be disabled by setting the appropriate bits to zero in the GSK_HW_CRYPTO value. The corresponding software algorithms are used when a hardware function is disabled. These bit assignments are defined: 1 = SHA-1 digest generation 2 = 56-bit DES encryption/decryption 4 = 168-bit Triple DES encryption/decryption 8 = Public key encryption/decryption 16 = AES 128-bit encryption/decryption 32 = SHA-256 digest generation 64 = AES-256-bit encryption/decryption 128 = SHA-224 digest generation 256 = SHA-384 digest generation 512 = SHA-512 digest generation **Note:** If a hardware function bit is set on and the hardware function is unavailable, processing takes place in software.	A value of 0 disables the use of hardware support while a value of 65535 enables the use of hardware support. The default value is 65535 and only available hardware support is used.
GSK_KEY_LABEL	Specifies the label of the key that is used to authenticate the application.	Any key label. The default key is used if a key label is not specified.
GSK_KEYRING_FILE	Specifies the name of the key database file, PKCS #12 file, SAF key ring, or z/OS PKCS #11 token. A key database or PKCS #12 file is used if the GSK_KEYRING_PW environment variable is also specified. A key database file is used if GSK_KEYRING_STASH environment variable is also specified. Otherwise, a SAF key ring or z/OS PKCS #11 token is used. Note that certificate private keys are not available when using a SAF key ring owned by another user. The user must have READ access to resource USER.tokenname in the CRYPTOZ class when using a z/OS PKCS #11 token.	The SAF key ring name is specified as userid/keyring. The current user ID is used if the user ID is omitted. The z/OS PKCS #11 token name is specified as *TOKEN*/token-name. If no certificate source is specified, defaults to NULL.
GSK_KEYRING_PW	Specifies the password for the key database or PKCS #12 file.	NULL or value consisting of up to 128 characters. The default value is NULL

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_KEYRING_STASH	Specifies the name of the key database password stash file.	The stash file name always has an extension of .sth and the supplied name is changed if it does not have the correct extension. The GSK_KEYRING_PW environment variable is used instead of the GSK_KEYRING_STASH environment variable if it is also specified. The default value is NULL.
GSK_LDAP_PASSWORD	Specifies the password to use when connecting to the LDAP server.	The default value is NULL.
GSK_LDAP_PORT	Specifies the LDAP server port.	Port must be between 1 and 65535. Port 389 is used if no LDAP server port is specified.
GSK_LDAP_RESPONSE_TIMEOUT	Specifies the time in seconds to wait for a response from the LDAP server.	The valid time limits are 0 through 43200 seconds (12 hours). The default value is 15 seconds and a value of 0 indicates that there is no time limit.
GSK_LDAP_SERVER	Specifies one or more blank-separated LDAP server host names. The LDAP server is used to obtain CA certificates when validating a certificate and the local database does not contain the required certificate. The local database must contain the required certificates if no LDAP server is specified. Even when an LDAP server is used, root CA certificates must be found in the local database since the LDAP server is not a trusted data source. The LDAP server is also used to obtain certificate revocation lists.	Each host name can contain an optional port number that is separated from the host name by a colon. The default value is NULL.
GSK_LDAP_USER	Specifies the distinguished name to use when connecting to the LDAP server.	The default value is NULL.
GSK_MAX_SOURCE_REV_EXT_LOC_VALUES	Specifies the maximum number of location values that are contacted per data source when attempting validation of a certificate. The locations for revocation information are specified by the *accessLocation* in the AIA certificate extension for OCSP and the *distributionPoint* in the CDP extension for HTTP CRLs. When an HTTP URI is present in an AIA or CDP extension, validation attempts to contact the remote HTTP server to obtain revocation information. Both of these extensions can contain multiple location values and therefore have the potential to impact performance when there be a very large number of locations present.	The valid values are 0 through 256. The default value is 10 and a value of 0 indicates there is no limit on the number of locations contacted.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_MAX_VALIDATION_REV_EXT _LOC_VALUES	Specifies the maximum number of locations values that are contacted when performing validation of a certificate. The locations for revocation information are specified by the *accessLocation* in the AIA certificate extension for OCSP and the *distributionPoint* in the CDP extension for HTTP CRLs. When an HTTP URI is present in an AIA or CDP extension, validation attempts to contact the remote HTTP server to obtain revocation information. Both of these extensions can contain multiple location values and therefore has the potential to negatively impact performance when there be a very large number of locations present.	The valid values are 0 through 1024. The default value is 100 and a value of 0 indicates there is no limit on the number of locations contacted.
GSK_MIDDLEBOX_COMPAT_MODE	Specifies if the TLS V1.3 handshake process ought to use or tolerate handshake messages in a manner compliant with earlier TLS protocols to alleviate possible issues with middleboxes or proxies.	A value of 0, OFF, or DISABLED specifies that the TLS V1.3 handshake process should use the pure TLS V1.3 handshake message format. A value of 1, ON, or ENABLED specifies if the TLS V1.3 handshake process should use or tolerate handshake messages in a manner compliant with earlier TLS protocols to alleviate possible issues with middleboxes or proxies. The default value is OFF.
GSK_OCSP_CLIENT_CACHE_ENTRY _MAXSIZE	Specifies the maximum number of OCSP responses or cached certificate statuses that are allowed to be kept in the OCSP response cache for an issuing CA certificate.	The valid sizes are 0 through 32000. The default value is 0 which indicates that there is no limit on the number of cached certificate statuses allowed for a specific issuing CA certificate other than the limit imposed by GSK_OCSP_CLIENT_CACHE _SIZE. This cache size is rounded up to the nearest multiple of 16 with a minimum size of 16.
GSK_OCSP_CLIENT_CACHE_SIZE	Specifies the maximum number of OCSP responses or cached certificate statuses to be kept in the OCSP response cache.	The valid cache sizes are 0 through 32000 and defaults to 256. The OCSP response cache is disabled if 0 is specified. The OCSP response cache is allocated using the requested size rounded up to the nearest multiple of 16 with a minimum size of 16.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_OCSP_ENABLE	Specifies whether the AIA extensions are to be used for revocation checking. If GSK_OCSP_URL is specified, GSK_OCSP_ENABLE is set to ON and GSK_OCSP_URL_PRIORITY is set to ON, then the order the responders are used is GSK_OCSP_URL defined responder first and then the responders identified in the AIA extension. If GSK_OCSP_URL is specified, GSK_OCSP_ENABLE is set to ON and GSK_OCSP_URL_PRIORITY is set to OFF, then the order that responders are used is the responders identified in the AIA extension first and then the GSK_OCSP_URL defined responder.	A value of 0, OFF, or DISABLED disables OCSP revocation checking via the AIA extension. A value of 1, ON, or ENABLED enables OCSP revocation checking via the AIA extension. The default value is OFF.
GSK_OCSP_MAX_RESPONSE_SIZE	Specifies the maximum size in bytes that is accepted as a response from an OCSP responder. Setting the maximum response size too small could implicitly disable OCSP support.	The valid response sizes are 0 through 2147483647. The default value is 20480 (20K). A value of 0 disables checking of the OCSP response size and allows an OCSP response of any size.
GSK_OCSP_NONCE_CHECK_ENABLE	Specifies if OCSP response nonce checking is enabled. Nonce checking ensures the nonce in the OCSP response matches the nonce sent in the OCSP request. **Note:** Setting to ON sets GSK_OCSP_NONCE_GENERATION _ENABLE to ON.	A value of 0, OFF, or DISABLED disables OCSP nonce checking. A value of 1, ON, or ENABLED enables OCSP nonce checking. The default value is OFF.
GSK_OCSP_NONCE_GENERATION_ENABLE	Specifies if OCSP requests include a generated nonce.	A value of 0, OFF, or DISABLED disables OCSP nonce generation. A value of 1, ON, or ENABLED enables OCSP nonce generation. The default value is OFF.
GSK_OCSP_NONCE_SIZE	Specifies the size in bytes for the value of the nonce to be sent in OCSP requests.	The valid OCSP nonce sizes are 8 through 256 and defaults to 8.
GSK_OCSP_PROXY_SERVER_NAME	Specifies the DNS name or IP address of the OCSP proxy server.	The default value is NULL.
GSK_OCSP_PROXY_SERVER_PORT	Specifies the OCSP responder proxy server port.	Port must be between 1 and 65535. The default port value is 80.
GSK_OCSP_REQUEST_SIGALG	Specifies the hash and signature algorithm pair used to sign OCSP requests. Only requests sent to the OCSP responder identified by GSK_OCSP_URL are signed and not the ones selected from a certificate AIA extension. See Table 31 on page 805 for a list of valid 4-character signature algorithm pair specifications.	Default is 0401 (RSA with SHA256).
GSK_OCSP_REQUEST_SIGKEYLABEL	Specifies the label of the key used to sign OCSP requests. Only requests sent to the OCSP responder identified by GSK_OCSP_URL are signed.	Any key label. OCSP requests are not signed if a key label is not specified.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_OCSP_RESPONSE_SIGALG_PAIRS	Specifies a preference ordered list of hash and signature algorithm pair specifications that are sent on the OCSP request and may be used by the OCSP responder to select an appropriate algorithm for signing the OCSP response. The string consists of one or more 4-character values in order of preference for use. If specified, the OCSP response must be signed with one of these hash and signature algorithm pairs and if it is not, the OCSP response is rejected. It should be noted that not all OCSP responders support the preference ordered list and the OCSP response may be signed by a signature algorithm that was not specified. These signature algorithm pair specifications only have relevance when OCSP is enabled in the application. See Table 31 on page 805 for a list of valid 4-character signature algorithm pair specifications.	A value of NULL indicates that a preference ordered list is not sent to the OCSP responder. The default value is NULL.
GSK_OCSP_RESPONSE_TIMEOUT	Specifies the time in seconds to wait for a response from the OCSP responder server.	The valid time limits are 0 through 43200 seconds (12 hours). The default value is 15 seconds and a value of 0 indicates that there is no time limit.
GSK_OCSP_RETRIEVE_VIA_GET	Specifies if the HTTP GET method should be used when sending an OCSP request.	A value of 0 or OFF sends the OCSP request via the HTTP POST method. A value of 1 or ON sends the OCSP request via the HTTP GET method when the total request size after Base64 encoding is less than 255 bytes. The default value is OFF.

GSK_OCSP_URL	Specifies the URI of an OCSP responder. The OCSP responder is used to obtain certificate revocation status during certificate validation. A certificate does not need an AIA extension if a responder URL is configured using this option.
	
