z/OS
Cryptographic Services
Integrated Cryptographic Service Facility

System Programmer's Guide

SA22-7520-16

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 325.

This edition applies to Version 1 Release 13 of z/OS (5694–A01) and to all subsequent releases and modifications until otherwise indicated in new editions. This edition applies to ICSF FMID HCR7790.

This edition replaces SA22-7520-15.

Contents

Figures

Tables

About this information

This information supports z/OS (5694-A01). It describes how to initialize, customize, operate, and diagnose the z/OS Integrated Cryptographic Service Facility (ICSF). The z/OS Cryptographic Services includes these components:

- z/OS Integrated Cryptographic Service Facility (ICSF)
- z/OS Open Cryptographic Services Facility (OCSF)
- z/OS System Secure Socket Level Programming (SSL)
- z/OS Public Key Infrastructure Services (PKI)

ICSF is a software element of z/OS that works with the hardware cryptographic feature and the Security Server (RACF) to provide secure, high-speed cryptographic services. ICSF provides the application programming interfaces by which applications request the cryptographic services.

Who should use this information

This information is intended for the system programmer. It describes the tasks that a system programmer might perform:

- Programming installation options, installation-defined callable services, and installation exits
- Creating the data sets that ICSF uses
- Migrating the system from the Cryptographic Unit Support Program (CUSP) and Programmed Cryptographic Facility (PCF) to ICSF
- Migrating to z/OS ICSF
- Migrating from the IBM Network Security Processor Support Program (hereafter called 4753-HSP) to ICSF
- Starting and stopping ICSF
- Checking event recording
- Planning for security and performance considerations
- Debugging and recovering from problems

Defining and writing installation-defined callable services and installation exit routines is intended to be accomplished primarily by experienced system programmers. This information assumes that the reader has an advanced knowledge of z/OS.

How to use this information

This information is divided into descriptions of these tasks:

- Introducing ICSF
  - Chapter 1, “Introduction to z/OS ICSF,” on page 1, introduces the cryptographic key data set (CKDS), the public key data set (PKDS) and the token data set (TKDS) and provides basic information about running PCF and 4753-HSP applications on ICSF and preparing for installation.
- Initializing ICSF
  - Chapter 2, “Installation, Initialization, and Customization,” on page 13, describes how to customize SYS1.PARMLIB, create the CKDS, the PKDS, and the TKDS, the installations options data set, the startup procedure, and provide access to the ICSF panels. It also explains how to setup for SMP/E

electronic delivery, change the parameters in the installation options data set after the first start and introduces installation exits.

- Migration and coexistence issues

  Chapter 8, “Migration from PCF to z/OS ICSF,” on page 173, describes how to migrate application programs and cryptographic key data set information to z/OS ICSF from the IBM cryptographic products CUSP/PCF.

  Chapter 3, “Migration,” on page 53, describes migration to z/OS ICSF from previous releases of ICSF.

  Chapter 9, “Compatibility and Coexistence of 4753-HSP and ICSF,” on page 191, gives a brief overview of migrating 4753-HSP key storage to ICSF CKDS.
- Customizing ICSF
  - Chapter 6, “Installation-Defined Callable Services,” on page 159 gives information that an experienced system programmer can use to write installation-defined callable services. It also explains how to define these callable services to ICSF, and how to write service stubs to access them.
  - Chapter 5, “Installation Exits,” on page 111, describes the ICSF installation exits you can use to customize ICSF.
- Operating ICSF
  - Chapter 4, “Operating ICSF,” on page 75, describes how to add and remove cryptographic coprocessors and to start, modify, and stop ICSF and other operating considerations.
  - “Event Recording” on page 88, describes ICSF event recording on the Security Console and SMF.
- Planning ICSF
  - “Security Considerations” on page 98, describes methods you can use to protect ICSF resources.
- Diagnosing ICSF
  - “Debugging Aids” on page 101, describes the use of component trace and Interactive Problem Control System (IPCS) to debug ICSF.
  - Appendix A, “Diagnosis Reference Information,” on page 193, maps the cryptographic key data set and the cryptographic communication vector tables as reference information for use in debugging. This appendix also maps DES and PKA key tokens.
  - Appendix B, “ICSF SMF Records,” on page 283 describes SMF Record type 82, which is used to record information about the events and operations of ICSF. Record type 82 is written to the SMF data set at the completion of certain cryptographic functions.
  - Appendix C, “CICS-ICSF Attachment Facility,” on page 299, defines steps to install the CICS-ICSF Attachment Facility.
  - Appendix D, “Helpful Hints for ICSF First Time Startup,” on page 305, defines helpful hints and that you may encounter when starting ICSF for the first time.
  - Appendix E, “Using AMS REPRO Encryption,” on page 317, provides information on using IDCAMS REPRO ENCIPHER and DECIPHER options with ICSF.
  - Appendix F, “z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, or z196 without a PCIXCC, CEX2C, or CEX3C,” on page 319 describes processing and functionality support for this environment.
  - Appendix G, “Accessibility,” on page 323 contains information on accessibility features in z/OS.

