IBM i
7.5

Security
Transport Layer Security

Note

Before using this information and the product it supports, read the information in “Notices” on page 37.

This edition applies to IBM i 7.5 (product number 5770-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

Contents

Transport Layer Security

Transport Layer Security (TLS) describes how to use TLS on your system.

Secure Sockets Layer (SSL) and TLS are generic terms for a set of industry standards that are used for enabling applications for secure communication sessions over an unprotected network, such as the Internet. SSL evolved into and was replaced by TLS. TLS is the more accurate term; therefore, TLS is used throughout this topic.

The version of the TLS protocol specification identifies the relative level of security provided. No version of the SSL protocol specification should be used today. Not all versions of the TLS protocol are allowed for security compliance in some industries.

PDF file for TLS

You can view and print a PDF file of this information.

To view or download the PDF version of this document, select Transport Layer Security (TLS).

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF link in your browser.
2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

Downloading Adobe Reader

You need Adobe Reader installed on your system to view or print these PDFs. You can download a free copy from the Adobe Web site.

TLS implementations

The system contains multiple TLS implementations. Each implementation implements one or more versions of the TLS protocols according to the industry definitions.

The implementations must interoperate with other implementations according to the Internet Engineering Task Force (IETF) specifications for each protocol version. Each implementation has unique characteristics and provides different sets of optional functionality.

The set of APIs used determines which implementation is used for each secure application on the system. With Java™, the configured JSSE provider determines the implementation since the Java interfaces are standardized. An application can also embed an implementation that is only known to the application.

These implementations are available to develop applications with on the IBM® i.

- System TLS

  ILE applications use System TLS. Certificate management is performed with the Digital Certificate Manager (DCM) and the certificate store type is Certificate Management Services (CMS) with a file extension of *.KDB. Java applications can use System TLS, however it is not typical. The most obscure case, would be a Java application that uses System TLS while also using a Java Keystore.
- IBMJSSE2 (IBMJSSEProvider2)

  This Java Secure Socket Extension (JSSE) provider contains a pure Java implementation of the TLS protocols and is available on multiple platforms. This implementation is known as the

com.ibm.jsse2.IBMJSSEProvider2 in the java.security provider list. This is the default provider for Java 8. The certificates are typically found in a Java keystore file (JKS) and are managed by using the Java keytool command or IBM Key Management (iKeyman) utility.

For general JSSE information on the system, see Java Secure Socket Extension (JSSE).

For specific details, see the IBMJSSE2 platform independent documentation for the appropriate JDK version. For JDK8, see Security Reference for IBM SDK, Java Technology Edition, Version 8.

- Oracle Java

  This JSSE provider contains Oracle's pure Java implementation of the TLS protocols. This implementation is known as SunJSSE in the java.security provider list. This is the default provider for Java 11.

  For general JSSE information on the system, see Java Secure Socket Extension (JSSE).

  For specific details about this JDK11 provider, see The SunJSSE Provider in the JDK Providers Documentation section of the Oracle Security Developer's Guide.

- OpenSSL

  OpenSSL is an Open Source toolkit that implements TLS protocols and a full-strength general-purpose cryptography library. It is only available in the IBM Portable Application Solutions Environment for i (PASE for i). The certificates are typically found in PEM files and are managed with OpenSSL commands.

  Common Information Model Object Manager is an application that uses this implementation. For more information, see Common Information Model.

Related concepts

Security bulletins

A security bulletin is created when an implementation provides a mitigation to a disclosed vulnerability. The vulnerability can be specific to an implementation or generic to the protocol.

Security bulletins

A security bulletin is created when an implementation provides a mitigation to a disclosed vulnerability. The vulnerability can be specific to an implementation or generic to the protocol.

It is important for administrators to keep current with all mitigations provided by the implementations. Due to implementation differences, not all implementations are vulnerable to a specific vulnerability. If vulnerable, the mitigation might not be the same for each of the implementations.

The security bulletin contains the actions that must be taken to mitigate the issue. When a protocol or algorithm cannot be fixed, the mitigation might disable that feature.

Note: The disabled feature might result in interoperability issues for environments that depend on the vulnerable protocol.

Subscribe to receive notification when TLS security bulletins are created or updated. For more information about this process, see Stay up to date with My Notifications in the IBM Support Portal.

Manage My Notifications to subscribe to "Security bulletin" to receive a notification each time the Security Bulletin publication is updated.

The Security Bulletin publication is an aggregation of individual security bulletins that are created for specific vulnerabilities. Read each individual security bulletin to keep up to date. Security bulletins can be updated after their initial publication when new information becomes available. Updated bulletins are moved back to the top of the aggregation bulletin. The updated bulletin does not have change flags but includes a brief note that indicates what was added to or changed in the bulletin. Re-read the full bulletin contents to see all the updates.

System TLS

System TLS is a set of generic services that are provided in the IBM i Licensed Internal Code (LIC) to protect TCP/IP communications by using the TLS protocol. System TLS is tightly coupled with the operating system and the LIC sockets code specifically providing extra performance and security.

How to code to use System TLS

System TLS is accessible to application developers from the following programming interfaces and JSSE implementation.

- Global Security Kit (GSKit) APIs
  - These ILE C APIs are accessible from other ILE languages.
  - For more information about the Global Security Kit (GSKit) APIs, see the Socket programming topic.
- Integrated IBM i SSL_ APIs
  - These ILE C APIs are accessible from other ILE languages.
  - Do not code to this deprecated interface. This API set does not provide the newer security features available in GSKit, which is the recommended C interface.
- Integrated IBM i JSSE implementation
  - The IBM i JSSE implementation is available for JDK 8 and JDK 11. This implementation is known as com.ibm.i5os.jsse.JSSEProvider in the java.security provider list.

TLS applications that are created by IBM, IBM Business Partners, independent software vendors (ISV), or customers that use one of the three System TLS interfaces previously listed use System TLS. For example, FTP and Telnet are IBM applications that use System TLS. Not all TLS enabled applications that run on IBM i use System TLS.

System TLS system level settings

System TLS has many attributes that determine how secure sessions are negotiated.

Each attribute value is set in one of three ways:

1. The application developer sets an explicit value for the attribute by using code.
2. The application developer provides a user interface to allow the application administrator to indirectly set the attribute value.
3. The application developer does not set a value for the attribute. System TLS uses the default value for the attribute.

Security compliance requirements change over the lifespan of a release. To remain compliant, system administrators need to override some attribute values. System TLS provides various system level settings to implement this level of control.

There are two types of system level control:

- Completely disable the value for an attribute
  - The disabled value is ignored when it is used by any of the three methods of setting the attribute value
  - Application encounters a hard failure if no valid value remains enabled for the attribute
  - Application encounters a soft failure if peer requires the disabled value
- Disable a default value for an attribute
  - Changes only applications that use System TLS defaults for setting this specific attribute
  - Application soft failure if peer requires the disabled value

The system level settings are controlled by using a combination of these interfaces:

- TLS System Values
- System Service Tools (SST) Advanced Analysis command **TLSCONFIG** as specified.

The following System TLS attributes can have their enabled values, default values, or both changed at the system level.

Related concepts

TLSCONFIG command

TLSCONFIG is a System Service Tools (SST) Advanced Analysis command that allows viewing or altering of system-wide System TLS default properties.

Related information

TLS system value: QSSLPCL

TLS system value: QSSLCSLCTL

TLS system value: QSSLCSL

Protocol configuration

System TLS has the infrastructure to support multiple protocols.

The following protocols can be supported by System TLS:

- Transport Layer Security version 1.3 (TLSv1.3)
- Transport Layer Security version 1.2 (TLSv1.2)
- Transport Layer Security version 1.1 (TLSv1.1)
- Transport Layer Security version 1.0 (TLSv1.0)
- Secure Sockets Layer version 3.0 (SSLv3)

CAUTION:

IBM strongly recommends that you always run your IBM i server with the following network protocols disabled. Using configuration options that are provided by IBM to enable the weak protocols results in your IBM i server being configured to allow use of the weak protocols. This configuration results in your IBM i server potentially being at risk of a network security breach. IBM DISCLAIMS AND YOU ASSUME ALL RESPONSIBILITY AND LIABILITY FOR ANY DAMAGE OR LOSS, INCLUDING LOSS OF DATA, ARISING OUT OF OR RELATED TO YOUR USE OF THE SPECIFIED NETWORK PROTOCOL.

Weak protocols (as of November 2021):

- Transport Layer Security version 1.1 (TLSv1.1)
- Transport Layer Security version 1.0 (TLSv1.0)
- Secure Sockets Layer version 3.0 (SSLv3)
- Secure Sockets Layer version 2.0 (SSLv2)

Enabled protocols

The QSSLPCL system value setting identifies the specific protocols that are enabled on the system. Applications can negotiate secure sessions with only protocols that are listed in QSSLPCL. For example, to restrict the System TLS implementation to use only TLSv1.3 and not allow any older protocol versions, set QSSLPCL to contain only *TLSV1.3.

The QSSLPCL special value *OPSYS allows the operating system to change the protocols that are enabled on the system. The value of QSSLPCL remains the same when the system upgrades to a newer operating system release. If the value of QSSLPCL is not *OPSYS, then the administrator must manually add newer protocol versions to QSSLPCL after the system moves to a new release.

Table defining system value QSSLPCL option *OPSYS value for a particular IBM i release.

IBM i release	QSSLPCL *OPSYS definition
i 7.1	*TLSV1, *SSLV3
i 7.2	*TLSV1.2, *TLSV1.1, *TLSV1
i 7.3	*TLSV1.2, *TLSV1.1, *TLSV1
i 7.4	*TLSV1.3, *TLSV1.2
i 7.5	*TLSV1.3, *TLSV1.2

Default protocols

When an application does not specify the protocols to enable, the System TLS default protocols are used. Applications use this design to pick up new TLS support without requiring application code changes. The default protocol setting has no meaning for applications that explicitly specify the protocols to enable for the application.

The default protocols on a system are the intersection of the enabled protocols from QSSLPCL and the eligible default protocols. The eligible default protocol list is configured by using the System Service Tools (SST) Advanced Analysis command **TLSCONFIG**.

To determine the current value of the eligible default protocol list and the default protocol list on the system, use **TLSCONFIG** option `display`. The Retrieve TLS Attributes (QsoRtvTLSA) API retrieves TLS attributes allowing the eligible default protocol list to be retrieved from a program.

An administrator should consider changing the default protocol settings only when no other configuration setting allows an application to interoperate with peers successfully. It is preferred to enable an older protocol for only the specific application that requires it. When the application has an “application definition,” then this enablement is accomplished through the Digital Certificate Manager (DCM).

Warning: Adding an older protocol version to the default list results in opening up all applications that use the default list to known security vulnerabilities. Loading a Group Security PTF might result in the removal of a protocol from the default protocol list. Subscribe to the Security Bulletin to receive notification when a security mitigation includes this type of change. If an administrator adds back an eligible protocol that was removed by a Security PTF, the system remembers this change and does not remove it a second time when the next Security PTF is applied.