	If GSK_OCSP_URL is specified, GSK_OCSP_ENABLE is set to ON, and GSK_OCSP_URL_PRIORITY is set to ON, the order that responders are used is GSK_OCSP_URL defined responder first and then the responders identified in the AIA extension.
	
	If GSK_OCSP_URL is specified, GSK_OCSP_ENABLE is set to ON, and GSK_OCSP_URL_PRIORITY is set to OFF, the order that responders are used is the responders identified in the AIA extension first and then the GSK_OCSP_URL defined responder.
		The value must conform to the definition of an HTTP url:
		
		`http_URL = "http:" "//" host`
		`[ ":" port ] [ abs_path`
		`[ "?" query ]]`
		
		where *host* can be an IPv4 or IPv6 IP address, or a domain name.
		
		The default value is NULL.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_OCSP_URL_PRIORITY	Specifies the priority order for contacting OCSP responder locations if both GSK_OCSP_URL and GSK_OCSP_ENABLE are active.	A value of 1 or ON indicates that the order that responders are used is the GSK_OCSP_URL defined responder first and then the responders identified in the AIA extension. A value of 0 or OFF indicates that the order that responders are used is the responders identified in the AIA extension first and then the GSK_OCSP_URL defined responder. The default value is ON.
GSK_PEER_CERT_MIN_VERSION	Specifies that certificate validation should ensure that the partner's end-entity certificate is a minimum X.509 version. This setting is ignored during a TLS V1.3 handshake as TLS V1.3 requires a minimum X.509 certificate version of 3.	A value of 3 specifies that the partner’s end-entity certificate must be an X.509 version 3. A value of ANY specifies that the partner’s end entity certificate can be any supported System SSL X.509 version. The default value is ANY.
GSK_PEER_DH_MIN_KEY_SIZE	Specifies the minimum allowed X.509 certificate Diffie-Hellman key size for a peer end-entity certificate.	Valid values are 0 through 2048. The default value in non-FIPS mode is 1024. The default value in FIPS mode is 2048.
GSK_PEER_DSA_MIN_KEY_SIZE	Specifies the minimum allowed X.509 certificate DSA key size for a peer end-entity certificate.	Valid values are 0 through 2048. The default value in non-FIPS mode and FIPS mode is 1024.
GSK_PEER_ECC_MIN_KEY_SIZE	Specifies the minimum allowed X.509 certificate Elliptic Curve key size for a peer end-entity certificate.	Valid values are 0 through 521. The default value in non-FIPS mode and FIPS mode ON or LEVEL 1 is 192.
GSK_PEER_RSA_MIN_KEY_SIZE	Specifies the minimum allowed X.509 certificate RSA key size for a peer end-entity certificate.	Valid values are 0 through 4096. The default value in non-FIPS mode and FIPS mode ON or LEVEL 1 is 1024.
GSK_PROTOCOL_SSLV2	Specifies whether the SSL V2 protocol is supported. The SSL V2 and SSL V3 protocols should be disabled whenever possible because the TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 protocols provide significant security enhancements. This variable has no effect when operating in FIPS mode.	A value of 0, OFF or DISABLED disables the SSL V2 protocol while a value of 1, ON or ENABLED enables the SSL V2 protocol. The default value is OFF.
GSK_PROTOCOL_SSLV3	Specifies whether the SSL V3 protocol is supported. The SSL V2 and SSL V3 protocols should be disabled whenever possible because the TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 protocols provide significant security enhancements. This variable has no effect when operating in FIPS mode.	A value of 0, OFF or DISABLED disables the SSL V3 protocol while a value of 1, ON or ENABLED enables the SSL V3 protocol. The default value is OFF.
GSK_PROTOCOL_TLSV1	Specifies whether the TLS V1.0 protocol is supported.	A value of 0, OFF or DISABLED disables the TLS V1.0 protocol while a value of 1, ON or ENABLED enables the TLS V1.0 protocol. The default value is ON.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_PROTOCOL_TLSV1_1	Specifies whether the TLS V1.1 protocol is supported.	A value of 0, OFF or DISABLED disables the TLS V1.1 protocol while a value of 1, ON or ENABLED enables the TLS V1.1 protocol. The default value is OFF.
GSK_PROTOCOL_TLSV1_2	Specifies whether the TLS V1.2 protocol is supported.	A value of 0, OFF or DISABLED disables the TLS V1.2 protocol. A value of 1, ON or ENABLED enables the TLS V1.2 protocol. The default value is OFF.
GSK_PROTOCOL_TLSV1_3	Specifies whether the TLS V1.3 protocol is supported. **Note:** The TLS V1.3 protocol is not currently supported in FIPS mode. If an attempt is made to enable this protocol while running in FIPS mode, an error is returned during environment initialization.	A value of 0, OFF, or DISABLED disables the TLS V1.3 protocol. A value of 1, ON, or ENABLED enables the TLS V1.3 protocol. The default value is OFF.
GSK_REFERENCE_ID_CN	Specifies a list of CN values to compare against the server's certificate subject DN common name. For more information, see "Server certificate domain-based validation" on page 70.	Fully qualified domain names containing at least three labels. The list can be separated by a comma or a space. If any of the values contains either a comma, space, or backslash, it can be escaped with a backslash '\'. The maximum number of characters allowed is 16384. If any of the ID values ends in a period, the period will be removed prior to comparison.
GSK_REFERENCE_ID_DNS	Specifies a list of DNS values to compare against the server's subject alternative name DNS. For more information, see "Server certificate domain-based validation" on page 70.	Fully qualified domain names containing at least three labels. The list can be separated by a comma or a space. If any of the values contains either a comma, space or backslash, it can be escaped with a backslash '\'. The maximum number of characters allowed is 16384. If any of the ID values ends in a period, the period will be removed prior to comparison.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_RENEGOTIATION	Specifies the type of session renegotiation allowed for an SSL environment.	A value of NONE disables SSL V3 and TLS handshake renegotiation as a server and allow RFC 5746 renegotiation. This is the default. A value of DISABLED disables SSL V3 and TLS handshake renegotiation as a server and also disable RFC 5746 renegotiation. A value of ALL allows SSL V3 and TLS handshake renegotiation as a server while also allowing RFC 5746 renegotiation. A value of ABBREVIATED allows SSL V3 and TLS abbreviated handshake renegotiation as a server for resuming the current session only, while disabling SSL V3 and TLS full handshake renegotiation as a server. With this value specified, the System SSL session ID cache is not checked when resuming the current session. RFC 5746 renegotiation is allowed if this value is specified.
GSK_RENEGOTIATION_PEER_CERT_ CHECK	Specifies if the peer certificate is allowed to change during renegotiation.	A value of OFF or 0 does not perform an identity check against the peer's certificate during renegotiation. This allows the peer certificate to change during renegotiation. This is the default. A value of ON or 1 performs a comparison against the peer's certificate to ensure that certificate does not change during renegotiation.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_REVOCATION_SECURITY_LEVEL	Specifies the level of security to be used when contacting an OCSP responder or an HTTP server specified in a URI value of the CDP extension. An attempt to contact either an OCSP responder or HTTP server is performed when revocation information is not found in cache. To enforce contact with either the OCSP responder or HTTP server for each validation, caching must be disabled. For OCSP caching, see GSK_OCSP_CLIENT_CACHE_SIZE. For HTTP CRL caching, see GSK_HTTP_CDP_CACHE_SIZE.	A value of LOW indicates that certificate validation does not fail if the OCSP responder or HTTP server specified in the URI value of the CDP extension cannot be contacted. A value of MEDIUM requires the OCSP responder or the HTTP server in a URI value in the CDP extension to be contactable. For an OCSP responder, it must be able to provide a valid certificate revocation status. If the certificate status is revoked or unknown, certificate validation fails. For an HTTP server in a CDP extension, it must be contactable and able to provide an CRL. A value of HIGH requires revocation information to be provided by the OCSP responder or HTTP server. If OCSP revocation checking with the AIA extension is enabled, there must be HTTP URI values present in the certificate that are able to be contactable and able to provide a valid certificate revocation status. If HTTP CRL checking is enabled, there must be HTTP URI values in the CDP extension that are able to be contactable and able to provide a CRL. The default value is MEDIUM.
GSK_RNG_ALLOW_ZERO_BYTES	Specifies whether the SSL random number generator, gsk_generate_random_bytes includes bytes with a zero value in the random byte output stream, or remove them. The GSK_RNG_ALLOW_ZERO_BYTES environment variable is processed during System SSL initialization and is not checked afterward.	A value of TRUE, ON or 1 sets the random number generator to retain bytes with a zero value in the output stream. A value of FALSE, OFF or 0 results in bytes with a zero value being removed. The default setting is TRUE.
GSK_SERVER_ALLOWED_KEX_ECURVES	Specifies the list of elliptic curve specifications that are allowed by the server for the TLS V1.0, TLS V1.1, and TLS V1.2 server key exchange when using ECDHE-based cipher suites as a string consisting of one or more 4-character values. See Table 29 on page 804 for a list of valid 4-character elliptic curve and group specifications. For the TLS V1.3 protocol, this setting is ignored and the server allowed elliptic curve specifications and groups are defined by the GSK_SERVER_TLS_KEY_SHARES setting. When Suite B profile is defined, this setting is ignored. The server allowed elliptic curves are defined by the Suite B profile setting. For more information, see "Suite B cryptography support" on page 49.	The default specification is 00230024002500210019.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_SERVER_EPHEMERAL_DH_GROUP_SIZE	Specifies the minimum Diffie-Hellman group size to be used by the server for an ephemeral Diffie-Hellman key exchange.	A value of LEGACY specifies the Diffie-Hellman group size to be 1024 in non-FIPS mode and 2048 in FIPS mode. A value of 2048 specifies the Diffie-Hellman group size to be 2048. A value of MATCH specifies the Diffie-Hellman group size to be determined by the strength of the server's certificate. If the key being matched is less than or equal to key size 1024, group size 1024 is used. If the key size is greater than 1024, group size 2048 is used. The default value is LEGACY.
GSK_SERVER_EXTENDED_MASTER_SECRET	Specifies if the TLS server supports negotiating the extended master secret extension from clients. This option is only applicable for TLS V1.0, TLS V1.1, and TLS V1.2 handshakes.	A value of 0, OFF, or DISABLED specifies that the TLS server does not support negotiating the extended master secret extension from clients. A value of 1, ON, or ENABLED specifies that the TLS server supports negotiating the extended master secret extension from clients, but does not require the extension. A value of REQUIRED specifies that the TLS server requires negotiating the extended master secret extension from clients. If a client does not send the extended master secret extension, the handshake fails. Before setting this option to REQUIRED, ensure that all clients communicating with this server support the extended master secret extension. If the remote client partner is a z/OS System SSL application, it must be running z/OS V2R3 or later and have PTFs for APAR OA60105 (z/OS V2R3 and V2R4) applied and active before setting this option to REQUIRED. If the server is enabled for sysplex session ID caching (GSK_SYSPLEX_SIDCACHE is set to ON), all systems must be running z/OS V2R3 or later and any z/OS V2R3 or z/OS V2R4 systems must have the PTFs for APAR OA60105 applied and active before setting this option to REQUIRED for maximum compatibility. The default value is ON.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_SERVER_FALLBACK_SCSV	Specifies if the server accepts the TLS fallback Signaling Cipher Suite Value (SCSV) when the client's cipher list includes it during an SSL or TLS handshake. The SCSV indicates to the server that the client is attempting to fallback to an earlier TLS or SSL protocol version after a previous handshake attempt failed.	A value of ON or 1 indicates that the server supports the TLS fallback Signaling Cipher Suite Value (SCSV) when included in the client's supported cipher list during an SSL or TLS handshake. If the SCSV is present in the client's supported list and the TLS or SSL protocol level specified by the client during the handshake is less than the highest TLS or SSL protocol level supported by the server, the SSL or TLS handshake attempt fails. A value of OFF or 0 indicates that the server ignores the SCSV when included in the client's supported cipher list during an SSL or TLS handshake. This is the default setting.
GSK_SERVER_KEY_LABEL_LIST	Specifies 1 to 8 labels that are used to authenticate the server application in order of preference.	Multiple labels are delimited by a comma or a blank space. If a comma or a blank space appears in the label name, a backslash (\\) character must be used as an escape character. The maximum length of a label name is 127 characters not including the use of an escape character. The maximum number of label names allowed is 8. If GSK_KEY_LABEL is specified along with GSK_SERVER_KEY_LABEL_LIST, GSK_KEY_LABEL is used when an SSL V2 secure connection is being established. Otherwise, GSK_KEY_LABEL is ignored. The default value is NULL.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_SERVER_OCSP_STAPLING	Specifies if the server supports the retrieval of the OCSP responses for the server's end entity certificate or the server's certificate chain if the client specifies support for the OCSP responses in the TLS handshake. The client indicates support for the retrieval of the OCSP responses by including the Certificate Status Request or the Multiple Certificate Status Request TLS extensions in a TLS handshake message. The OCSP responses are retrieved by the server and are sent to the client as part of the TLS handshake. The client can then parse the OCSP responses to determine the revocation status of the server's end entity certificate or the server's certificate chain. The inclusion of the OCSP responses in a TLS handshake message is commonly referred to as OCSP stapling. The GSK_OCSP_URL or the GSK_OCSP_ENABLE settings must be specified prior to initializing the TLS environment. These settings are required in order to contact the desired OCSP responders to retrieve the OCSP responses for the server's certificates. **Notes:** • When OCSP stapling is enabled, additional processing time will be required by the server to contact the OCSP responder to retrieve the OCSP responses. • See “Enabling OCSP server stapling” on page 57 for information about the OCSP related options that are ignored or allowed when OCSP stapling is enabled.	A value of ENDENTITY indicates that the server will contact the configured OCSP responders to retrieve the OCSP response for the server's end entity certificate. A value of ANY indicates that the server will contact the configured OCSP responders to retrieve the OCSP responses for the server's end entity certificate or the server's certificate chain. If the negotiated handshake protocol is TLS V1.2 and earlier, the OCSP responses that are retrieved by the server and sent to the client depend on the Certificate Status Request and the Multiple Certificate Status Request extensions being present in the TLS handshake message from the client. If both extensions are specified by the client in a TLS V1.2 and earlier handshake, the Multiple Certificate Status Request extension takes precedence. If the negotiated handshake protocol is TLS V1.3, the Multiple Certificate Status Request extension is not supported and the Certificate Status Request extension allows for the retrieval of the OCSP response for only the server’s end entity certificate. A value of OFF indicates that the server is not enabled for contacting the configured OCSP responders to retrieve the OCSP responses for the server's end entity certificate or the server's certificate chain. The default value is OFF.
GSK_SERVER_TLS_KEY_SHARES	Specifies the list of the key share groups that are supported by the server during a TLS V1.3 handshake. During a TLS V1.3 handshake, the server uses the client’s preferred key share group order and selects a group that is in common with this list. The client and server use the selected group to encrypt and decrypt TLS V1.3 handshake messages. See Table 27 on page 799 for a list of valid 4-character key share specifications.	There is no default value. This setting must be specified when enabled for TLS V1.3.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_SESSION_TICKET_CLIENT_ENABLE	Specifies if the client supports caching session tickets received from a server after a TLS V1.3 handshake has completed and supports TLS V1.3 resumption attempts to the server. The GSK_V3_SESSION_TIMEOUT and GSK_V3_SIDCACHE_SIZE settings also must be set to values greater than 0 to allow client session ticket caching.	A value of 0, OFF, or DISABLED disables client caching of session tickets received from a server after a TLS V1.3 handshake has completed and does not support TLS V1.3 resumption attempts to the server. A value of 1, ON, or ENABLED enables client caching of session tickets received from a server after a TLS V1.3 handshake has completed and supports TLS V1.3 resumption attempts to the server. The default value is ON.
GSK_SESSION_TICKET_CLIENT_MAXSIZE	Specifies the maximum size in bytes of a session ticket that can be stored in the client session ticket cache. Setting the maximum session ticket size too small could implicitly disable session ticket caching on the client side.	The valid sizes are 0 through 2147483647. The default size is 8192 (8K). A value of 0 disables checking the session ticket size and allows a session ticket of any size.
GSK_SESSION_TICKET_CLIENT_MAXCACHED	Specifies the maximum number of session tickets that are allowed to be cached by the client for each unique TLS V1.3 session.	The valid maximum number of cached session tickets is 1 through 128 tickets and defaults to 8.
GSK_SESSION_TICKET_SERVER_ALGORITHM	Specifies the algorithm to be used by the server to encrypt and decrypt the session tickets used for TLS V1.3 session resumption.	Valid values are AESCBC128 and AESCBC256. The default value is AESCBC128.
GSK_SESSION_TICKET_SERVER_COUNT	Specifies the number of session tickets that will be sent by the server after the initial TLS V1.3 handshake has completed. Each subsequent resumed TLS V1.3 handshake will also send a single session ticket to replace the one used for resumption.	Valid values are 0 through 16. The default value is 2.
GSK_SESSION_TICKET_SERVER_ENABLE	Specifies if the server supports sending session tickets after a TLS V1.3 handshake has completed and if it will accept resumption attempts from the client.	A value of 0, OFF, or DISABLED disables TLS V1.3 server session resumption. A value of 1, ON, or ENABLED enables TLS V1.3 server session resumption. The default value is ON.
GSK_SESSION_TICKET_SERVER_KEY_REFRESH	Specifies the key refresh interval in seconds of the encryption key used by the server to encrypt session tickets. In order to encrypt and decrypt session tickets, GSK_SESSION_TICKET_SERVER _ENABLE must be ON and the server must be configured to send session tickets, either via GSK_SESSION_TICKET_SERVER _COUNT or via the GSK_SEND_SESSION_TICKET option in **gsk_secure_socket_misc()**. When the encryption key is refreshed and a new primary encryption key is generated, the former encryption key is retained as a secondary key that can be used only for decryption until the subsequent refresh occurs. When the ticket is decrypted, the server only accepts the ticket if the GSK_SESSION_TICKET_SERVER _TIMEOUT has not yet passed.	Valid values are 0 through 604800. The default value is 300. A value of 0 disables session ticket encryption key refresh.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_SESSION_TICKET_SERVER_TIMEOUT	Specifies the maximum time that a server accepts a TLS V1.3 session resumption request from the client measured in seconds from the initial handshake. The server will continue to generate new session tickets for each new resumed handshake until the timeout has been reached, provided GSK_SESSION_TICKET_SERVER_COUNT is greater than 0 and GSK_SESSION_TICKET_SERVER_ENABLE is set to ON. Each session ticket generated by the server will be valid until the timeout has passed. If sysplex session ticket caching is not enabled (GSK_SYSPLEX_SESSION_TICKET_CACHE is set to OFF) and GSK_SESSION_TICKET_SERVER_ENABLE is set to ON, the session ticket encryption key must be available when the client attempts TLS V1.3 resumption as the ticket needs to be decrypted. In this configuration, the GSK_SESSION_TICKET_SERVER_KEY_REFRESH value impacts the lifetime of a session ticket.	Valid values are 1 through 604800 seconds (seven days). If sysplex session ticket caching is not enabled (GSK_SYSPLEX_SESSION_TICKET_CACHE is set to OFF), the default session ticket timeout value is 300. If sysplex session ticket caching is enabled, (GSK_SYSPLEX_SESSION_TICKET_CACHE is set to ON), the default session ticket timeout value is 600.
GSK_SSL_HW_DETECT_MESSAGE	Setting this environment variable to 1 causes a series of messages to be written to stderr during System SSL initialization. These messages displays the current status of the hardware cryptographic support. These messages are intended for diagnostic use only and are not translated based on the setting of the LANG environment variable.	Specify 1 to have messages written. Any other value is ignored, which is the default.
GSK_SSL_ICSF_ERROR_MESSAGE	Setting this environment variable to 1 causes a message to be written to stderr when an ICSF callable service returns an error. These messages are intended for diagnostic use only and are not translated based on the setting of the LANG environment variable.	Specify 1 to have messages written. Any other value is ignored, which is the default.
GSK_STDERR_FILE	Specifies the fully-qualified name of the file to receive standard error messages generated using SSL message services. Messages displayed from externally documented messages is written to stderr if this environment variable is not defined.	If fully qualified file not specified, the default action is to write standard errors to stderr.
GSK_STDOUT_FILE	Specifies the fully-qualified name of the file to receive standard output messages generated using SSL message services. Messages displayed from externally documented messages is written to stdout if this environment variable is not defined.	If fully qualified file not specified, the default action is to write standard output to stdout.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_SUITE_B_PROFILE	Specifies the Suite B profile to be applied to TLS sessions. A Suite B compliant TLS V1.2 client must offer only the following cipher suites when conversing with a TLS V1.2 Suite B compliant server. 128-bit security level: • C023 = 128-bit AES encryption with SHA-256 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate. • C02B = 128-bit AES in Galois Counter Mode encryption with SHA-256 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate. 128-bit minimum security level: • C02B = 128-bit AES in Galois Counter Mode encryption with SHA-256 message authentication and ephemeral ECDH key. • C02C = 256-bit AES in Galois Counter Mode encryption with SHA-384 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate. 192-bit security level: • C024 = 256-bit AES encryption with SHA-384 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate. • C02C = 256-bit AES in Galois Counter Mode encryption with SHA-384 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate. 192-bit minimum security level: • C02C = 256-bit AES in Galois Counter Mode encryption with SHA-384 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate.	A value of OFF specifies that Suite B compliant profiles are not in use for TLS sessions. This is the default value. A value of 128 specifies that only ciphers defined within 128-bit Suite B compliant profile can be used for a TLS session. A value of 128MIN specifies that only AES-GCM ciphers defined within the 128-bit minimum Suite B compliant profile can be used for a TLS session. A value of 192 specifies that only ciphers defined within 192-bit Suite B compliant profile can be used for a TLS session. A value of 192MIN specifies that only the AES-GCM cipher defined within the 192-bit minimum Suite B compliant profile can be used for a TLS session. A value of ALL specifies that ciphers defined within both the 128-bit and 192-bit Suite B compliant profiles can be used for a TLS session.
GSK_SYSPLEX_SESSION_TICKET_CACHE	Specifies if sysplex session ticket caching for TLS V1.3 sessions is enabled for this server application.	A value of 0, OFF, or DISABLED specifies that sysplex session ticket caching for TLS V1.3 server sessions is not enabled. A value of 1, ON, or ENABLED specifies that sysplex session ticket caching for TLS V1.3 server sessions is enabled. The default value is OFF.
GSK_SYSPLEX_SIDCACHE	Specifies whether sysplex session caching for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 sessions is enabled for this server application.	A value of 0, OFF or DISABLED disables sysplex session caching while a value of 1, ON or ENABLED enables sysplex session caching. The default value is OFF.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_T61_AS_LATIN1	Specifies the character set for ASN.1 TELETEXSTRING conversions. The T.61 character set is supposed to be used for strings tagged as TELETEXSTRING. The X.690 ASN.1 definition specifies the 7-bit T.61character set (ISO IR-102). However, many certificate authorities issue certificates using the 8-bit ISO8859-1 character set (ISO IR-100) instead of the 7-bit T.61 character set. This causes conversion errors when the certificate is decoded. To add to the confusion, the 8-bit T.61 character set (ISO IR-103) is also used by some implementations.	If the GSK_T61_AS_LATIN1 environment variable is set to YES or 1, the 8-bit ISO8859-1 character set is used when processing a TELETEX string. If the GSK_T61_AS_LATIN1 environment variable is set to NO or 0, the 8-bit T.61 character set is used. The default is to use the ISO8859-1 character set. The GSK_T61_AS_LATIN1 environment variable is processed during System SSL initialization and is not checked afterward. Note that selecting the incorrect character set can cause strings to be converted incorrectly.
GSK_TLS_CBC_PROTECTION_METHOD	Specifies an optional SSL V3.0 or TLS V1.0 CBC IV protection method when writing application data.	A value of NONE indicates that no CBC protection is enabled. This is the default. A value of ZEROBYTEFRAGMENT indicates that zero byte record fragmenting is enabled. When this value is specified, a zero byte record fragment is sent before the application data records are sent. A value of ONEBYTEFRAGMENT indicates that one byte record fragmenting is enabled. When this value is specified, the first record is sent in two record fragments with the first record fragment containing only one byte of application data. The rest of the application data in the first record is sent in the second record fragment. All following records are written whole.
GSK_TLS_CERT_SIG_ALG_PAIRS	Specifies the list of hash and signature algorithm pair specifications that are supported by the client or server as a string consisting of one or more 4-character values in order of preference for use in digital signatures of X.509 certificates. The certificate signature algorithm pair specifications are sent by either the client or server to the session partner to indicate which signature/hash algorithm combinations are supported for digital signatures in X.509 certificates. The GSK_TLS_CERT_SIG_ALG_PAIRS setting overrides the GSK_TLS_SIG_ALG_PAIRS setting when checking the digital signatures of the remote peer's X.509 certificates. The certificate signature algorithm pair specification only has relevance for TLS V1.2 client or TLS V1.3 client and server sessions. See Table 30 on page 804 for a list of valid 4-character certificate signature algorithm pair specifications.	There is no default value. If not specified, the GSK_TLS_SIG_ALG_PAIRS setting is used to indicate the signature/hash algorithm combinations that are supported by digital signatures in X.509 certificates.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_TLS_SIG_ALG_PAIRS	Specifies the list of hash and signature algorithm pair specifications supported by the client or server as a string consisting of one or more 4-character values in order of preference for use in digital signatures of X.509 certificates and TLS handshake messages. The signature algorithm pair specifications are sent by either the client or server to the session partner to indicate which signature/hash algorithm combinations are supported for digital signatures in X.509 certificates and TLS handshake messages. If the GSK_TLS_CERT_SIG_ALG_PAIRS setting is specified, the GSK_TLS_SIG_ALG_PAIRS setting is only used to indicate the signature/hash algorithm combinations supported for digital signatures in TLS handshake messages. The signature algorithm pair specification only has relevance for sessions using TLS V1.2 or higher protocols. See Table 30 on page 804 for a list of valid 4-character signature algorithm pair specifications.	The default value is: "0601060305010503040104030402030103030302020102030202". If TLS V1.3 is enabled, 080608050804 is appended to the end of the default list.
GSK_TRACE	Specifies a bit mask enabling System SSL trace options. No trace option is enabled if the bit mask is 0 and all trace options are enabled if the bit mask is 0xffff. The bit mask can be specified as a decimal (nnn), octal (0nnnn) or hexadecimal (0xhh) value.	These trace options are available: 0x01 = Trace function entry 0x02 = Trace function exit 0x04 = Trace errors 0x08 = Include informational messages 0x10 = Include EBCDIC data dumps 0x20 = Include ASCII data dumps The default value is 0x00.
GSK_TRACE_FILE	Specifies the name of the trace file. The **gsktrace** command is used to format the trace file. The trace file is not used if the GSK_TRACE environment variable is not defined or is set to 0. The current process identifier is included as part of the trace file name when the name contains a percent sign (%). For example, if GSK_TRACE_FILE is set to /tmp/gskssl.%.trc and the current process identifier is 247, then the trace file name is /tmp/gskssl.247.trc.	Must be set to the name of an UNIX System Services file in a directory for which the executing application has write permission. The default trace file is /tmp/gskssl.%.trc.
GSK_V2_CIPHER_SPECS	Specifies the SSL V2 cipher specifications in order of preference as a string consisting of 1 or more 1-character values. See Table 25 on page 795 for the list of the supported ciphers. **Note:** If Suite B support is enabled in the SSL environment, the SSL V2 cipher specifications are ignored.	If United States only encryption is enabled (System SSL Security Level 3 FMID or CPACF Feature 3863 is installed), the default is 34. Otherwise, the default is 4.
GSK_V2_SESSION_TIMEOUT	Specifies the session timeout value in seconds for the SSL V2 protocol.	The valid timeout values are 0 through 100, default value is 100.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_V2_SIDCACHE_SIZE	Specifies the number of session identifiers that can be contained in the SSL V2 cache.	The valid cache sizes are 0 through 32000 and defaults to 256. The SSL V2 cache is disabled if 0 is specified. The session identifier cache is allocated using the requested size rounded up to a power of 2 with a minimum size of 16.
GSK_V3_CIPHER_SPECS	Specifies the SSL V3 cipher specifications as a string consisting of one or more 2-character values. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 protocols. Any ciphers which are not supported by the protocol being negotiated are ignored and not used. See Table 26 on page 795 for the list of the supported 2-character cipher specifications for each protocol. **Note:** If Suite B support is enabled, the 2-character cipher specifications are ignored.	If executing in non-FIPS mode and United States only encryption is enabled (System SSL Security Level 3 FMID or CPACF Feature 3863 is installed), the default is: "3538392F3233" If executing in non-FIPS mode and United States only encryption is not enabled (System SSL Security Level 3 FMID and CPACF Feature 3863 are not installed), the default is: "" (empty string - no default) If executing in FIPS mode, the default is: "3538392F3233"
GSK_V3_CIPHER_SPECS_EXPANDED	Specifies the SSL V3 or TLS cipher specifications in order of preference as a string consisting of one or more 4-character values. The SSL V3 cipher specifications are used for the SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 protocols. Any ciphers which are not supported by the protocol being negotiated are ignored and not used. See Table 26 on page 795 for the list of the supported 4-character cipher specifications for SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3. **Note:** If Suite B support is enabled, the 4-character cipher specifications are ignored.	If executing in non-FIPS mode and United States only encryption is enabled (System SSL Security Level 3 FMID or CPACF Feature 3863 is installed), the default is: "003500380039002F00320033" If executing in non-FIPS mode and United States only encryption is not enabled (System SSL Security Level 3 FMID and CPACF Feature 3863 are not installed), the default is: "" (empty string – no default) If executing in FIPS mode, the default is: "003500380039002F00320033"
GSK_V3_SESSION_TIMEOUT	Specifies the session timeout value in seconds for the SSL V3 to TLS V1.2 session identifiers and TLS V1.3 session tickets in the cache. This is the number of seconds until an SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 session identifier or TLS V1.3 session ticket expires. System SSL keeps the SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 session identifiers or TLS V1.3 session tickets for this amount of time in the cache. This reduces the amount of data exchanged during the SSL/TLS handshake when a complete initial handshake has already been performed. Session identifiers and session tickets are not kept if a value of 0 is specified.	The range is 0-86400 and defaults to 86400. The timeout is disabled if 0 is specified.