If the default protocols must be changed on the system, use **TLSCONFIG** option `eligibleDefaultProtocols` to change the value. **TLSCONFIG** option h displays the help panel that describes how to set the protocol list. Only protocol versions that are listed in the help text can be added to the list.

Note: The **TLSCONFIG** `eligibleDefaultProtocols` setting is reset by installing the Licensed Internal Code (LIC).

Example of setting TLSv1.3 to be the only default protocol on the system:

```
TLSCONFIG -eligibleDefaultProtocols:20
```

Table defining eligible default protocol list with latest Security Group PTF for a particular IBM i release.

IBM i release	Eligible default protocol list with latest Security Group PTF
i 7.1	TLSv1.2, TLSv1.1, TLSv1.0
i 7.2	TLSv1.2, TLSv1.1, TLSv1.0
i 7.3	TLSv1.2, TLSv1.1, TLSv1.0
i 7.4	TLSv1.3, TLSv1.2
i 7.5	TLSv1.3, TLSv1.2

Related concepts

TLSCONFIG command

TLSCONFIG is a System Service Tools (SST) Advanced Analysis command that allows viewing or altering of system-wide System TLS default properties.

Security bulletins

A security bulletin is created when an implementation provides a mitigation to a disclosed vulnerability. The vulnerability can be specific to an implementation or generic to the protocol.

Related information

TLS system value: QSSLPCL

Retrieve TLS Attributes (QsoRtvTLSA) API

Cipher suite configuration

System TLS has infrastructure to support multiple cipher suites.

The cipher suites are specified in different ways for each programming interface. The following table shows the cipher suite specifications, which are shown here in the system value format, that can be supported by System TLS for each protocol version. The supported cipher suite specifications for each protocol are indicated by the "X" in the appropriate column.

Table 1. Supported cipher suite specifications for TLS protocols

QSSLCSL System Value Representation	TLSv1.3	TLSv1.2	TLSv1.1	TLSv1.0	SSLv3
*AES_128_GCM_SHA256	X				
*AES_256_GCM_SHA384	X				
*CHACHA20_POLY1305_SHA256	X				
*ECDHE_ECDSA_AES_128_GCM_SHA256		X			
*ECDHE_ECDSA_AES_256_GCM_SHA384		X			
*ECDHE_RSA_AES_128_GCM_SHA256		X			
*ECDHE_RSA_AES_256_GCM_SHA384		X			
*ECDHE_ECDSA_CHACHA20_POLY1305_SHA256		X			
*ECDHE_RSA_CHACHA20_POLY1305_SHA256		X			
*RSA_AES_128_GCM_SHA256		X			
*RSA_AES_256_GCM_SHA384		X			
*ECDHE_ECDSA_AES_128_CBC_SHA256		X			
*ECDHE_ECDSA_AES_256_CBC_SHA384		X			
*ECDHE_RSA_AES_128_CBC_SHA256		X			

Table 1. Supported cipher suite specifications for TLS protocols (continued)					
QSSLCSL System Value Representation	**TLSv1.3**	**TLSv1.2**	**TLSv1.1**	**TLSv1.0**	**SSLv3**
*ECDHE_RSA_AES_256_CBC_SHA384		X			
*RSA_AES_128_CBC_SHA256		X			
*RSA_AES_128_CBC_SHA		X	X	X	
*RSA_AES_256_CBC_SHA256		X			
*RSA_AES_256_CBC_SHA		X	X	X	
*ECDHE_ECDSA_3DES_EDE_CBC_SHA		X			
*ECDHE_RSA_3DES_EDE_CBC_SHA		X			
*RSA_3DES_EDE_CBC_SHA		X	X	X	X
*ECDHE_ECDSA_RC4_128_SHA		X			
*ECDHE_RSA_RC4_128_SHA		X			
*RSA_RC4_128_SHA		X	X	X	X
*RSA_RC4_128_MD5		X	X	X	X
*RSA_DES_CBC_SHA			X	X	X
*RSA_EXPORT_RC4_40_MD5				X	X
*RSA_EXPORT_RC2_CBC_40_MD5				X	X
*ECDHE_ECDSA_NULL_SHA		X			
*ECDHE_RSA_NULL_SHA		X			
*RSA_NULL_SHA256		X			
*RSA_NULL_SHA		X	X	X	X
*RSA_NULL_MD5		X	X	X	X

Enabled cipher suites

The QSSLCSL system value setting identifies the specific cipher suites that are enabled on the system. Applications can negotiate secure sessions with only a cipher suite that is listed in QSSLCSL. No matter what an application does with code or configuration, it cannot negotiate secure sessions with a cipher suite if it is not listed in QSSLCSL. Individual application configuration determines which of the enabled cipher suites are used for that application.

To restrict the System TLS implementation from using a particular cipher suite, follow these steps:

1. Change QSSLCSLCTL system value to special value *USRDFN to allow the QSSLCSL system value to be edited.
2. Remove all cipher suites to be restricted from the list in QSSLCSL.

The QSSLCSLCTL system value special value *OPSYS allows the operating system to change the cipher suites that are enabled on the system. The value of QSSLCSLCTL remains the same when the system upgrades to a newer operating system release. If the value of QSSLCSLCTL is *USRDFN, then the administrator must manually add in newer cipher suites to QSSLCSL after the system moves to a new release. Setting QSSLCSLCTL back to *OPSYS also adds the new values to QSSLCSL.

A cipher suite cannot be added to QSSLCSL if the TLS protocol that is required by the cipher suite is not set in QSSLPCL.

The cipher suites that are enabled with QSSLCSLCTL *OPSYS in the latest PTF CUM package are displayed in the QSSLCSL system value. They are as follows:

- *AES_128_GCM_SHA256
- *AES_256_GCM_SHA384
- *CHACHA20_POLY1305_SHA256
- *ECDHE_ECDSA_AES_128_GCM_SHA256
- *ECDHE_ECDSA_AES_256_GCM_SHA384
- *ECDHE_RSA_AES_128_GCM_SHA256
- *ECDHE_RSA_AES_256_GCM_SHA384
- *ECDHE_ECDSA_CHACHA20_POLY1305_SHA256
- *ECDHE_RSA_CHACHA20_POLY1305_SHA256

CAUTION:

IBM strongly recommends that you always run your IBM i server with the following cipher suites disabled. Using configuration options that are provided by IBM to enable the weak cipher suites results in your IBM i server being configured to allow use of the weak cipher suite list. This configuration results in your IBM i server potentially being at risk of a network security breach. IBM DISCLAIMS AND YOU ASSUME ALL RESPONSIBILITY AND LIABILITY FOR ANY DAMAGE OR LOSS, INCLUDING LOSS OF DATA, ARISING OUT OF OR RELATED TO YOUR USE OF THE SPECIFIED CIPHER SUITES.

Weak cipher suites (as of November 2021):

- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- TLS_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
- TLS_ECDHE_ECDSA_WITH_NULL_SHA
- TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
- TLS_ECDHE_RSA_WITH_NULL_SHA
- TLS_ECDHE_RSA_WITH_RC4_128_SHA
- TLS_ECDHE_ECDSA_3DES_EDE_CBC_SHA
- TLS_ECDHE_RSA_3DES_EDE_CBC_SHA
- TLS_RSA_3DES_EDE_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA256

Default cipher suites

When an application does not specify the cipher suites to enable, the ordered System TLS default cipher suite list is used. Applications use this design to pick up future new TLS support without requiring application code changes. The default cipher suite setting has no meaning for applications that explicitly specify the cipher suites to enable for the application.

The default cipher suites on a system are the intersection of the enabled cipher suites from QSSLCSL and the eligible default cipher suites. The eligible default cipher suites list is configured by using the System Service Tools (SST) Advanced Analysis command **TLSCONFIG**. The order of the default cipher suite list is the order the cipher suites appear in the QSSLCSL system value. To change the order, change QSSLCSL.

To determine the current value of the eligible default cipher suite list and the default cipher suite list on the system, use **TLSCONFIG** option `display`. The Retrieve TLS Attributes (QsoRtvTLSA) API retrieves TLS attributes allowing the eligible default cipher suite list to be retrieved from a program.

An administrator should only consider changing the default cipher suite list settings when no other configuration setting allows an application to interoperate with peers successfully. It is preferred to enable an older cipher suite for only the specific application that requires it. When the application has an “application definition,” then this enablement is accomplished through the Digital Certificate Manager (DCM).

Warning: Adding an older cipher suite to the default list results in opening up all applications that use the default list to known security vulnerabilities. Loading a Group Security PTF might result in the removal of a cipher suite from the default cipher suite list. Subscribe to the Security Bulletin to receive notification when a security mitigation includes this type of change. If an administrator adds back an eligible cipher suite that was removed by a Security PTF, the system remembers this change and does not remove it a second time when the next Security PTF is applied.

If the default cipher suite list must be changed on the system, use **TLSCONFIG** option `eligibleDefaultCipherSuites` to change the value. **TLSCONFIG** Option h displays the help panel that describes how to specify the changed cipher suite list. The help text includes the short hand values that are required by the option. Only cipher suites that are listed in the help text can be added to the list.

Note: The **TLSCONFIG** `eligibleDefaultCipherSuites` setting is reset by installing the Licensed Internal Code (LIC).

Example of setting only ECDHE cipher suites as the default on the system:

```
TLSCONFIG -eligibleDefaultCipherSuites:YF,YG,YH,YE,YD,YC,YB,YA,Y9,Y8,Y7
```

The cipher suites included in the shipped eligible default cipher suite list with the latest PTF CUM package installed are as follows:

- *AES_128_GCM_SHA256
- *AES_256_GCM_SHA384
- *CHACHA20_POLY1305_SHA256
- *ECDHE_ECDSA_AES_128_GCM_SHA256
- *ECDHE_ECDSA_AES_256_GCM_SHA384
- *ECDHE_RSA_AES_128_GCM_SHA256
- *ECDHE_RSA_AES_256_GCM_SHA384
- *ECDHE_ECDSA_CHACHA20_POLY1305_SHA256
- *ECDHE_RSA_CHACHA20_POLY1305_SHA256

Related concepts
TLSCONFIG command

TLSCONFIG is a System Service Tools (SST) Advanced Analysis command that allows viewing or altering of system-wide System TLS default properties.

Related information

TLS system value: QSSLCSLCTL

TLS system value: QSSLCSL

Retrieve TLS Attributes (QsoRtvTLSA) API

Signature algorithms

The TLSv1.3 protocol provides two extensions for indicating which signature algorithms may be used in digital signatures. The "signature_algorithms_cert" extension applies to signatures in certificates and the "signature_algorithms" extension, which originally appeared in TLSv1.2, applies to signatures in handshake messages.

If no “signature_algorithms_cert" extension is present, then the "signature_algorithms" extension also applies to signatures appearing in certificates. The "signature_algorithms_cert" extension was added to allow implementations which support different sets of algorithms for certificates and in TLS itself to clearly signal their capabilities.