Table 23. SSL-Specific environment variables (continued)

Environment variables	Usage	Valid values
GSK_V3_SIDCACHE_SIZE	Specifies the size in number of entries in the SSL V3 to TLS V1.2 session identifier and TLS V1.3 session ticket cache. The oldest entry will be removed when the cache is full in order to add a new entry. Session identifiers and session tickets are not remembered if a value of 0 is specified. For the SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 protocols, the cache stores session identifiers for use on the server and client sides. For the TLS V1.3 protocol on the client side, the cache is used to store session tickets when GSK_SESSION_TICKET_CLIENT _ENABLE is set to ON.	The valid cache sizes are 0 through 64000 and defaults to 512. The SSL V3 to TLS V1.2 session identifier and TLS V1.3 session ticket cache is disabled if 0 is specified. The session identifier and session ticket cache is allocated by using the requested size rounded up to a power of 2 with a minimum size of 16.
GSKV2CACHESIZE	Used to control the size limit for a V2 session cache. This variable is for use only with the deprecated API set.	The valid cache sizes are 0 through 32000 and defaults to 256.
GSKV3CACHESIZE	Used to control the size limit for a V3 session cache. This variable is for use only with the deprecated API set.	The valid cache sizes are 0 through 64000 and defaults to 512 entries.
GSK_WILDCARD_VALIDATION_ENABLE	Specifies whether the reference list server validation supports the asterisk as the wildcard character to replace zero or more characters within the server's certificate subject alternative name DNS or subject DN common name value or values.	A value of 1, ON, or ENABLED indicates that the client will accept an asterisk as the wildcard character when checking the server's certificate. A value of 0, OFF, or DISABLED indicates that the client will not accept an asterisk as the wildcard character when checking the server's certificate. The default value is OFF.