Some TLSv1.2 implementations process and use the "signature_algorithms_cert" extension. TLSv1.2 implementations that do not process it continue to use "signature_algorithms" extension for both purposes.

System TLS system level settings and GSKit attributes are tailored for TLSv1.3. During the transition while TLSv1.2 is supported alongside TLSv1.3, System TLS merges both settings into the original "signature_algorithms" extension when appropriate for increased interoperability.

Related concepts

TLSCONFIG command

TLSCONFIG is a System Service Tools (SST) Advanced Analysis command that allows viewing or altering of system-wide System TLS default properties.

Related information

The Transport Layer Security (TLS) Protocol Version 1.3

Certificate selection

The ordered signature algorithm certificate list is sent to the peer when System TLS requests a certificate during the handshake. The peer uses the received list to guide the certificate selection process.

The peer should select a certificate that conforms to the list, however, that is not true for all implementations and configurations. System TLS treats a received certificate with an undesired signature algorithm as a session error unless optional client authentication is configured. When System TLS receives a client certificate request and is unable to select a conforming certificate, its behavior depends on the TLS protocol version. TLSv1.3 does not send a client certificate. TLSv1.2 sends an available nonconforming RSA or ECDSA certificate. The peer determines whether this certificate results in a session error. For more information about System TLS certificate selection logic, see “Multiple certificate selection” on page 23.

System TLS has the infrastructure to support the following certificate signature algorithms:

- ECDSA_SHA512
- ECDSA_SHA384
- ECDSA_SHA256
- ECDSA_SHA224
- ECDSA_SHA1
- RSA_PSS_SHA512
- RSA_PSS_SHA384
- RSA_PSS_SHA256

- RSA_SHA512
- RSA_SHA384
- RSA_SHA256
- RSA_SHA224
- RSA_SHA1
- RSA_MD5

Enabled certificate signature algorithms

The System Service Tools (SST) Advanced Analysis command **TLSCONFIG** identifies the signature algorithms that are enabled for use in certificates on the system. Applications can negotiate secure sessions only with certificates that use the signature algorithms that are listed for **TLSCONFIG** option `supportedSignatureAlgorithmCertificateList`.

To determine the current value of the enabled signature algorithm certificate list on the system, use **TLSCONFIG** option `display` or the Retrieve TLS Attributes (`QsoRtvTLSA`) API. If the enabled signature algorithm certificate list must be changed on the system, use **TLSCONFIG** option `supportedSignatureAlgorithmCertificateList` to change the value. **TLSCONFIG** option h displays the help text that describes how to set the signature algorithm certificate list. Only signature algorithm values that are listed in the help text can be added to the list.

Note: The **TLSCONFIG** `supportedSignatureAlgorithmCertificateList` setting is reset by installing the Licensed Internal Code (LIC).

Example of setting SHA2 signature algorithms as the supported certificate signature algorithms on the system:

```
TLSCONFIG -supportedSignatureAlgorithmCertificateList:36,35,34,16,15,14
```

System TLS with the latest PTF CUM package installed has the following list of supported certificate signature algorithms:

- ECDSA_SHA512
- ECDSA_SHA384
- ECDSA_SHA256
- ECDSA_SHA224
- ECDSA_SHA1
- RSA_PSS_SHA512
- RSA_PSS_SHA384
- RSA_PSS_SHA256
- RSA_SHA512
- RSA_SHA384
- RSA_SHA256
- RSA_SHA224
- RSA_SHA1
- RSA_MD5

Default certificate signature algorithms

When an application does not specify a signature algorithm certificate list, the System TLS default signature algorithm certificate list is used. Applications use this design to pick up new TLS support without requiring application code changes. The default signature algorithm certificate list has no meaning for applications that explicitly specify the signature algorithm certificate list for the application.

The default signature algorithm certificate list on a system is the intersection of the enabled signature algorithm certificate list and the eligible default signature algorithm certificate list. The eligible default signature algorithm certificate list is configured by using **TLSCONFIG** option `defaultSignatureAlgorithmCertificateList`.

To determine the current value of the eligible default signature algorithm certificate list on the system, use **TLSCONFIG** option `display` or the Retrieve TLS Attributes (`QsoRtvTLSA`) API.

Consider changing the default signature algorithm certificate settings only when no other configuration setting allows an application to interoperate with peers successfully. It is preferred to enable an older signature algorithm for only the specific application that requires it. When the application has an "application definition," this enablement is accomplished through the Digital Certificate Manager (DCM).

If the default signature algorithm certificate list must be changed on the system, use **TLSCONFIG** option `defaultSignatureAlgorithmCertificateList` to change the value. **TLSCONFIG** option h displays the help text that describes how to set the signature algorithm certificate list. Only signature algorithm versions that are listed in the help text can be added to the list.

Note: The **TLSCONFIG** `defaultSignatureAlgorithmCertificateList` setting is reset by installing the Licensed Internal Code (LIC).

Example of setting the ECDSA signature algorithms as the default signature algorithms allowed for certificates on the system:

```
TLSCONFIG -defaultSignatureAlgorithmCertificateList:36,35,34
```

The following displays the order of the shipped default signature algorithm certificate list with the latest PTF CUM package installed:

- ECDSA_SHA512
- ECDSA_SHA384
- ECDSA_SHA256
- RSA_PSS_SHA512
- RSA_PSS_SHA384
- RSA_PSS_SHA256
- RSA_SHA512
- RSA_SHA384
- RSA_SHA256

Message signature

The list of algorithm pairs restricts which signature and hash algorithms can be used for handshake message digital signatures. A handshake message signature algorithm might be different from the signature algorithm of the certificate that is used for the session.

For instance, the handshake message might be protected by SHA256 even though a SHA1 certificate is selected for the session.

System TLS has the infrastructure to support the following message signature algorithms:

- ECDSA_SHA512
- ECDSA_SHA384
- ECDSA_SHA256
- ECDSA_SHA224
- ECDSA_SHA1
- RSA_PSS_SHA512
- RSA_PSS_SHA384
- RSA_PSS_SHA256

- RSA_SHA512
- RSA_SHA384
- RSA_SHA256
- RSA_SHA224
- RSA_SHA1
- RSA_MD5

Enabled signature algorithms

The System Service Tools (SST) Advanced Analysis command **TLSCONFIG** identifies the signature algorithms for use in handshake messages that are enabled on the system. Applications can negotiate secure sessions with only the signature algorithms that are listed for **TLSCONFIG** option `supportedSignatureAlgorithmList`.

To determine the current value of the enabled signature algorithm list on the system, use **TLSCONFIG** option `display` or the Retrieve TLS Attributes (`QsoRtvTLSA`) API.

If the enabled signature algorithm list must be changed on the system, use **TLSCONFIG** option `supportedSignatureAlgorithmList` to change the value. **TLSCONFIG** option h displays the help text that describes how to set the signature algorithm list. Only signature algorithm values that are listed in the help text can be added to the list.

Note: The **TLSCONFIG** `supportedSignatureAlgorithmList` setting is reset by installing the Licensed Internal Code (LIC).

Example of setting the ECDSA signature algorithms as the supported signature algorithms on the system:

```
TLSCONFIG -supportedSignatureAlgorithmList:36,35,34
```

System TLS with the latest PTF CUM package installed has the following list of supported signature algorithms:

- ECDSA_SHA512
- ECDSA_SHA384
- ECDSA_SHA256
- RSA_PSS_SHA512
- RSA_PSS_SHA384
- RSA_PSS_SHA256
- RSA_SHA512
- RSA_SHA384
- RSA_SHA256

Default signature algorithms

When an application does not specify a signature algorithm list, the System TLS default signature algorithm list is used. Applications use this design to pick up new TLS support without requiring application code changes. The default signature algorithm list has no meaning for applications that explicitly specify the signature algorithm list for the application.

The default signature algorithm list on a system is the intersection of the enabled signature algorithm list and the eligible default signature algorithm list. The eligible default signature algorithm list is configured by using **TLSCONFIG** option `defaultSignatureAlgorithmList`.

To determine the current value of the eligible default signature algorithm list on the system, use **TLSCONFIG** option `display` or the Retrieve TLS Attributes (`QsoRtvTLSA`) API.

Consider changing the default signature algorithm settings only when no other configuration setting allows an application to interoperate with peers successfully. It is preferred to enable an older signature

algorithm for only the specific application that requires it. When the application has an “application definition,” then this enablement is accomplished through the Digital Certificate Manager (DCM).

If the default signature algorithm list must be changed on the system, use **TLSCONFIG** option `defaultSignatureAlgorithmList` to change the value. **TLSCONFIG** option h displays the help text that describes how to set the signature algorithm list. Only signature algorithm values that are listed in the help text can be added to the list.

Note: The **TLSCONFIG** `defaultSignatureAlgorithmList` setting is reset by installing the Licensed Internal Code (LIC).

Example of setting the ECDSA signature algorithms as the default signature algorithms on the system:

```
TLSCONFIG -defaultSignatureAlgorithmList:36,35,34
```

The following displays the order of the shipped default signature algorithm list with the latest PTF CUM package installed:

- ECDSA_SHA512
- ECDSA_SHA384
- ECDSA_SHA256
- RSA_PSS_SHA512
- RSA_PSS_SHA384
- RSA_PSS_SHA256
- RSA_SHA512
- RSA_SHA384
- RSA_SHA256

Minimum RSA key size

The minimum RSA key size that is allowed for a certificate that is used by either side of a handshake can be restricted for System TLS.

An RSA digital certificate has a public/private key pair. The key size for the pair is measured in bits. Security compliance policies exist that require the RSA key size meet a minimum level.

The RSA key size is set when the certificate is created. The system administrator must create new certificates with a larger RSA key to replace any certificate that uses a key size smaller than the minimum RSA key size value set.

System TLS has a system level setting to restrict the RSA key size that is allowed for a certificate it uses. The restriction applies to local and peer certificates and includes both client and server certificates. Setting a minimum key size results in a handshake failure when either side's certificate contains an RSA key smaller than the minimum size.

Before the administrator changes the system level setting for minimum key size, manually check and replace existing local certificates that have keys smaller than the desired minimum to avoid application failures. Setting the minimum RSA key size restriction prevents inter-operation with peers that use RSA certificates with a key size smaller than the minimum.

The initial minimum RSA key size system level setting has a value of 0. A value of 0 places no restriction on the key size.

The System TLS system level setting is changed by using the System Service Tools (SST) Advanced Analysis command **TLSCONFIG**. To determine the current setting, use **TLSCONFIG** option `display`. Use **TLSCONFIG** option `minimumRsaKeySize` to change the value. **TLSCONFIG** option h displays the help panel that describes how to set the key size.

For example, use the following command to set 2 K (2048 bits) as the minimum RSA key size allowed on the system:

```
TLSCONFIG -minimumRsaKeySize:2048
```

The GSKit attribute GSK_MIN_RSA_KEY_SIZE can be set by the application to a higher minimum key size to allow individual applications to be more restrictive than the rest of the system. If the application sets GSK_MIN_RSA_KEY_SIZE to a value lower than **TLSCONFIG** option `minimumRsaKeySize`, the **TLSCONFIG** `minimumRsaKeySize` value is used, ignoring the application's attempt to reduce the minimum key size.

ECDSA certificates are not impacted by this setting since they do not have an RSA key.

Related concepts

TLSCONFIG command

TLSCONFIG is a System Service Tools (SST) Advanced Analysis command that allows viewing or altering of system-wide System TLS default properties.

Supported groups

The TLSv1.3 and TLSv1.2 protocols share an extension in the handshake messages that each protocol label and interpret differently. The TLSv1.3 protocol refers to it as "supported_groups" and uses it to determine the elliptic curve group that is used for key exchange. The TLSv1.2 protocol refers to it as "elliptic_curves" and uses it to determine the elliptic curve group that is used for key exchange and also uses it to determine supported certificates.

System TLS system level settings and GSKit attributes are tailored for TLSv1.3 though also used for TLSv1.2 when appropriate.

System TLS supports Elliptic Curve Digital Signature Algorithm (ECDSA) based certificates. The key size for an ECDSA certificate is determined by the named curve set when the certificate is created.

System TLS and the Digital Certificate Manager (DCM) have the infrastructure to support the following named curves:

- x25519
- x448
- Secp521r1
- Secp384r1
- Secp256r1

When you view a certificate in DCM, the key size that is associated with the named curve is displayed in bits.

Enabled named elliptic curve groups

The System Service Tools (SST) Advanced Analysis command **TLSCONFIG** identifies the system level setting to restrict the supported named elliptic curve groups.

When used for TLSv1.3 protocol negotiation, it restricts which named elliptic curve groups are allowed for key exchange. It has no impact on certificate selection or support.

When used for TLSv1.2 protocol negotiation, it is used for two different purposes. Like TLSv1.3, it restricts which named elliptic curve groups are allowed for key exchange. The second purpose is to restrict the ECDSA key sizes that are allowed for a certificate. The restriction applies to local and peer certificates and includes both client and server certificates. Restricting the supported list of named elliptic curves results in a handshake failure when the server or client certificate contain an ECDSA key size not in the supported list.

To determine the current value of the enabled named elliptic curve group list, use **TLSCONFIG** option `display` or the Retrieve TLS Attributes (QsoRtvTLSA) API. If the enabled named elliptic curve group list on the system must be changed, use **TLSCONFIG** option `supportedNamedCurve` to change the value. **TLSCONFIG** option h displays the help text that describes how to set the named elliptic curve group values. Only named curve values that are listed in the help text can be added to the list.

Note: The **TLSCONFIG** supportedNamedCurve setting is reset by installing the Licensed Internal Code (LIC).

Example of setting 256 and 384-bit key sizes as the supported named elliptic curve group list on the system:

```
TLSCONFIG -supportedNamedCurve:23,24
```

System TLS with the latest PTF CUM package installed has the following list of supported named elliptic curve groups:

- x25519
- x448
- Secp256r1
- Secp384r1
- Secp521r1

Default named elliptic curve groups

When an application does not specify a named elliptic curve group list, the System TLS default named elliptic curve group list is used. Applications use this design to pick up new TLS support without requiring application code changes. The default named elliptic curve group list has no meaning for applications that explicitly specify the named elliptic curve group list for the application.

The default named elliptic curve group list on a system is the intersection of the enabled named elliptic curve group list and the eligible default named elliptic curve group list. The eligible default named elliptic curve group list is configured by using **TLSCONFIG** option defaultNamedCurve.

To determine the current value of the default named elliptic curve group list on the system, use **TLSCONFIG** option display or the Retrieve TLS Attributes (QsoRtvTLSA) API.

Consider changing the default named elliptic curve groups settings only when no other configuration setting allows an application to interoperate with peers successfully. It is preferred to enable a weaker named elliptic curve group for only the specific application that requires it. When the application has an “application definition,” then this enablement is accomplished through the Digital Certificate Manager (DCM).

If the default named elliptic curve group list must be changed on the system, use **TLSCONFIG** option defaultNamedCurve to change the value. **TLSCONFIG** option h displays the help text that describes how to set the named elliptic curve group list. Only named elliptic curve groups that are listed in the help text can be added to the list.

Note: The **TLSCONFIG** defaultNamedCurve setting is reset by installing the Licensed Internal Code (LIC).

Example of setting 384 and 256-bit key sizes as the default named elliptic curve group list on the system:

```
TLSCONFIG -defaultNamedCurve:23,24
```

The following displays the order of the shipped default named elliptic curve groups with the latest PTF CUM package installed:

- Secp256r1
- Secp384r1
- x25519
- Secp521r1
- x448

Related concepts
TLSCONFIG command

TLSCONFIG is a System Service Tools (SST) Advanced Analysis command that allows viewing or altering of system-wide System TLS default properties.

Middlebox compatibility mode

TLSv1.3 RFC 8446 Appendix D defines an optional Middlebox Compatibility Mode applications can use to mitigate middlebox issues in the network.

The TLSv1.3 protocol significantly altered existing handshake message formats, and added and removed other handshake messages. Early TLSv1.3 adopters encountered handshake failures as a result of middleboxes on the network between the client and server not understanding the new protocol. The problem middleboxes didn't adhere to previous TLS protocol versions' specifications making them brittle to the new protocol.

Middlebox Compatibility Mode makes the TLSv1.3 handshake flow look more like a TLSv1.2 handshake. This is accomplished by filling in legacy fields in handshake messages and by sending a TLSv1.2 handshake message eliminated from the pure TLSv1.3 implementation. These changes make the TLSv1.3 handshake appear similar to a TLSv1.2 session resumption which misbehaving middleboxes understand and generally allow through.

System TLS does not initiate middlebox compatibility mode by default. If needed in your network, this mode can be turned on globally for System TLS using **TLSCONFIG** option `middleboxCompatibilityMode`. Individual GSKit applications can enable this mode with gsk_attribute_set_enum(). System TLS always responds with middlebox compatibility mode processing when a peer is detected using this mode.

Related information

The Transport Layer Security (TLS) Protocol Version 1.3

Renegotiation

The TLSv1.3 protocol eliminated renegotiation. The information in this section is only applicable to TLSv1.2 and earlier protocol versions.

Starting a new handshake negotiation inside of an existing secure session is called renegotiation. There are two properties that determine System TLS renegotiation characteristics.

Multiple reasons exist for an application to use renegotiation. Renegotiation can be started by either the client or server. The application layer might not be aware that a secure session is renegotiated at the request of a peer.

Note: A GSKit System TLS application uses `gsk_secure_soc_misc()` to initiate renegotiation.

The TLS protocol architecture as defined by their base RFCs contain a flaw with renegotiation. The protocols fail to provide cryptographic verification that a session renegotiation is linked to the existing secure session. Supplemental RFC 5746 defines an optional extension to the base protocols to correct the issue.

Since RFC 5746 is an addition to a previously defined protocol, not all TLS implementations currently support it. Some TLS implementations have not or cannot be updated to support RFC 5746. To allow for business continuity/interoperability during various stages of the transition, two renegotiation properties exist to specify the renegotiation mode.

TLS Renegotiation Mode

The System TLS default requires RFC 5746 semantics are used for all renegotiated handshakes. The default mode can be changed with option `renegotiation` on the System Service Tools (SST) Advanced Analysis command **TLSCONFIG**.

The mode can be set to allow all unsecure renegotiations or to allow only abbreviated unsecure renegotiations. Use these modes only after careful consideration. **TLSCONFIG** option h displays the help panel that describes how to set the TLS renegotiation mode

A mode exists to disable all peer initiated handshake renegotiation. This mode prevents secure (RFC 5746 semantics) and unsecure renegotiation. This mode can result in interoperability issues for applications that require the use of renegotiation. Locally initiated secure renegotiation such as `gsk_secure_soc_misc()` is still allowed in this mode.

TLS Extended Renegotiation Critical Mode

Extended Renegotiation Critical Mode determines when System TLS requires all peers provide the RFC 5746 renegotiation indication during initial session negotiation.

To completely protect both sides of the secure session against the renegotiation weakness, all initial negotiations must indicate support for RFC 5746. This indication can be in the form of the "renegotiation_info" TLS Extension or the Signaling Cipher Suite Value (SCSV) as defined by RFC 5746.

Critical mode is disabled by default to maintain interoperability with TLS implementations that do not implement RFC 5746. When critical mode is enabled, System TLS is restricted to negotiating with systems that implement RFC 5746. The restriction exists even if neither side supports or uses renegotiation. If it is determined that all System TLS peers support RFC 5746, then this mode should be changed to be enabled.

The default extended renegotiation critical mode can be changed with the System Service Tools (SST) Advanced Analysis command **TLSCONFIG**. There is a property for client applications, **TLSCONFIG** option `rfc5746NegotiationRequiredClient`, and a separate property, **TLSCONFIG** option `rfc5746NegotiationRequiredServer`, for server applications.

System TLS always sends the "renegotiation_info" TLS Extension or the SCSV in the ClientHello. SCSV is sent only if no other extensions are part of the ClientHello.

Related concepts

TLSCONFIG command

TLSCONFIG is a System Service Tools (SST) Advanced Analysis command that allows viewing or altering of system-wide System TLS default properties.

Related information

RFC 5746: "Transport Layer Security (TLS) Renegotiation Indication Extension"

Key update

The TLSv1.3 protocol uses key update processing to generate new read and write keys for an existing secure session. This replaces renegotiation and its deficiencies with a simple key update mechanism.

Note: A GSKit System TLS application uses `gsk_secure_soc_misc()` to initiate key update for TLSv1.3 the same way it would initiate renegotiation for TLSv1.2 and earlier.

Online Certificate Status Protocol

Online Certificate Status Protocol (OCSP) provides applications a way to determine the revocation status for a digital certificate. Certificate revocation status that is checked via OCSP provides more up-to-date status information than is available through CRLs.

The implementation of OCSP revocation status checking is done in accordance with RFC 2560. OCSP certificate revocation status checking is available for the end entity certificate. Protocol version 1 over HTTP and the basic response type are supported.

Certificate revocation status is checked on behalf of the application via OCSP when at least one of the following conditions are true:

- A URL address of an OCSP responder is configured.
- Authority Information Access (AIA) checking is enabled and the certificate to be validated has an AIA extension. The AIA extension must contain a PKIK_AD_OCSP access method with a URI that indicates the HTTP location of the OCSP responder.

Note: Only the first OCSP responder that is identified in the AIA extension is queried for revocation status.

When URL and AIA checking are enabled, the URL responder is queried first. This order can be changed for an individual application by setting the Global Security Kit (GSKit) API attribute, GSK_OCSP_CHECK_AIA_FIRST. A query to the second responder is only sent if the query sent to the first responder results in an undetermined revocation status.