Table 24 on page 788 contains system environment variables used by SSL.

Table 24. System environment variables used by SSL

System environment variables	Usage	Valid values
LIBPATH	Used to specify the directory to search for a DLL (Dynamic Link Library) file name. If it is not set, the working directory is searched.	
NLSPATH	Specifies where the message catalogs are to be found.	The default location is /usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/En_US.IBM-1047/%N
PATH	Contains a list of directories that the system searches to find executable commands. Directories in this list are separated with colons. Searches each directory in the order specified in the list until it finds a matching executable. If you want the shell to search the working directory, put a null string in the list of directories (for example, to tell the shell to search the working directory first, start the list with a colon or semicolon).	

Table 24. System environment variables used by SSL (continued)

System environment variables	Usage	Valid values
STEPLIB	Identifies a STEPLIB variable to be used in building a process image for running an executable file. A STEPLIB is a set of private libraries used to store a new or test version of an application program, such as a new version of a runtime library.	STEPLIB can be set to the values CURRENT or NONE or to a list of z/OS data set names. The default is CURRENT, which passes on the TASKLIB, STEPLIB, or JOBLIB allocations that are part of the invoker's z/OS program search order environment to the process image created for an executable file. The value NONE indicates that you do not want a STEPLIB environment for executable files. You can specify up to 255 z/OS data set names, separated by colons, as a list of data sets used to build a STEPLIB variable.

Appendix B. Sample C++ SSL files

A sample set of files is shipped to provide an example of what is needed to build a C++ System SSL application. These files build one DLL (SECURES) and five programs: **client**, **client_tls**, **server**, **server_tls**, and **display_certificate**. These sample files are in `/usr/lpp/gskssl/examples`:

- Makefile
- client.cpp
- client_tls.cpp
- server.cpp
- server_tls.cpp
- common.hpp
- common.cpp
- secures.h
- secures.cpp
- utils.hpp
- utils.cpp
- display_certificate.c

Note: Reference the sample source for SSL environment and connection attributes. File name and password attributes are hard-coded in the sample files.

server (source file: server.cpp) is a multithreaded program that opens a socket on IP address 127.0.0.1, port 4321 and listens for client requests. **server** can run in either secure (using SSL) mode or nonsecure (using normal socket reads and writes) mode. In secure mode, SSL V2, SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 are enabled by default in **server** unless the -protocol parameter is specified. By default, **server** runs with one socket listen thread and 20 work threads. The socket listen thread listens for connections from clients and puts each request onto the work list. The work threads check the work list for work and then perform the work. The number of work threads can be specified using the -numthreads parameter when starting **server**.

To get information about the parameters accepted when invoking the server program, issue `server -?`

server_tls (source file: server_tls.cpp) is a multithreaded program that opens a socket on IP address 127.0.0.1, port 4321 and listens for client requests. **server_tls** can run in either secure (using SSL) mode or nonsecure (using normal socket reads and writes) mode. In secure mode, the -protocol parameter must be specified which allows enablement of one or more of the following protocols: TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3. By default, **server_tls** runs with one socket listen thread and 20 work threads. The socket listen thread listens for connections from clients and puts each request onto the work list. The work threads check the work list for work and then perform the work. The number of work threads can be specified using the -numthreads parameter when starting **server_tls**.

To get information about the parameters accepted when invoking the server_tls program, issue `server_tls -?`

client (source file: client.cpp) is a single threaded program that connects to the **server** or **server_tls** program and exchanges one or more data packets. **client** can also run in secure or nonsecure mode, but its mode must match the mode of the server to which it is connecting. In secure mode, SSL V2, SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 are enabled by default in **client** unless the -protocol parameter is specified. The number of connections, the number of read/write packets per connection, the number of bytes in each write packet, and the number of bytes in each read packet can be specified. Multiple clients can be run simultaneously to the same server.

To get information about the parameters accepted when invoking the client program, issue `client -?`

client_tls (source file: client_tls.cpp) is a single threaded program that connects to the **server** or **server_tls** program and exchanges one or more data packets. **client_tls** can also run in secure or nonsecure mode, but its mode must match the mode of the server to which it is connecting. In secure mode, the -protocol parameter must be specified which allows enablement of one or more of the following protocols: TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3. The number of connections, the number of read/write packets per connection, the number of bytes in each write packet, and the number of bytes in each read packet can be specified. Multiple clients can be run simultaneously to the same server.

To get information about the parameters accepted when invoking the client_tls, issue `client_tls -?`

display_certificate (source file: display_certificate.c) is a program that can display an X.509 certificate stored in a file. The **display_certificate** program is only supported as a 31-bit application.

The files included in the examples are:

Makefile
: This file builds the example programs and DLLs in the z/OS shell. The resulting executable DLLs are **client**, **client_tls**, **server**, **server_tls**, and **display_certificate**.

  To build the examples as a 31-bit application (default), issue:

```
/bin/make
```

  To build the client, client_tls, server, and server_tls examples as a 64-bit application, issue:

```
/bin/make AMODE=64
```

  Remove all compiled *.o* and *.x* artifacts, issue:

```
/bin/make clean
```

  Remove all compiled *.o*, *.x* and DLL artifacts, issue:

```
/bin/make clobber
```

  Note: For an alternative to building in the z/OS shell, see "Sample JCL to compile and run an SSL application outside of the z/OS shell" on page 793.

client.cpp
: This file contains the routines that implement the client function.

client_tls.cpp
: This file contains the routines that implement the client_tls function.

server.cpp
: This file contains the routines that implement the server function.

server_tls.cpp
: This file contains the routines that implement the server_tls function.

common.hpp
: This contains the prototypes and defines for the routines in common.cpp.

common.cpp
: This file contains a set of routines called by client, client_tls, server, and server_tls to set up, accept, open, and close connections, and to read and write data. All data that is read or written in the form of packets that contain a header containing a command, length, and cookie. This implements a higher level communication protocol used between the client and server programs. For example, this higher level protocol allows the client to send a "STOP" request to the server, which stops the server program.

secures.h
: This file contains prototypes and defines for the routines in secures.cpp.

secures.cpp
This file implements a set of APIs that are similar to the normal sockets APIs, except that the routines work in either secure (SSL) or nonsecure mode. These routines are called by code in client.cpp, client_tls.cpp, server.cpp, server_tls.cpp and common.cpp.

utils.hpp
This file contains the prototype for the routine in utils.cpp, some structure definitions, and several defined constants.

utils.cpp
This file contains routines that server, server_tls, client, and client_tls programs use to check command line options.

display_certificate.c
This file is a sample program to decode and display an X.509 certificate.

Sample JCL to compile and run an SSL application outside of the z/OS shell

The following sample JCL can be used as an alternative to compiling inside the z/OS shell:

```
//MYLIBS1  JCLLIB  ORDER=CBC.SCCNPRC
//STEP1    EXEC  EDCCBG,
//         BPARM='MAP,XREF,DYNAM(DLL)',
//         GPARM='//tmp/examples/userCert.b64',
//  CPARM='SOURCE,LIST,DEFINE(MVS),LONGNAME,PHASEID,RENT,DLL,EXPMAC,DEF
//             INE(_XOPEN_SOURCE_EXTENDED)'
//COMPILE.SYSLIB DD DSN=SYS1.SIEAHDR.H,DISP=SHR
//COMPILE.SYSIN  DD DSN=LDANIEL.EXAMPLE.PDS(DISPCERT),DISP=SHR
//BIND.IMP       DD DSN=SYS1.SIEASID,DISP=SHR
//BIND.SYSIN     DD *
 INCLUDE IMP(GSKSSL)
 INCLUDE IMP(GSKCMS31)
 ENTRY CEESTART
 NAME DISPCERT(R)
//BIND.SYSLMOD DD DSN=LDANIEL.CCODE.PDSE(DISPCERT),DISP=MOD
//GO.SYSUDUMP  DD SYSOUT=*
//GO.CAOESTOP  DD DUMMY
//GO.SYSPRINT  DD DISP=SHR,DSN=LDANIEL.RESULT.PDS(DISPCERT)
```

An existing binary or base64 encoded certificate file can be displayed with this JCL.

In this example, the file to display is /tmp/examples/userCert.b64. You can substitute any existing certificate file path and name. Or, create a certificate with the **gskkyman** utility and export the certificate in base64 format to /tmp/examples/userCert.b64.

In this example, the C source code is in 'LDANIEL.EXAMPLE.PDS(DISPCERT). You can substitute an existing PDS name.

To get the sample C code from /usr/lpp/gskssl/example/display_certificate.c into a PDS, you can use the oget TSO command. The PDS must already be allocated. Here is an example:

```
oget '/usr/lpp/gskssl/examples/display_certificate.c' 'LDANIEL.EXAMPLE.PDS(DISPCERT)'
```

- The code is compiled into LDANIEL.CCODE.PDSE(DISPCERT). You can substitute an existing PDSE name. If the member does not exist, it creates it. If it does exist, it replaces it.
- The certificate is displayed in LDANIEL.RESULT.PDS(DISPCERT). You can substitute an existing PDS name. If the member does not exist, it creates it. If it does exist, it replaces it.

Appendix C. Cipher suite definitions

The following tables outline:

- Cipher suite definitions for SSL V2.
- 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3.
- Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 by supported protocol, symmetric algorithm, and message authentication algorithm.
- Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 by key-exchange method and signing certificate.
- Supported elliptic curve (group) definitions for TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 and supported key share definitions for TLS V1.3.

Note: When executing in non-FIPS mode, if either the System SSL Security Level 3 FMID is installed or the CPACF Feature 3863 is installed, the ciphers listed under the Security Level 3 FMID column are allowed to be used.