Client sessions with certificate status request processing enabled can ask the server session to send a stapled OCSP response as part of session negotiation for TLS protocols TLSv1.3 and TLSv1.2. The client session processes the stapled OCSP response from the server eliminating the need for the client to query an OCSP responder for certificate revocation status. A server session must also enable certificate status request processing in order to support OCSP stapling when requested by a client application.

Related concepts

Certificate revocation
Certificate revocation checking is one phase of certificate validation that is done as part of session negotiation. The certificate chain is validated to ensure that the certificate is not revoked.

Related information

RFC 2560: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP

OCSP configuration

In addition to enabling Online Certificate Status Protocol (OCSP), there are a number of properties that can be configured by an application to customize the OCSP behavior.

When OCSP revocation checking is enabled, an HTTP request is sent to an OCSP responder. The request contains information to identify the certificate for which revocation status is being queried and an optional signature. The optional signature on the request is used to help the responder verify valid requests that are received from clients. Request signatures are disabled by default. The request is sent to the responder over HTTP by the GET or the POST method. Requests that are sent by the GET method enable HTTP caching. If configuration indicates that the GET method is preferred and the request is smaller than 255 bytes, the request is sent by the GET method. Otherwise, the request is sent by the POST method. The GET method is preferred by default.

After a request is sent, OCSP revocation checking blocks until a response is received from the responder or it times out. Revocation checking happens as part of the session negotiation; therefore, the session negotiation blocks while revocation checking is done. If the timeout set on the session negotiation is smaller than the OCSP timeout configured, the smaller value is used for the OCSP timeout. The OCSP timeout value defaults to 10 seconds, but can be configured by an application.

A valid response is signed and includes data to indicate the revocation status of the certificate queried. The revocation status of the certificate can be good, unknown, or revoked. The response must be signed by a certificate that meets at least one of the following requirements:

- The signing certificate is trusted by the local certificate store.
- The signing certificate is the certificate authority (CA) that issued the certificate to be validated.
- The signing certificate includes a value of id-ad-ocspSigning in an ExtendedKeyUsage extension and is issued by the CA that issued the certificate in question.

The size of a response can vary. It is up to the application to determine the maximum response size allowed. By default the maximum response size that is allowed is 20480 bytes. If a response is larger than the maximum size allowed, the response is ignored and treated the same as a response that indicates unknown certificate status.

A cryptographic nonce value is a security mechanism that can be used to verify that the response received is a reply to a particular request. The nonce value, which is a random generated bit string, is computed and included as part of both the request and response. If nonce checking is enabled, the nonce value included on the response is verified with the value that is sent in the request. If the nonce values do not match, the response is ignored. Nonce checking is disabled by default.

Revocation checking can slow down session negotiation. However, caching OCSP responses allows the client to obtain revocation status from previous requests without sending the same request again. The OCSP response cache is enabled by default, but can be disabled for an application.

A proxy HTTP server can be used as an intermediate server to handle OCSP requests from cached responses, or forward requests to the configured responder. If a proxy server is configured for an application, all the OCSP requests for the application are sent to the configured server. The default proxy port is 80. A proxy server is not configured by default.

Client sessions have the ability to request that a server session send the OCSP request for the server certificate on behalf of the client. Enabling this option on the client causes the client to send the certificate status request extension to the server. When the certificate status request extension is sent by the client and certificate status request processing is enabled on the server session, the server sends a stapled OCSP response to the client as part of session negotiation. A client session can require certificate status request processing. This means that if the server certificate has the feature extension indicating that it must staple, the server must staple an OCSP response or else the client fails session negotiation. Certificate status request processing is disabled on both client and server sessions by default.

The Global Security Kit (GSKit) APIs, `gsk_attribute_set_buffer()`, `gsk_attribute_set_numeric_value()`, and `gsk_attribute_set_enum()`, can be used to configure OCSP with the following API attributes:

- GSK_OCSP_URL - URL of the OCSP responder to which OCSP requests are sent
- GSK_OCSP_ENABLE - Enable AIA checking
- GSK_OCSP_CHECK_AIA_FIRST - Determine if OCSP URL or OCSP AIA extension checking should be done first
- GSK_OCSP_REQUEST_SIGKEYLABEL - Certificate label of the certificate that is used to sign the OCSP request
- GSK_OCSP_REQUEST_SIGALG - Signature algorithm that is used to generate the signature of the OCSP request
- GSK_OCSP_RETRIEVE_VIA_GET - Method with which the OCSP request is sent
- GSK_HTTP_CONNECT_TIMEOUT - Number of seconds to wait for a connection to the OCSP responder to complete
- GSK_OCSP_TIMEOUT - Number of seconds to wait for a response from the OCSP responder
- GSK_OCSP_MAX_RESPONSE_SIZE - Maximum response size in bytes to be accepted from the OCSP responder
- GSK_OCSP_CLIENT_CACHE_SIZE - Enable or disable the OCSP client response cache
- GSK_OCSP_NONCE_GENERATION_ENABLE - Send a nonce extension as part of the OCSP request
- GSK_OCSP_NONCE_CHECK_ENABLE - Verify that the nonce extension in the OCSP response matches the one sent in the OCSP request
- GSK_OCSP_NONCE_SIZE - Number of bytes to be used to generate the nonce value
- GSK_OCSP_PROXY_SERVER_NAME - Server name of the proxy server to which OCSP requests are sent
- GSK_OCSP_PROXY_SERVER_PORT - Port number of the proxy server to which OCSP requests are sent
- GSK_SSL_EXTN_CERTSTATUSREQ_ENABLE - Enable or disable certificate status request processing on a client or server session
- GSK_TLS_FEATURES_EXTN_ENABLE - Indicate if certificate status request processing is required by a client session when the server's certificate includes the feature extension for must staple

Applications that use the integrated IBM i SSL_ APIs or IBM i JSSE implementation do not have an interface to configure OCSP. However, any programs that use an "application ID" can enable or disable OCSP revocation checking through DCM. The default values are used for all other OCSP configuration options.

Related concepts

Online Certificate Status Protocol attributes

The Online Certificate Status Protocol (OSCP) attribute fields in the application definitions are used to control OCSP enablement.

Related information

Global Security Kit (GSKit) APIs

OCSP revocation status

OCSP revocation status is determined by the OCSP response sent in reply to an OCSP request. Two types of responses can be received. One indicates that the OCSP responder sent a valid response; the other signals that the responder encountered an issue as it processed the prior request.

The issues a responder might encounter while it processes a request include the following:

- malformedRequest - The request was malformed.
- internalError - An internal error occurred on the OCSP responder.
- tryLater - The OCSP responder is temporarily unable to respond; try the request again later.
- sigRequired - The OCSP responder requires that the request must be signed.
- unauthorized - The OCSP client is not authorized to query the OCSP responder.

A valid response contains certificate revocation status for the queried certificate. The possible certificate status values are good, revoked, or unknown. OCSP revocation status checking is complete if it receives a good or revoked revocation status for the certificate. A good status allows the handshake to continue and revoked status causes the handshake to fail.

If the revocation status from both URL and AIA checking is undetermined, the certificate is allowed to be used. Information about the certificate with an undetermined revocation status can be retrieved with `gsk_attribute_get_buffer()` and attribute GSK_UNKNOWNREVOCATIONSTATUS_SUBJECT. The application should close the connection if an undetermined revocation status is not allowed.

Undetermined revocation status

The following responses result in undetermined revocation status:

- No response received within the specified timeout
- OCSP response status that indicates the responder encountered an issue
- Valid response with one of the following conditions:
  - Unknown response type (only PKIX_AD_OCSP_basic response type supported)
  - Unknown response version (only version 1 supported)
  - Invalid signature or invalid signing certificate
    - The signing certificate must meet one of the following criteria:
      - Is trusted by the local keystore
      - Is the certificate of the certificate authority (CA) that issued the certificate in question
      - Includes a value of id-ad-ocspSigning in an ExtendedKeyUsage extension and is issued by the CA that issued the certificate in question
  - No nonce included on the response when nonce checking is required
  - Bad nonce value on the response when nonce checking is required
  - Invalid or expired nextUpdate value is specified on the response
  - Unknown certificate status value that indicates the responder does not know the status of the certificate

Certificate revocation

Certificate revocation checking is one phase of certificate validation that is done as part of session negotiation. The certificate chain is validated to ensure that the certificate is not revoked.

The following steps apply for certificate revocation checking:

1. Check revocation status with a Certificate Revocation List (CRL) location.

   a. When a CRL location is configured through the Digital Certificate Manager (DCM), a CRL database (LDAP) server is queried for CRLs containing the revocation status of the certificate.

      - If the certificate is revoked, the certificate revocation phase of certificate validation is complete and the session negotiation fails.
      - Otherwise, continue with certificate revocation processing.

      Note: CRL locations are configured individually for each certificate authority (CA) in a local certificate store.

2. Check revocation status with Online Certificate Status Protocol (OCSP).

   When both URL and AIA checking are enabled, the order in which the responders are queried is determined by the Global Security Kit (GSKit) API attribute, `GSK_OCSP_CHECK_AIA_FIRST`. The default is to check the URL responder address first.

   a. If a client session enables certificate status request processing, then the client uses a stapled OCSP response received from the server to determine revocation status first. This step only applies to client session negotiation using TLS protocols TLSv1.3 or TLSv1.2.

      - If the client requires certificate status request processing and the server's certificate includes the feature extension defined in RFC 7633 indicating that the server must staple, the client fails the handshake when a stapled OCSP response is not received from the server. This completes the certificate revocation phase of certificate validation and the session negotiation fails.
      - If the certificate is revoked in the stapled OCSP response, the certificate revocation phase of certificate validation is complete and the session negotiation fails.
      - If the certificate is good in the stapled OCSP response, the certificate revocation phase of certificate validation is complete and certificate validation continues.
      - If the certificate revocation status is undetermined from the stapled OCSP response, the certificate revocation phase of certificate validation is complete and certificate validation continues.

   b. When an OCSP URL responder address is configured, query the responder.

      - If the certificate is revoked, the certificate revocation phase of certificate validation is complete and the session negotiation fails.
      - If the certificate is good, the certificate revocation phase of certificate validation is complete and certificate validation continues.
      - If the certificate revocation status is undetermined, continue with certificate revocation processing.

   c. When AIA checking is enabled and the certificate has a PKIK_AD_OCSP access method with a URI that indicates the HTTP location, query the responder.

      - If the certificate is revoked, the certificate revocation phase of certificate validation is complete and the session negotiation fails.
      - If the certificate is good, the certificate revocation phase of certificate validation is complete and certificate validation continues.
      - If the certificate revocation status is undetermined, the certificate revocation phase of certificate validation is complete and certificate validation continues.

   Note: If revocation status is undetermined, GSKit stores information about the certificate for which revocation status is undetermined and continues as if the status is not revoked. The application can retrieve undetermined certificate status information with `gsk_attribute_get_buffer()` and

attribute GSK_UNKNOWNREVOCATIONSTATUS_SUBJECT and make a policy decision on whether to continue or end the connection.

Note: An application definition that is configured in DCM can override CRL and OCSP revocation checking that is configured by an application that uses the application definition.

Related concepts

Online Certificate Status Protocol

Online Certificate Status Protocol (OCSP) provides applications a way to determine the revocation status for a digital certificate. Certificate revocation status that is checked via OCSP provides more up-to-date status information than is available through CRLs.

Related information

Managing CRL locations

Multiple certificate selection

System TLS allows up to four certificates to be assigned to a secure environment. The purpose of multiple certificates is to enable the use of 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 TLS capabilities. When one client does not support ECDSA certificates, the server must retain the ability to use an RSA certificate in order to negotiate a secure connection with that client.

Two methods exist to configure multiple certificates for an environment:

- Use the Digital Certificate Manager (DCM) to assign multiple certificates to the application definition configured for the secure environment.
- Use the GSKit API gsk_attribute_set_buffer(GSK_KEYRING_LABEL_EX) to configure a list of certificate labels to be used.

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

Selection Considerations

When the negotiated protocol is TLSv1.1, TLSv1.0, or SSLv3 the first RSA certificate found that also has an RSA signature is used.

When the negotiated protocol is TLSv1.2, the selection process involves several steps.

1. The key type of the server's most preferred cipher suite that is supported by the client starts the selection process. A certificate has either an ECDSA or RSA key. If the cipher suite name contains "ECDSA", then an ECDSA key/certificate must be used. If the cipher suite name contains "RSA", then an RSA key/certificate must be used. If a matching certificate does not exist, then selection moves to the next cipher suite in the list until it finds a cipher suite that works with one of the certificates.

   When one or more certificate key type matches are found, additional criteria are checked to determine which if any can be used based on the rest of the handshake attributes.
2. When ECDSA is the key type that is being considered, the named curve (which equates to key size) must be supported by the peer. If the ECDSA certificates under consideration have different named curves, the peer's preferred order of named curves is used to determine the preferred certificate. One or more certificates can be tried for selection after this step. If no certificates remain eligible after this step, revert to the previous step.
3. Both RSA and ECDSA certificates have a signature from the certificate's issuing certificate authority. The signature key type can be different from the server certificate key type. When multiple certificates remain eligible for selection in this phase, the one with the signature most preferred by the peer is selected.

An ECDSA certificate must have a signature algorithm that is allowed by the peer's ordered signature algorithm list. If no ECDSA certificates in this step have a supported signature algorithm, revert to the previous step where a less preferred named curve can be considered for additional selection processing.

If no RSA certificates in this step have a supported signature algorithm, revert to the first step where a different key type can be considered for additional selection processing.

If there are multiple certificates still equivalent after this step, the first one processed is selected. The other equivalent certificates are never selected as they are cryptographically identical to the first certificate. Remove the additional certificates from the configuration to eliminate the extra selection processing.

4. When processing completes with no certificate that meets the preceding selection criteria, the first RSA certificate that is processed is selected provided an RSA cipher suite is in the list. If no RSA certificate is selected, then the first ECDSA certificate that is processed is selected provided an ECDSA cipher suite is in the list.

The peer makes the final decision to allow the certificate and thus whether the connection works or not.

When the negotiated protocol is TLSv1.3, the selection process involves fewer steps than TLSv1.2.

1. The negotiated cipher suite is not part of the certificate selection process.
2. The named elliptic curve group list is not part of the certificate selection process.
3. The signature algorithm certificate list is the primary factor for certificate selection. Both RSA and ECDSA certificates have a signature from the certificate's issuing certificate authority. The certificate with the signature most preferred by the peer list is selected. When ECDSA is the key type of the certificate under consideration, the certificate's key size must match the signature algorithm's hash. If a match, the certificate is selected. If key size does not match hash size, the certificate is selected if no configured certificates match this requirement.
4. When processing completes with no certificate that meets the preceding selection criteria, the first certificate encountered with a signature that is not SHA1 or MD5 is selected.

The peer makes the final decision to allow the certificate and thus whether the connection works or not.

DCM application definitions

Digital Certificate Manager (DCM) manages an application database that contains application definitions. Each application definition encapsulates certificate processing information for a specific application. As of the IBM i 7.1 release, the application definition also encapsulates some System TLS attributes for the application. System TLS users know this application definition as an "Application ID."

Many of the IBM i provided applications use application definitions to configure certificate information for their application. Any application developer can design an application to use application definitions.

Application definitions can be managed under the **Manage Application Definitions** menu in DCM.

- Manage application definitions in the *SYSTEM certificate store
  1. Click **Open a Certificate Store** in the left navigation pane.
  2. Click *SYSTEM .
  3. Enter the password that is associated with *SYSTEM certificate store and click **Open**.
  4. Click **Manage Application Definitions**.
- View application definitions in the *SYSTEM certificate store
  1. From the **Application Definitions** page, click **View** in the box for the application that you want to view.
- Update application definitions in the *SYSTEM certificate store
  1. From the **Application Definitions** page, click the **+** to expand the actions for the application that you want to update.

2. Click **Update** to update the application definition.
3. Update the TLS attributes that you want to change and click **Update** to save your changes.

The DCM application definition contains two fields that are used to identify it. The **Application ID** field is used by System TLS to identify the application definition that holds the configuration information. The **Application description** field describes the **Application ID** and is displayed in DCM.

Each of the following System TLS programming interfaces has a method for identifying the “Application ID” to use.

- Global Security Kit (GSKit) APIs
  - gsk_attribute_set_buffer(with attribute GSK_IBMI_APPLICATION_ID)
- Integrated IBM i SSL_ APIs
  - SSL_Init_Application(set value in struct SSLInitAppStr)
- Integrated IBM i JSSE implementation
  - Java system property os400.secureApplication

The following DCM application definition fields can be used to control the corresponding System TLS attributes of an application:

TLS protocols

The **TLS protocols** application definition field determines which TLS protocol versions are supported by the application.

The default value is *PGM, which means the program that uses this "application ID" sets the TLS protocol attribute to the appropriate value. All System TLS programs have a protocol attribute value that is set explicitly through an API call, or implicitly by allowing the system default to be used. Use *PGM unless it is known that the required attribute value is not set by the program.

If *PGM does not result in the appropriate protocols, this application definition field can override the protocols that are supported by this application. If at least one of the protocols that are identified here are enabled on the system by the QSSLPCL system value, protocols that are not enabled on the system are silently ignored. Where possible, follow the configuration steps that are included in the application documentation to set the protocols instead of using the application definition field. An administrator can configure weaker security properties for an IBM application than was previously possible by using this field.

Related information

TLS system value: QSSLPCL

TLS cipher specification options

The **TLS cipher specification options** application definition field determines which TLS cipher suites are supported by the application.

The default value is *PGM, which means the program that uses this "application ID" sets the supported cipher suites attribute to the appropriate value. The program might set the value explicitly through an API call or implicitly by allowing the system default to be used.

If *PGM results in the incorrect value, this field can define the TLS cipher suites that are supported by this application. Cipher suites that are disabled by the QSSLCSL system value are ignored when at least one cipher suite is enabled.

The server application controls the cipher suites that are supported by a prioritized list. A combination of security policy, performance, and interoperability considerations is used by the administrator to determine the appropriate configuration. Use caution when you consider changes to the list. The flexibility of the user-defined list allows for a weaker security configuration than might be possible with *PGM. Security can be weakened in several ways:

- Selecting a higher priority for a relatively weak encryption algorithm

- Disabling a relatively strong encryption algorithm
- Enabling a relatively weak encryption algorithm

A server is only as secure as the weakest cipher suite it allows regardless of its position in the ordered list.

Related information
TLS system value: QSSLCSL

Extended renegotiation critical mode

This field has meaning for TLSv1.2 and prior versions; it does not apply to TLSv1.3 and newer versions.

This application definition field determines whether the application requires the peer provide the RFC 5746 renegotiation indication during the initial handshake.

For more information, see TLS Extended Renegotiation Critical Mode in the “Renegotiation” on page 17 topic under “System TLS system level settings” on page 3.

The default value is *PGM, which means the program that uses this "application ID" already set the mode to the appropriate value. The program is either using the System TLS default value or a value that is set explicitly by the `gsk_attribute_set_enum()` API call for this attribute.

Set to “Enable” to require the RFC 5746 renegotiation indication is included in the initial handshake for the initial handshake to be successful. By design, this application is no longer able to handshake with peers that have not or cannot be updated to support RFC 5746.

Set this application definition field to “Disable” if the application does not require RFC 5746 renegotiation indication from the peer on initial handshake. The RFC 5746 renegotiation indication is still required for all renegotiated handshakes.

Note: The application always provides the RFC 5746 renegotiation indication information to the peer regardless of the value of this setting.

Related information
RFC 5746: "Transport Layer Security (TLS) Renegotiation Indication Extension"

Server Name Indication

The **Server Name Indication (SNI)** application definition field is used to tell System TLS to provide limited SNI support for this application.

SNI, as defined by RFC 6066, allows TLS clients to provide to the TLS server the name of the server they are contacting. This function is used to facilitate secure connections to servers that host multiple 'virtual' servers at a single underlying network address. To use SNI in this way, as either the client or the server, gsk_attribute_set_buffer() must be used to configure SNI in the application.

If limited SNI support is needed, enter the fully qualified domain name (FQDN) for the server application. If not required, accept the default value of *NONE for the limited SNI support which does not override existing SNI application configuration.

If the application configures SNI information with gsk_attribute_set_buffer(), then the value that is set for this application definition field is appended to the end of that existing information. If the existing information is configured to be critical, then this value would also be critical. Critical means that if the client FQDN does not match a name in the server list, then the server generates a failure for the session negotiation. If no existing information exists, then the limited SNI support is not critical.

A use case for setting the limited support SNI field: Your Company is contacted by a user of your services. The user has a new security requirement that all TLS servers they communicate with provide the server SNI acknowledgement. Your server is a simple server that is used for just one service with no need for virtual server configuration.

By setting the FQDN of the server, System TLS can send the SNI server acknowledgement if asked. The server certificate that is sent is the one assigned to the application definition. Nothing is changed from the server perspective, yet the peer client satisfied their requirement.

If the client requested FQDN does not match (because this field was not set or had a different value), no server SNI acknowledgment is sent. The server continues with handshake negotiation as if no SNI request was made. The client determines whether this condition is a critical error for the session negotiation.

Related information

RFC 6066: "Transport Layer Security (TLS) Extensions: Extension Definitions"

Online Certificate Status Protocol attributes

The Online Certificate Status Protocol (OSCP) attribute fields in the application definitions are used to control OCSP enablement.

OSCP is a mechanism for determining the revocation status of a certificate. See the detailed OCSP description to understand how that processing works. The GSKit APIs allow for configuring numerous attributes that determine OCSP processing.