Table 25. Cipher suite definitions for SSL V2

Cipher number	Description	FIPS 140-2	Base security level FMID HCPT450	Security level 3 FMID JCPT451
1	128-bit RC4 encryption with MD5 message authentication (128-bit secret key)			X
2	128-bit RC4 export encryption with MD5 message authentication (40-bit secret key)		X	X
3	128-bit RC2 encryption with MD5 message authentication (128-bit secret key)			X
4	128-bit RC2 export encryption with MD5 message authentication (40-bit secret key)		X	X
6	56-bit DES encryption with MD5 message authentication (56-bit secret key)		X	X
7	168-bit Triple DES encryption with MD5 message authentication (168-bit secret key)			X

Note: When executing in non-FIPS mode, if either the System SSL Security Level 3 FMID is installed or the CPACF Feature 3863 is installed, the ciphers listed under the Security Level 3 FMID column are allowed to be used.

Table 26. 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3

2-character cipher number	4-character cipher number	Short name	Description [1]	FIPS 140-2	Base security level FMID HCPT450	Security level 3 FMID JCPT451
00	0000	TLS_NULL_WITH_NULL_NULL	No encryption or message authentication and RSA key exchange		X	X
01	0001	TLS_RSA_WITH_NULL_MD5	No encryption with MD5 message authentication and RSA key exchange		X	X
02	0002	TLS_RSA_WITH_NULL_SHA	No encryption with SHA-1 message authentication and RSA key exchange		X	X
03	0003	TLS_RSA_EXPORT_WITH_RC4_40_MD5	40-bit RC4 encryption with MD5 message authentication and RSA (export) key exchange		X	X
04	0004	TLS_RSA_WITH_RC4_128_MD5	128-bit RC4 encryption with MD5 message authentication and RSA key exchange			X
05	0005	TLS_RSA_WITH_RC4_128_SHA	128-bit RC4 encryption with SHA-1 message authentication and RSA key exchange			X
06	0006	TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	40-bit RC2 encryption with MD5 message authentication and RSA (export) key exchange		X	X
09	0009	TLS_RSA_WITH_DES_CBC_SHA	56-bit DES encryption with SHA-1 message authentication and RSA key exchange		X	X

Table 26. 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 (continued)

2-character cipher number	4-character cipher number	Short name	Description [1]	FIPS 140-2	Base security level FMID HCPT450	Security level 3 FMID JCPT451
0A	000A	TLS_RSA_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and RSA key exchange	X		X
0C	000C	TLS_DH_DSS_WITH_DES_CBC_SHA	56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate		X	X
0D	000D	TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate	X		X
0F	000F	TLS_DH_RSA_WITH_DES_CBC_SHA	56-bit DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate		X	X
10	0010	TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	X		X
12	0012	TLS_DHE_DSS_WITH_DES_CBC_SHA	56-bit DES encryption with SHA-1message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate		X	X
13	0013	TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate	X		X
15	0015	TLS_DHE_RSA_WITH_DES_CBC_SHA	56-bit DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate		X	X
16	0016	TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	X		X
2F	002F	TLS_RSA_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and RSA key exchange	X		X
30	0030	TLS_DH_DSS_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate	X		X
31	0031	TLS_DH_RSA_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	X		X
32	0032	TLS_DHE_DSS_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate	X		X
33	0033	TLS_DHE_RSA_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	X		X
35	0035	TLS_RSA_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and RSA key exchange	X		X
36	0036	TLS_DH_DSS_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate	X		X
37	0037	TLS_DH_RSA_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	X		X
38	0038	TLS_DHE_DSS_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate	X		X
39	0039	TLS_DHE_RSA_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	X		X
3B	003B	TLS_RSA_WITH_NULL_SHA256	No encryption with SHA-256 message authentication and RSA key exchange		X	X
3C	003C	TLS_RSA_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and RSA key exchange	X		X
3D	003D	TLS_RSA_WITH_AES_256_CBC_SHA256	256-bit AES encryption with SHA-256 message authentication and RSA key exchange	X		X
3E	003E	TLS_DH_DSS_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate	X		X

Table 26. 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 (continued)

2-character cipher number	4-character cipher number	Short name	Description [1]	FIPS 140-2	Base security level FMID HCPT450	Security level 3 FMID JCPT451
3F	003F	TLS_DH_RSA_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	X		X
40	0040	TLS_DHE_DSS_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate	X		X
67	0067	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	X		X
68	0068	TLS_DH_DSS_WITH_AES_256_CBC_SHA256	256-bit AES encryption with SHA-256 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate	X		X
69	0069	TLS_DH_RSA_WITH_AES_256_CBC_SHA256	256-bit AES encryption with SHA-256 message authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	X		X
6A	006A	TLS_DHE_DSS_WITH_AES_256_CBC_SHA256	256-bit AES encryption with SHA-256 message authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate	X		X
6B	006B	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	256-bit AES encryption with SHA-256 message authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	X		X
9C	009C	TLS_RSA_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and RSA key exchange	X		X
9D	009D	TLS_RSA_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and RSA key exchange	X		X
9E	009E	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	X		X
9F	009F	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and ephemeral Diffie-Hellman key exchange signed with an RSA certificate	X		X
A0	00A0	TLS_DH_RSA_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	X		X
A1	00A1	TLS_DH_RSA_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and fixed Diffie-Hellman key exchange signed with an RSA certificate	X		X
A2	00A2	TLS_DHE_DSS_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate	X		X
A3	00A3	TLS_DHE_DSS_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and ephemeral Diffie-Hellman key exchange signed with a DSA certificate	X		X
A4	00A4	TLS_DH_DSS_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate	X		X
A5	00A5	TLS_DH_DSS_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate	X		X
	1301	TLS_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and HKDF (HMAC-based Extract-and-Expand Key Derivation Function) with SHA256			X
	1302	TLS_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and HKDF (HMAC-based Extract-and-Expand Key Derivation Function) with SHA384			X

Table 26. 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 (continued)

2-character cipher number	4-character cipher number	Short name	Description [1]	FIPS 140-2	Base security level FMID HCPT450	Security level 3 FMID JCPT451
	1303	TLS_CHACHA20_POLY1305_SHA256	ChaCha20 encryption with 256-bit AEAD authentication and HKDF (HMAC-based Extract-and-Expand Key Derivation Function) with SHA256			X
	C001	TLS_ECDH_ECDSA_WITH_NULL_SHA	NULL encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an ECDSA certificate		X	X
	C002	TLS_ECDH_ECDSA_WITH_RC4_128_SHA	128-bit RC4 encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an ECDSA certificate			X
	C003	TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an ECDSA certificate	X		X
	C004	TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an ECDSA certificate	X		X
	C005	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an ECDSA certificate	X		X
	C006	TLS_ECDHE_ECDSA_WITH_NULL_SHA	NULL encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate		X	X
	C007	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	128-bit RC4 encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate			X
	C008	TLS_ECDHE_ECDSA_WITH_3DES_EDE_ CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate	X		X
	C009	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate	X		X
	C00A	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate	X		X
	C00B	TLS_ECDH_RSA_WITH_NULL_SHA	NULL encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an RSA certificate		X	X
	C00C	TLS_ECDH_RSA_WITH_RC4_128_SHA	128-bit RC4 encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an RSA certificate			X
	C00D	TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an RSA certificate	X		X
	C00E	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an RSA certificate	X		X
	C00F	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and fixed ECDH key exchange signed with an RSA certificate	X		X
	C010	TLS_ECDHE_RSA_WITH_NULL_SHA	NULL encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an RSA certificate		X	X
	C011	TLS_ECDHE_RSA_WITH_RC4_128_SHA	128-bit RC4 encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an RSA certificate			X
	C012	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	168-bit Triple DES encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an RSA certificate	X		X
	C013	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	128-bit AES encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an RSA certificate	X		X
	C014	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	256-bit AES encryption with SHA-1 message authentication and ephemeral ECDH key exchange signed with an RSA certificate	X		X
	C023	TLS_ECDHE_ECDSA_WITH_AES_128_ CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate	X		X

Table 26. 2-character and 4-character cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 (continued)

2-character cipher number	4-character cipher number	Short name	Description [1]	FIPS 140-2	Base security level FMID HCPT450	Security level 3 FMID JCPT451
	C024	TLS_ECDHE_ECDSA_WITH_AES_256_ CBC_SHA384	256-bit AES encryption with SHA-384 message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate	X		X
	C025	TLS_ECDH_ECDSA_WITH_AES_128_ CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and fixed ECDH key exchange signed with an ECDSA certificate	X		X
	C026	TLS_ECDH_ECDSA_WITH_AES_256_ CBC_SHA384	256-bit AES encryption with SHA-384 message authentication and fixed ECDH key exchange signed with an ECDSA certificate	X		X
	C027	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and ephemeral ECDH key exchange signed with an RSA certificate	X		X
	C028	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	256-bit AES encryption with SHA-384 message authentication and ephemeral ECDH key exchange signed with an RSA certificate	X		X
	C029	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256	128-bit AES encryption with SHA-256 message authentication and fixed ECDH key exchange signed with an RSA certificate	X		X
	C02A	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384	256-bit AES encryption with SHA-384 message authentication and fixed ECDH key exchange signed with an RSA certificate	X		X
	C02B	TLS_ECDHE_ECDSA_WITH_AES_128_ GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD authentication and ephemeral ECDH key exchange signed with an ECDSA certificate	X		X
	C02C	TLS_ECDHE_ECDSA_WITH_AES_256_ GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD message authentication and ephemeral ECDH key exchange signed with an ECDSA certificate	X		X
	C02D	TLS_ECDH_ECDSA_WITH_AES_128_ GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD message authentication and fixed ECDH key exchange signed with an ECDSA certificate	X		X
	C02E	TLS_ECDH_ECDSA_WITH_AES_256_ GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD message authentication and fixed ECDH key exchange signed with an ECDSA certificate	X		X
	C02F	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD message authentication and ephemeral ECDH key exchange signed with an RSA certificate	X		X
	C030	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD message authentication and ephemeral ECDH key exchange signed with an RSA certificate	X		X
	C031	TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256	128-bit AES in Galois Counter Mode encryption with 128-bit AEAD message authentication and fixed ECDH key exchange signed with an RSA certificate	X		X
	C032	TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384	256-bit AES in Galois Counter Mode encryption with 128-bit AEAD message authentication and fixed ECDH key exchange signed with an RSA certificate	X		X

[1] See Table 28 on page 802 for more information about the signing algorithm required for the key exchanges.