The application definition has two OCSP attribute fields that are used to control OCSP enablement. The other OCSP attributes cannot be changed by the application definition. Those other attribute values are determined by GSKit API settings or by internal default values.

Related concepts

Online Certificate Status Protocol
Online Certificate Status Protocol (OCSP) provides applications a way to determine the revocation status for a digital certificate. Certificate revocation status that is checked via OCSP provides more up-to-date status information than is available through CRLs.

Related information

RFC 2560: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP

OCSP URL

The **Online Certificate Status Protocol (OCSP) URL** application definition field determines whether this application uses a general OCSP responder to send requests during certificate validation for end entity certificates.

When a URL is present, the specified OCSP responder is contacted for all end entity certificates to determine revocation status.

The default value for the field is *PGM meaning the program that uses this "application ID" sets the attribute to the appropriate value. All System TLS attributes have an initial default value, which for this attribute is no URL value. Programs can also call gsk_attribute_set_buffer() to explicitly set a URL value.

If *PGM does not result in the required OCSP responder, enter the appropriate OCSP responder URL in this field. HTTP is the only supported URL protocol; therefore, this value must begin with "http://". Setting this value overrides the configuration that is set internally by the program for the URL destination. However, the other configured OCSP attributes continue to be used.

If *PGM results in the application that uses an OCSP responder, yet no general OCSP responder processing is wanted, set this field to “Disable.” This setting overrides a URL internally configured by using gsk_attribute_set_buffer(). Disabling OCSP weakens the security model for the application, so use due diligence before you make this choice.

Related concepts

Certificate revocation
Certificate revocation checking is one phase of certificate validation that is done as part of session negotiation. The certificate chain is validated to ensure that the certificate is not revoked.

OCSP certificate revocation checking

The **Online Certificate Status Protocol (OCSP) Certificate Revocation Checking** application definition field determines whether OCSP certificate revocation checking is done with AIA certificate extension information and if OCSP stapling is enabled or required.

Revocation checking is done with AIA certificate extension information if OCSP revocation status is undetermined and both of the following conditions are true:

- OCSP AIA checking is enabled.
- The certificate to be validated has an AIA extension with a PKIK_AD_OCSP access method that contains a URI of the HTTP location of the OCSP responder.

  Note: The first OCSP responder that is identified in the AIA extension is queried for revocation status.

Enabling OCSP stapling on a client application causes the client to request OCSP stapling by sending the certificate status request extension as part of the client hello. Server applications with OCSP stapling enabled support the certificate status request extension and query the OCSP responder on behalf of the client when receiving the extension.

A client application can also indicate that they require OCSP stapling. This means that if the client does not receive a stapled OCSP response from the server and the server's certificate indicates that it must staple, then the client fails the secure connection.

The default value for the field is *PGM meaning the program that uses this "application ID" sets the attribute to the appropriate value. All System TLS attributes have an initial default value. For this attribute, the default value is disabled for both AIA checking and OCSP stapling. Programs can explicitly enable or disable OCSP AIA and OCSP stapling with gsk_attribute_set_enum().

If *PGM does not result in OCSP AIA validation and revocation checking is wanted, set this field to “Enable.” The internal setting is overridden and OCSP checking happens when AIA information is available.

If *PGM does not result in OCSP stapling and support is wanted, set this field to “Stapling.” The internal setting is overridden and OCSP stapling is requested by client applications and supported by server applications. Additionally, this enables OCSP AIA checking when AIA information is available on the certificate.

Setting this field to “Stapling Required” overrides the internal setting and enables OCSP AIA checking for client and server applications, enables OCSP stapling for client and server applications, and requires OCSP stapling for client applications.

If *PGM results in OCSP AIA validation or OCSP stapling, yet revocation checking is not wanted, set this field to “Disable.” Disabling OCSP weakens the security model for the application, so use due diligence before you make this choice.

Related concepts

Certificate revocation

Certificate revocation checking is one phase of certificate validation that is done as part of session negotiation. The certificate chain is validated to ensure that the certificate is not revoked.

TLS signature algorithms for key exchange

The **TLS signature algorithms** application definition field determines which algorithms are supported by the application for handshake message digital signatures.

For more information about signature algorithms and how they are used, see the “Message signature” on page 12 under “System TLS system level settings” on page 3.

The default value is *PGM, which means the program that uses this "application ID" sets the TLS signature algorithms used for handshake message signatures to the appropriate value. All System TLS attributes have an initial default value. Programs can explicitly define the list by using gsk_attribute_set_buffer().

If *PGM results in an inappropriate configuration, set this field to contain the required ordered list of supported signature algorithms.

TLS signature algorithms for certificate

The **TLS signature algorithms for certificate** application definition field determines which algorithms are supported by the application for certificate selection.

For more information about signature algorithms and how they are used, see the “Certificate selection” on page 10 under “System TLS system level settings” on page 3.

The default value is *PGM, which means the program that uses this "application ID" sets the TLS signature algorithms certificate list to the appropriate value. All System TLS attributes have an initial default value. Programs can explicitly define the list by using gsk_attribute_set_buffer().

If *PGM results in an inappropriate configuration, set this field to contain the required ordered list of supported signature algorithms for the certificate.

TLS secure session cache time to live

The **TLS secure session cache time to live** application definition field determines the session cache time out used by the application.

The default value is *PGM, which means the program that uses this "application ID" sets the secure session cache time to live to the appropriate value. All System TLS attributes have an initial default value. Programs can explicitly define this value by using gsk_attribute_set_numeric().

If *PGM results in an inappropriate configuration, set this field to contain the required secure session cache time to live.

TLSCONFIG command

TLSCONFIG is a System Service Tools (SST) Advanced Analysis command that allows viewing or altering of system-wide System TLS default properties.

To use the TLS configuration IBM-supplied command support, follow these steps:

1. Access System Service Tools by using the CL command Start System Service Tools (**STRSST**).
2. Select **Start a service tool**.
3. Select **Display/Alter/Dump**.
4. Select **Display/Alter storage**.
5. Select **Licensed Internal Code (LIC) data**.
6. Select **Advanced analysis**. (You must page down to see this option.)
7. Page down until you find the TLSCONFIG option. Then, place a 1 (Select) next to the option and press Enter. You are now on the Specify Advanced Analysis Options window. The command shows as **TLSCONFIG**.
8. Enter '-h' without the quotation marks and press Enter to display the available options.

If auditing is enabled for **TLSCONFIG**, it generates a C3 (Advanced Analysis Command Configuration) journal entry with an entry type of A (TLSCONFIG advanced analysis changes) when a security change is successfully made. Auditing for TLSCONFIG is enabled when *SECURITY or *SECCFG is set for the QAUDLVL/QAUDLVL2 system value. For more information on the C3 journal entry, see C3 (Advanced Analysis Command Configuration) journal entries.

Retrieve TLS Attributes

The Retrieve TLS Attributes (QsoRtvTLSA) API allows the retrieval of the system-wide System TLS current default properties.

The properties can be changed and viewed with **TLSCONFIG**. The benefit of QsoRtvTLSA is it can be included in programs or scripts for health check actions.

Related concepts

TLSCONFIG command
TLSCONFIG is a System Service Tools (SST) Advanced Analysis command that allows viewing or altering of system-wide System TLS default properties.

Related information

Retrieve TLS Attributes (QsoRtvTLSA) API

How to determine what System TLS protocols and cipher suites are used on the system

The System TLS protocols and cipher suites that are used on the system can be determined by audit or trace functions available on the system.

Audit journal

The protocol and cipher suite for a System TLS connection are captured when secure socket connections are audited.

Secure socket connection auditing is enabled when *NETSECURE is set for the QAUDLVL/QAUDLVL2 system value. Each successful connection generates an SK (Sockets Connections) journal entry with an entry type of S (Successful secure connection). The journal entry contains the protocol information in the "Secure version" field and the cipher suite in the "Secure properties" field.

In the following example SK journal entry type S, the "Secure version" field indicates TLSv1.2 was negotiated along with cipher suite TLS_RSA_WITH_AES_128_CBC_SHA256 shown in the "Secure properties" field. The signature algorithm used was ECDSA_SHA512, which is also in the "Secure properties" field.

```
           Entry specific data
Column      *...+....1....+....2....+....3....+....4....+....5
00901      '                                                   '
00951      '          TLSV1.2   TLS_RSA_WITH_AES_128_CBC_SHA25'
01001      '6 ECDSA_SHA512                                     '
                                                          More...
```

In the following example SK journal entry type S, the "Secure version" and the "Secure properties" fields indicate TLSv1.2 was negotiated along with cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384. The signature algorithm was ECDSA_SHA512 and the named elliptic curve, which determines key size, was Secp521r1.

```
           Entry specific data
Column      *...+....1....+....2....+....3....+....4....+....5
00901      '                                                   '
00951      '          TLSV1.2   TLS_ECDHE_ECDSA_WITH_AES_256_G'
01001      'CM_SHA384 ECDSA_SHA512 SECP521R1                   '
                                                          More...
```

For more information about interpreting all the SK entry fields, see SK (Sockets Connections) journal entries. For more information about analyzing the logged events, see Analyzing audit journal entries in the Security reference topic.

Related information

Copy Audit Journal Entries (CPYAUDJRNE)

LIC trace

The Trace Licensed Internal Code (LIC) service tool can capture a System TLS trace point that contains the System TLS protocols and cipher suites. The Trace Internal (TRCINT) command is the command interface to the Trace LIC Service tool.

To trace all protocol versions, issue the following command to start the trace:

```
TRCINT SET(*ON) TRCTBL('SCKTLS-1700x') TRCTYPE(*SCKSSL) SLTTRCPNT((17000 17009))
```

Wait for new secure connections to establish and end the trace with the command:

```
TRCINT SET(*OFF) TRCTBL('SCKTLS-1700x') OUTPUT(*PRINT)
```

To delete the trace, issue the following command:

```
TRCINT SET(*END) TRCTBL('SCKTLS-1700x')
```

To trace specific protocol version connections, use one or more of the following trace points.