Table 27. Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 by supported protocol, symmetric algorithm, and message authentication algorithm

Cipher suite		Protocol support					Symmetric algorithm							Message MAC				
4 Char	2 Char	SSL V3	TLS V1.0	TLS V1.1	TLS V1.2	TLS V1.3	RC2 or RC4	DES or 3DES	AES-CBC 128	AES-CBC 256	AES-GCM 128	AES-GCM 256	Cha-Cha Poly 1305	MD5	SHA 1	SHA 256	SHA 384	AEAD
0000	00	X	X	X	X													
0001	01	X	X	X	X									X				
0002	02	X	X	X	X										X			

Table 27. Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 by supported protocol, symmetric algorithm, and message authentication algorithm (continued)

Cipher suite		Protocol support					Symmetric algorithm							Message MAC				
4 Char	**2 Char**	**SSL V3**	**TLS V1.0**	**TLS V1.1**	**TLS V1.2**	**TLS V1.3**	**RC2 or RC4**	**DES or 3DES**	**AES-CBC 128**	**AES-CBC 256**	**AES-GCM 128**	**AES-GCM 256**	**Cha-Cha Poly 1305**	**MD5**	**SHA 1**	**SHA 256**	**SHA 384**	**AEAD**
0003	03	X	X				RC4							X				
0004	04	X	X	X	X		RC4							X				
0005	05	X	X	X	X		RC4								X			
0006	06	X	X				RC2							X				
0009	09	X	X	X				DES							X			
000A	0A	X	X	X	X			3DES							X			
000C	0C	X	X	X				DES							X			
000D	0D	X	X	X	X			3DES							X			
000F	0F	X	X	X				DES							X			
0010	10	X	X	X	X			3DES							X			
0012	12	X	X	X				DES							X			
0013	13	X	X	X	X			3DES							X			
0015	15	X	X	X				DES							X			
0016	16	X	X	X	X			3DES							X			
002F	2F	X	X	X	X				X						X			
0030	30	X	X	X	X				X						X			
0031	31	X	X	X	X				X						X			
0032	32	X	X	X	X				X						X			
0033	33	X	X	X	X				X						X			
0035	35	X	X	X	X					X					X			
0036	36	X	X	X	X					X					X			
0037	37	X	X	X	X					X					X			
0038	38	X	X	X	X					X					X			
0039	39	X	X	X	X					X					X			
003B	3B				X											X		
003C	3C				X				X							X		
003D	3D				X					X						X		
003E	3E				X				X							X		
003F	3F				X				X							X		
0040	40				X				X							X		
0067	67				X				X							X		
0068	68				X					X						X		
0069	69				X					X						X		
006A	6A				X					X						X		
006B	6B				X					X						X		
009C	9C				X						X							X
009D	9D				X							X						X
009E	9E				X						X							X
009F	9F				X							X						X
00A0	A0				X						X							X
00A1	A1				X							X						X
00A2	A2				X						X							X
00A3	A3				X							X						X
00A4	A4				X						X							X

Table 27. Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 by supported protocol, symmetric algorithm, and message authentication algorithm (continued)

Cipher suite		**Protocol support**					**Symmetric algorithm**							**Message MAC**				
4 Char	**2 Char**	**SSL V3**	**TLS V1.0**	**TLS V1.1**	**TLS V1.2**	**TLS V1.3**	**RC2 or RC4**	**DES or 3DES**	**AES-CBC 128**	**AES-CBC 256**	**AES-GCM 128**	**AES-GCM 256**	**Cha-Cha Poly 1305**	**MD5**	**SHA 1**	**SHA 256**	**SHA 384**	**AEAD**
00A5	A5				X							X						X
1301						X					X							X
1302						X						X						X
1303						X							X					X
C001			X	X	X										X			
C002			X	X	X		RC4								X			
C003			X	X	X			3DES							X			
C004			X	X	X				X						X			
C005			X	X	X					X					X			
C006			X	X	X										X			
C007			X	X	X		RC4								X			
C008			X	X	X			3DES							X			
C009			X	X	X				X						X			
C00A			X	X	X					X					X			
C00B			X	X	X										X			
C00C			X	X	X		RC4								X			
C00D			X	X	X			3DES							X			
C00E			X	X	X				X						X			
C00F			X	X	X					X					X			
C010			X	X	X										X			
C011			X	X	X		RC4								X			
C012			X	X	X			3DES							X			
C013			X	X	X				X						X			
C014			X	X	X					X					X			
C023					X				X							X		
C024					X					X							X	
C025					X				X							X		
C026					X					X							X	
C027					X				X							X		
C028					X					X							X	
C029					X				X							X		
C02A					X					X							X	
C02B					X						X							X
C02C					X							X						X
C02D					X						X							X
C02E					X							X						X
C02F					X						X							X
C030					X							X						X
C031					X						X							X
C032					X							X						X

Table 28. Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 by key-exchange method and signing certificate

Cipher suite		**RSA key exchange**	**Fixed Diffie-Hellman key exchange**		**Ephemeral Diffie-Hellman key exchange**		**Fixed EC Diffie-Hellman key exchange**		**Ephemeral EC Diffie-Hellman key exchange**	
4 Char	**2 Char**		**Signed by RSA**[1]	**Signed by DSA**[1]	**Signed by RSA**[1]	**Signed by DSA**[1]	**Signed by RSA**[1]	**Signed by ECDSA**[1]	**Signed by RSA**[1]	**Signed by ECDSA**[1]
0000	00	X								
0001	01	X								
0002	02	X								
0003	03	X								
0004	04	X								
0005	05	X								
0006	06	X								
0009	09	X								
000A	0A	X								
000C	0C			X						
000D	0D			X						
000F	0F		X							
0010	10		X							
0012	12					X				
0013	13					X				
0015	15				X					
0016	16				X					
002F	2F	X								
0030	30			X						
0031	31		X							
0032	32					X				
0033	33				X					
0035	35	X								
0036	36			X						
0037	37		X							
0038	38					X				
0039	39				X					
003B	3B	X								
003C	3C	X								
003D	3D	X								
003E	3E			X						
003F	3F		X							
0040	40					X				
0067	67				X					
0068	68			X						
0069	69		X							
006A	6A					X				
006B	6B				X					
009C	9C	X								
009D	9D	X								
009E	9E				X					

Table 28. Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 by key-exchange method and signing certificate (continued)

Cipher suite		RSA key exchange	Fixed Diffie-Hellman key exchange		Ephemeral Diffie-Hellman key exchange		Fixed EC Diffie-Hellman key exchange		Ephemeral EC Diffie-Hellman key exchange	
4 Char	**2 Char**		**Signed by RSA[1]**	**Signed by DSA[1]**	**Signed by RSA[1]**	**Signed by DSA[1]**	**Signed by RSA[1]**	**Signed by ECDSA[1]**	**Signed by RSA[1]**	**Signed by ECDSA[1]**
009F	9F				X					
00A0	A0		X							
00A1	A1		X							
00A2	A2					X				
00A3	A3					X				
00A4	A4			X						
00A5	A5			X						
C001								X		
C002								X		
C003								X		
C004								X		
C005								X		
C006										X
C007										X
C008										X
C009										X
C00A										X
C00B							X			
C00C							X			
C00D							X			
C00E							X			
C00F							X			
C010									X	
C011									X	
C012									X	
C013									X	
C014									X	
C023										X
C024										X
C025								X		
C026								X		
C027									X	
C028									X	
C029							X			
C02A							X			
C02B										X
C02C										X
C02D								X		
C02E								X		
C02F									X	
C030									X	

Table 28. Cipher suite definitions for SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 by key-exchange method and signing certificate (continued)

Cipher suite		RSA key exchange	Fixed Diffie-Hellman key exchange		Ephemeral Diffie-Hellman key exchange		Fixed EC Diffie-Hellman key exchange		Ephemeral EC Diffie-Hellman key exchange	
4 Char	2 Char		Signed by RSA[1]	Signed by DSA[1]	Signed by RSA[1]	Signed by DSA[1]	Signed by RSA[1]	Signed by ECDSA[1]	Signed by RSA[1]	Signed by ECDSA[1]
C031							X			
C032							X			

[1] SSL V3, TLS V1.0, and TLS V1.1 imposed restrictions on the signing algorithm that must be used to sign a server certificate when using any cipher suites that use a Diffie-Hellman based key-exchange. The TLS V1.2 protocol does not impose such restriction. If the server certificate signing algorithm is listed in the signature algorithm pairs that are specified by the client, the certificate can be used.

Table 29. Supported elliptic curve (group) definitions for TLS V1.0, TLS V1.1, TLS V1.2, and TLS V1.3 and supported key share definitions for TLS V1.3

I.A.N.A Elliptic curve enumerator (decimal)	Named curve by standards organizations						
	SECG	ANSI X9.62	NIST	TLS V1.0	TLS V1.1	TLS V1.2	TLS V1.3
0019	secp192r1	prime192v1	NIST P-192	X	X	X	
0021	secp224r1		NIST P-224	X	X	X	
0023	secp256r1	prime256v1	NIST P-256	X	X	X	X
0024	secp384r1		NIST P-384	X	X	X	X
0025	secp521r1		NIST P-521	X	X	X	X
0029	x25519			X	X	X	X
0030	x448			X	X	X	X

Notes:

- Elliptic curves or supported groups that are not supported for an enabled protocol are ignored. For example, the 0019 (secp192r1) and 0021 (secp224r1) elliptic curves or supported groups are ignored if the application is only enabled for TLS V1.3.
- TLS V1.0, TLS V1.1, and TLS V1.2 elliptic curves X25519 and X448 are only used for the key exchange portion of the handshake processing and not for certificates.

Table 30. Signature algorithm pair and certificate signature algorithm pair definitions for TLS V1.2 and TLS V1.3

Signature algorithm enumerator	Hash and signature algorithm	TLS V1.2	TLS V1.3
0101*	MD5 with RSA	X	
0201	SHA-1 with RSA	X	
0202	SHA-1 with DSA	X	
0203	SHA-1 with ECDSA	X	
0301	SHA-224 with RSA	X	
0302	SHA-224 with DSA	X	
0303	SHA-224 with ECDSA	X	
0401	SHA-256 with RSA	X	X
0402	SHA-256 with DSA	X	
0403	SHA-256 with ECDSA	X	X
0501	SHA-384 with RSA	X	X
0503	SHA-384 with ECDSA	X	X

Table 30. Signature algorithm pair and certificate signature algorithm pair definitions for TLS V1.2 and TLS V1.3 (continued)

Signature algorithm enumerator	Hash and signature algorithm	TLS V1.2	TLS V1.3
0601	SHA-512 with RSA	X	X
0603	SHA-512 with ECDSA	X	X
0804**	SHA-256 with RSASSA-PSS	X	X
0805**	SHA-384 with RSASSA-PSS	X	X
0806**	SHA-512 with RSASSA-PSS	X	X

* - For TLS V1.2 signature algorithm pairs, this algorithm is not allowed to be used while in FIPS mode.

** - For TLS V1.2, this algorithm is ignored for signing if the local certificate is an RSA certificate with a key size of less than 2048.

Table 31. Signature algorithm pair definitions for OCSP request signing and OCSP response signing

Signature algorithm enumerator	Hash and signature algorithm
0101*	MD5 with RSA
0201	SHA-1 with RSA
0202	SHA-1 with DSA
0203	SHA-1 with ECDSA
0301	SHA-224 with RSA
0302	SHA-224 with DSA
0303	SHA-224 with ECDSA
0401	SHA-256 with RSA
0402	SHA-256 with DSA
0403	SHA-256 with ECDSA
0501	SHA-384 with RSA
0503	SHA-384 with ECDSA
0601	SHA-512 with RSA
0603	SHA-512 with ECDSA
0804	SHA-256 with RSASSA-PSS
0805	SHA-384 with RSASSA-PSS
0806	SHA-512 with RSASSA-PSS

* - For OCSP request signing and OCSP response signature algorithm pairs, this algorithm is not allowed to be set or specified while in FIPS mode.

Appendix D. Object identifiers

The following table shows the object identifiers (OIDS) supported by System SSL.

Table 32. System SSL supported object identifiers (OIDS)

Type	Description	OID
Asymmetric Encryption Algorithms	RSA Encryption DSA Diffie-Hellman (dhPublicNumber) ECC (ecPublicKey)	1.2.840.113549.1.1.1 1.2.840.10040.4.1 1.2.840.10046.2.1 1.2.840.10045.2.1
Deprecated Password Based Encryption Algorithms	pb1WithSha1And128BitRc4 pb1WithSha1And40BitRc4 pb1WithSha1And3DesCbc pb1WithSha1And128BitRc2Cbc pb1WithSha1And40BitRc2Cbc	1.2.840.113549.1.12.5.1.1 1.2.840.113549.1.12.5.1.2 1.2.840.113549.1.12.5.1.3 1.2.840.113549.1.12.5.1.4 1.2.840.113549.1.12.5.1.5
Digest Algorithms	MD2 MD5 SHA-1 SHA-224 SHA-256 SHA-394 SHA-512	1.2.840.113549.2.2 1.2.840.113549.2.5 1.3.14.3.2.26 2.16.840.1.101.3.4.2.4 2.16.840.1.101.3.4.2.1 2.16.840.1.101.3.4.2.2 2.16.840.1.101.3.4.2.3
ECC Name Curves	secp192r1 secp224r1 secp256r1 secp384r1 secp521r1 brainpoolP160r1 brainpoolP192r1 brainpoolP224r1 brainpoolP256r1 brainpoolP320r1 brainpoolP384r1 brainpoolP512r1	1.2.840.10045.3.1.1 1.3.132.0.33 1.2.840.10045.3.1.7 1.3.132.0.34 1.3.132.0.35 1.3.36.3.3.2.8.1.1.1 1.3.36.3.3.2.8.1.1.3 1.3.36.3.3.2.8.1.1.5 1.3.36.3.3.2.8.1.1.7 1.3.36.3.3.2.8.1.1.9 1.3.36.3.3.2.8.1.1.11 1.3.36.3.3.2.8.1.1.13
Mask Generation Function Algorithm 1	rsaMgf1	1.2.840.113549.1.1.8
Password Base Encryption Algorithms	pbeWithMd2AndDesCbc pbeWithMd5AndDesCbc pbeWithSha1AndDesCbc pbeWithMd2AndRc2Cbc pbeWithMd5AndRc2Cbc pbeWithSha1AndRc2Cbc pbeWithSha1And40BitRc2Cbc pbeWithSha1And128BitRc2Cbc pbeWithSha1And40BitRc4 pbeWithSha1And128BitRc4 pbeWithSha1And3DesCbc	1.2.840.113549.1.5.1 1.2.840.113549.1.5.3 1.2.840.113549.1.5.10 1.2.840.113549.1.5.4 1.2.840.113549.1.5.6 1.2.840.113549.1.5.11 1.2.840.113549.1.12.1.6 1.2.840.113549.1.12.1.5 1.2.840.113549.1.12.1.2 1.2.840.113549.1.12.1.1 1.2.840.113549.1.12.1.3

Table 32. System SSL supported object identifiers (OIDS) (continued)

Type	Description	OID
Signature Algorithms	md2WithRsaEncryption md5WithRsaEncryption sha1WithRsaEncryption sha224WithRsaEncryption sha256WithRsaEncryption sha384WithRsaEncryption sha512WithRsaEncryption dsaWithSha1 dsaWithSha224 dsaWithSha256 ecdsaWithSha1 ecdsaWithSha224 ecdsaWithSha256 ecdsaWithSha384 ecdsaWithSha512 rsassa-pss	1.2.840.113549.1.1.2 1.2.840.113549.1.1.4 1.2.840.113549.1.1.5 1.2.840.113549.1.1.14 1.2.840.113549.1.1.11 1.2.840.113549.1.1.12 1.2.840.113549.1.1.13 1.2.840.10040.4.3 2.16.840.1.101.3.4.3.1 2.16.840.1.101.3.4.3.2 1.2.840.10045.4.1 1.2.840.10045.4.3.1 1.2.840.10045.4.3.2 1.2.840.10045.4.3.3 1.2.840.10045.4.3.4 1.2.840.113549.1.1.10
Symmetric Encryption Algorithms	DES CBC 3DES CBC RC2 ArcFour AES CBC 128 AES CBC 256	1.3.14.3.2.7 1.2.840.113549.3.7 1.2.840.113549.3.2 1.2.840.113549.3.4 2.16.840.1.101.3.4.1.2 2.16.840.1.101.3.4.1.42
x.500 Distinguished Name Attributes	name surname given name initials generation qualifier common name locality name state or province name organization name organizational unit name title dnQualifier country name email address domain component street address postal code mail serial number	2.5.4.41 2.5.4.4 2.5.4.42 2.5.4.43 2.5.4.44 2.5.4.3 2.5.4.7 2.5.4.8 2.5.4.10 2.5.4.11 2.5.4.12 2.5.4.46 2.5.4.6 1.2.840.113549.1.9.1 0.9.2342.19200300.100.1.25 2.5.4.9 2.5.4.17 0.9.2342.19200300.100.1.3 2.5.4.5

Appendix E. Migrating from deprecated SSL interfaces

In Version 1 Release 2 of z/OS, a new set of functions were added that superseded some functions from previous System SSL releases. The functions that were superseded are referred to collectively as the deprecated SSL interface. It is suggested that new application programs do not use the deprecated SSL interface. For application programs that currently use the deprecated SSL interface, this topic describes how to migrate to the most recent interface.

Note: When migrating from the deprecated SSL interface, the entire System SSL application must be migrated. The application must not contain a mixture of deprecated and superseding APIs.

- Replace manually initializing the *gsk_init_data* structure with **gsk_environment_open()**, plus a number of **gsk_attribute_set_buffer()**, **gsk_attribute_set_enum()** and **gsk_attribute_set_numeric_value()** functions (as needed) to set attributes.
- Replace **gsk_get_cipher_info()** with a call to **gsk_attribute_get_buffer()** to get the list of available ciphers. This call must be done after a successful **gsk_environment_open()** call.
- Replace **gsk_initialize()** with **gsk_environment_init()**.
- Replace manually initializing the *gsk_soc_init_data* structure with **gsk_secure_socket_open()**, plus a number of **gsk_attribute_set_buffer()**, **gsk_attribute_set_enum()** and **gsk_attribute_set_numeric_value()** functions (as needed) to set attributes.
- Replace manually initializing the *gsk_soc_init_data* structure with the addresses of your I/O callback routines with **gsk_attribute_set_callback()**. You specify the address of a *gsk_iocallback* structure that contains the addresses of the callback routines. The *gsk_iocallback* structure is defined in **gskssl.h**. Note that an additional parameter must be added to the function declarator for your existing callback routines.
- Replace **gsk_user_set()** with **gsk_attribute_set_callback()** for defining the address of your get peer ID callback routine. You specify the address of an *gsk_iocallback* structure that contains the address of the callback routine. The *gsk_iocallback* structure is defined in **gskssl.h**. Note that an additional parameter must be added to the function declarator for your existing callback routine.
- Replace **gsk_user_set()** with **gsk_attribute_set_callback()** for defining the address of your session ID cache callback routines. You specify the address of a *gsk_sidcache_callback* structure that contains the address of the callback routines. The *gsk_sidcache_callback* structure is defined in **gskssl.h**.
- Replace **gsk_get_dn_by_label()** with **gsk_get_cert_by_label()**.
- Replace **gsk_secure_soc_init()** with **gsk_secure_socket_init()**.
- Replace **gsk_secure_soc_read()** with **gsk_secure_socket_read()**. Note that **gsk_secure_socket_read()** has an extra parameter to return the length of the data read.
- Replace **gsk_secure_soc_write()** with **gsk_secure_socket_write()**. Note that **gsk_secure_socket_write()** has an extra parameter to return the length of the data written.
- To notify your partner application that you are done sending data on the secure connection, a call to **gsk_secure_socket_shutdown** should be issued before the **gsk_secure_socket_close** call.
- Replace **gsk_secure_soc_close()** with **gsk_secure_socket_close()**.
- Be sure that every **gsk_secure_socket_open()** is matched with a **gsk_secure_socket_close()** even if there is an error on **gsk_secure_socket_init()**. Normal sequence is **open**, **init**, **close**. So, if **init** gets an error return code, you still must do the **close**.
- Be sure that every **gsk_environment_open()** is matched with a **gsk_environment_close()** even if there is an error on **gsk_environment_init()**. Normal sequence is **open**, **init**, **close**. So, if **init** gets an error return code, you still must do the **close**.
- A method is provided to display certificates after **gsk_secure_socket_init()** is issued. You may use **gsk_attribute_get_cert_info()**, if you prefer.

- Note that all of the error return values are renamed and renumbered. Program logic must be changed accordingly.
- There is a **gsk_strerror()** debug routine that returns a text string (in English only) when an error number is passed to it.

Appendix F. Accessibility

Accessible publications for this product are offered through IBM Documentation (www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the Contact the z/OS team web page (www.ibm.com/systems/campaignmail/z/zos/contact_z) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Notices

Terms and conditions for product documentation

Applicability

Personal use

Commercial use

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user’s name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at ibm.com®/privacy and IBM’s Online Privacy Statement at ibm.com/privacy/details in the section entitled “Cookies, Web Beacons and Other Technologies,” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Trademarks

Index

Product Number: 5650-ZOS

SC14-7495-50