Protocol Version	Trace Identifier
TLSv1.3	17005
TLSv1.2	17004
TLSv1.1	17003
TLSv1.0	17002
SSLv3	17001

For example, to find only TLSv1.0 connections use trace point 17002:

```
TRCINT SET(*ON) TRCTBL('SCKTLS-17002') TRCTYPE(*SCKSSL) SLTTRCPNT((17002))
```

To trace a range of protocol versions, specify the beginning Trace ID followed by the end Trace ID. This example illustrates tracing SSLv3 through TLSv1.1.

```
TRCINT SET(*ON) TRCTBL('SCKTLS-1700x') TRCTYPE(*SCKSSL) SLTTRCPNT((17001 17003))
```

A spooled file named QPCSMPRT is created for the user that ran the `TRCINT SET(*OFF)` command. Submit the `TRCINT SET(*OFF)` command to a background job when you are managing a large trace capture. The following trace point output outlines the connection properties included in the trace point.

```
SOCKETS              IDENTIFIER : SC#17003                          TIME 02/17/15  11:03:33.151908   TDE# 000000003C94
  #1    (   21)    +0000   C3D6D5D5C5C3E3C9  D6D540D7D9D6D7C5   D9E3C9C5E2                    *CONNECTION PROPERTIES
  #2    (    7)    +0000   E3D3E2E5F14BF1                                                  *TLSV1.1
  #3    (   28)    +0000   E3D3E26DD9E2C16D  E6C9E3C86DC1C5E2   6DF1F2F86DC3C2C3  6DE2C8C1   *TLS_RSA_WITH_AES_128_CBC_SHA
  #4    (   10)    +0000   D3D6C3C1D340D7D6  D9E3                                           *LOCAL PORT
  #5    (    3)    +0000   F9F9F2                                                         *992
  #6    (   16)    +0000   D3D6C3C1D340C9D7  40C1C4C4D9C5E2E2                             *LOCAL IP ADDRESS
  #7    (   20)    +0000   7A7A868686867AF1  F9F84BF5F14BF1F0   F04BF1F5                   *::ffff:198.51.100.15
  #8    (   11)    +0000   D9C5D4D6E3C540D7  D6D9E3                                       *REMOTE PORT
  #9    (    5)    +0000   F6F1F8F5F2                                                     *61852
  #10   (   17)    +0000   D9C5D4D6E3C540C9  D740C1C4C4D9C5E2   E2                       *REMOTE IP ADDRESS
  #11   (   20)    +0000   7A7A868686867AF1  F9F84BF5F14BF1F0   F04BF1F6                   *::ffff:198.51.100.16
  #12   (   16)    +0000   E3D5C1C3C3C5D7E3  E3C1E2D240404040                             *TNACCEPTTASK
  #13   (   22)    +0000   D8C9C2D46DD8E3E5  6DE3C5D3D5C5E36D   E2C5D9E5C5D9              *QIBM_QTV_TELNET_SERVER
```

The following information is in the trace point entry data:

- Protocol Negotiated
- Cipher suite Negotiated
- Local port and IP address
- Remote port and IP address
- Job/Task/Device name
- Application ID (if used)

System TLS protocol version counters

The System Service Tools Advanced Analysis command **TLSCONFIG** can be used to turn on System TLS protocol version counters. The counters show protocols that are actively being negotiated by System TLS.

If you want to determine the System TLS protocols that are used on your system, you can use **TLSCONFIG** option `connectionCounts`. When enabled, **TLSCONFIG** option `connectionCounts` keeps a running count of new System TLS connections that are grouped by the negotiated TLS protocol. There is a slight performance cost to count the connections.

TLSCONFIG option `-h` displays the help panel that describes how to use **TLSCONFIG** option `connectionCounts`.

The following steps can be used to determine whether a particular protocol is used on your system before you disable support for the protocol.

1. Reset the `connectionCounts` to clear the current protocol version counts.

   ```
   TLSCONFIG -connectionCounts:reset
   ```

2. Track the System TLS connections to determine which protocols are used for active connections.

   ```
   TLSCONFIG -connectionCounts:enable
   ```

3. After the connection counts run over an interval that exhibits normal System TLS traffic on your system, display the number of TLS connections by protocol type since the last reset.

   ```
   TLSCONFIG -connectionCounts:display
   ```

4. Determine what applications are using the protocols that you would like to disable. Update the application's configuration to no longer use these protocols.

   Note: The count does not provide guidance as to which application is using a particular protocol. For more information about how to determine what application uses a particular protocol, see "How to determine what System TLS protocols and cipher suites are used on the system" on page 30.

5. Reset the `connectionCounts` to clear the current protocol version counts.

   ```
   TLSCONFIG -connectionCounts:reset
   ```

6. After another interval that exhibits normal System TLS traffic on your system, display the number of TLS connections by protocol type since the last reset.

   ```
   TLSCONFIG -connectionCounts:display
   ```

   If the protocol to disable has a connection count of 0, you know that protocol version was not used during the monitored interval.

7. Turn off TLS connection counting.

   ```
   TLSCONFIG -connectionCounts:disable
   ```

Troubleshooting TLS

This basic troubleshooting information is intended to help you reduce the list of possible problems that System TLS can encounter.

It is important to understand this information is not a comprehensive source for troubleshooting information, but rather a guide to aid in common problem resolution.

Verify that the following statements are true:

- Your IBM i meets all the "TLS prerequisites" on page 34.
- Your certificate authority and certificates are valid and are not expired. For more information about validating certificates, see Validating certificates and applications in Digital Certificate Manager.
- You verified that your IBM i and the remote server both support at least one of the same TLS protocols and ciphers suites.
  - Review the System TLS system values: QSSLPCL, QSSLCSLCTL, and QSSLCSL
  - Check the "DCM application definitions" on page 24 to identify the TLS protocols and cipher suites that are enabled for the application
  - Review settings in the System Service Tools (SST) Advanced Analysis command **TLSCONFIG**
    - Use **TLSCONFIG** option `display` to view the default TLS attributes
  - Review application-specific TLS attributes

More problem determination options:

- The TLS error code in the server job log can be cross referenced in an error table to find more information about the error. For example, this table maps the -93 that might be seen in a server job log to the constant SSL_ERROR_SSL_NOT_AVAILABLE.
  - A negative return code (indicated by the dash before the code number) indicates that the application used an SSL_ API.
  - A positive return code indicates that the application used a GSKit API.

  If more detailed information is required, the message ID provided in the table can be displayed on an IBM i to show potential cause and recovery for this error. More information that explains these error codes might be in the individual documentation for the secure socket API that returned the error.
- The following two header files contain the same constant names for System TLS return codes as the table, but without the message ID cross-reference:
  - QSYSINC/H.GSKSSL
  - QSYSINC/H.QSOSSL

  Remember although the names of the System TLS return codes remain constant in these two files, more than one unique error can be associated with each return code.

If you think an application is failing during TLS negotiation but is not logging or otherwise providing error details, you can try to find more information by using auditing.

System TLS failed connections are captured when secure socket connection auditing is enabled. Secure socket connection auditing is enabled when *NETSECURE is set for the QAUDLVL/QAUDLVL2 system value. Each failed connection generates an SK (Sockets Connections) journal entry with an entry type of X (Failed System TLS connection). The “Secure information” field contains a string that describes the failure.

In the following example Sockets Connection journal entry type X, the "Secure version" field indicates that TLSv1.2 protocol was used to try to negotiate a secure handshake. However, the handshake failed with error code GSK_ERROR_NO_CIPHERS, which is shown in the "Secure properties" field. The "Secure information" field contains a string that describes the failure.

```
            Entry specific data
Column      *...+....1....+....2....+....3....+....4....+....5
00901      '                                                  '
00951      '           TLSV1.2   GSK_ERROR_NO_CIPHERS         '
01001      '                                                  '
01051      '                     No compatible cipher suite ava'
01101      'ilable between SSL end points.                    '
01151      '                     '
```

For more information about all of the SK entry fields, see SK (Sockets Connections) journal entries.

Application problem determination options:

- Programmers can code the gsk_strerror() or SSL_Strerror() API in their programs to obtain a brief description of an error return code. Some applications use this API and print a message to the job log containing the brief error description.
- Additional information about the last certificate validation error on the current secure session can be retrieved by using the GSK_LAST_VALIDATION_ERROR attribute on gsk_attribute_get_numeric_value(). If gsk_secure_soc_init() or gsk_secure_soc_startInit() returned an error, this attribute might provide more error information.

Related concepts

TLS prerequisites

This topic describes the prerequisites for System TLS on the IBM i, as well as some helpful tips.

Related information

Secure socket API error code messages

TLS prerequisites

This topic describes the prerequisites for System TLS on the IBM i, as well as some helpful tips.

Verify that you have the following options installed before you use TLS:

- IBM Digital Certificate Manager (DCM) (5770-SS1 Option 34)

  Note: IBM Java Secure Socket Extension (JSSE), Oracle JSSE, and OpenSSL do not require DCM.
- IBM TCP/IP Connectivity Utilities for i (5770-TC1)
- IBM HTTP Server for i (5770-DG1)
- If you are trying to access the IBM Navigator for i or DCM applications on port 2006, ensure that you have the required IBM Developer Kit for Java (5770-JV1) product options installed. Otherwise, the IBM i HTTP admin server does not start. The required product options are defined by the IBM HTTP Server group PTF cover letter.

Useful information:

- The best System TLS performance is obtained by running on Power9 or newer hardware.

Related concepts

Troubleshooting TLS
This basic troubleshooting information is intended to help you reduce the list of possible problems that System TLS can encounter.

Related information

Cryptographic hardware
Public certificates versus private certificates
Configure DCM

Application security with TLS

A number of applications on IBM i can be secured with TLS. Refer to each application's documentation for details.

Related information

Enterprise Identity Mapping
Securing File Transfer Protocol
Setting up TLS for the administration (ADMIN) server for HTTP Server
Add TLS protection on HTTP Server
Setting up TLS for IBM Navigator for i
Telnet scenario: Secure Telnet with TLS
Secure Sockets Layer (SSL) and Transport Layer Security (TLS) with the Directory Server
Secure Sockets APIs
Scenario: Protect private keys with cryptographic hardware

Related information for TLS

Use this information to learn about other resources and information relevant to using Transport Layer Security(TLS).

Web sites

- RFC 8446: "The Transport Layer Security (TLS) Protocol Version 1.3" (https://www.ietf.org/rfc/rfc8446.txt)
- RFC 5246: "The Transport Layer Security (TLS) Protocol Version 1.2" (https://www.ietf.org/rfc/rfc5246.txt)

- RFC 4346: "The Transport Layer Security (TLS) Protocol Version 1.1" (https://www.ietf.org/rfc/rfc4346.txt)
- RFC 2246: "The TLS Protocol Version 1.0 " (https://www.ietf.org/rfc/rfc2246.txt)
- RFC2818: "HTTP Over TLS" (https://www.ietf.org/rfc/rfc2818.txt)
- RFC 5746: "Transport Layer Security (TLS) Renegotiation Indication Extension" (https://www.ietf.org/rfc/rfc5746.txt)
- RFC 6066: "Transport Layer Security (TLS) Extensions: Extension Definitions" (https://www.ietf.org/rfc/rfc6066.txt)
- RFC 2560: "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP" (https://www.ietf.org/rfc/rfc2560.txt)
- RFC 4492: "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)" (https://www.ietf.org/rfc/rfc4492.txt)

Other information

- SSL and Java Secure Socket Extension
- IBM Toolbox for Java

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department YBWA
2800 37th Street NW
Rochester, MN 55901-4441
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks of Oracle, Inc. in the United States, other countries, or both.

Other product and service names might be trademarks of IBM or other companies.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

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.

Product Number: 5770-SS1