– “Notices” on page 325 contains information on notices, programming interface information and trademarks.

Where to find more information

The publications in the z/OS ICSF library include:

- *z/OS Cryptographic Services ICSF Overview*
- *z/OS Cryptographic Services ICSF Administrator's Guide*
- *z/OS Cryptographic Services ICSF System Programmer's Guide*
- *z/OS Cryptographic Services ICSF Application Programmer's Guide*
- *z/OS Cryptographic Services ICSF Messages*
- *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*
- *z/OS Cryptographic Services ICSF TKE Workstation User's Guide*

This publication also refers to these publications:

- *IBM ES/3090 Processor Complex Recovery Guide*
- *z/OS Planning for Installation*, GA22-7504
- *z/OS Security Server RACF Auditor's Guide*
- *z/OS Security Server RACF Command Language Reference*
- *z/OS Security Server RACF Security Administrator's Guide*
- *z/OS Security Server RACF Macros and Interfaces*
- *z/OS Security Server RACF System Programmer's Guide*
- *z/OS MVS IPCS User's Guide*, SA22-7596
- *z/OS MVS System Codes*, SA22-7626
- *z/OS MVS System Management Facilities (SMF)*, SA22-7630
- *z/OS MVS Programming: Extended Addressability Guide*, SA22-7614
- *z/OS MVS Initialization and Tuning Guide*, SA22-7591
- *z/OS MVS Initialization and Tuning Reference*, SA22-7592
- *MVS Batch Local Shared Resources*, GC28-1469
- *z/OS DFSMS Access Method Services for Catalogs*
- *z/OS DFSMS Using Data Sets*
- *IBM Transaction Security System: General Information Manual and Planning Guide*
- *IBM Transaction Security System: Concepts and Programming Guide: Volume 1, Access Controls and DES Cryptography*
- *IBM Transaction Security System: Basic CCA Cryptographic Services*
- *IBM Transaction Security System: Concepts and Programming Guide: Volume II, Public-Key Cryptography*
- *IBM Distributed Key Management System, Installation and Customization Guide*
- *OS/VS1 and OS/VS2 MVS Cryptographic Unit Support: Installation Manual*
- *OS/VS1 and OS/VS2 MVS Programmed Cryptographic Facility*
- *CICS Customization Guide*, SC34-6227
- *CICS Resource Definition Guide*, SC34-6228

Do You Have Problems, Comments, or Suggestions?

Your suggestions and ideas can contribute to the quality and the usability of this document. If you have problems using this document, or if you have suggestions for improving it, complete and mail the Reader's Comment Form found at the back of the document.

How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or give us any other feedback that you might have.

Use one of the following methods to send us your comments:

1. Send an email to mhvrcfs@us.ibm.com
2. Visit the Contact z/OS web page at http://www.ibm.com/systems/z/os/zos/webqs.html
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRCFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   U.S.A.
4. Fax the comments to us as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:

- Your name and address
- Your email address
- Your telephone or fax number
- The publication title and order number:
  z/OS Cryptographic Services ICSF System Programmer's Guide
  SA22-7520-16
- The topic and page number related to your comment
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

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

If you have a technical problem

Do not use the feedback methods listed above. Instead, do one of the following:

- Contact your IBM service representative
- Call IBM technical support
- Visit the IBM zSeries support web page at http://www.ibm.com/systems/z/support/

Summary of changes

Changes made in z/OS Version 1 Release 13

This document contains information previously presented in *z/OS ICSF System Programmer's Guide*, SA22-7520-15, which supports z/OS Version 1 Release 12.

This document is for ICSF FMID HCR7790. This release runs on z/OS V1R11, z/OS V1R12, and z/OS V1R13 and only on zSeries hardware.

New information:

- Exit identifiers for new callable services described in Table 3 on page 42.
- New system abend codes summarized in “System Abend Codes” on page 68.
- Migration action for moving from a version of ICSF prior to FMID HCR7780 to FMID HCR7780 or HCR7790 described in “Ensure the expected master key support is available” on page 60
- A new message, CSFM540I, was added to indicate a card has been fenced off during ICSF initialization. See notes in “Starting and stopping ICSF” on page 76 for more information.
- A new health check, ICSF_COPROCESSOR_STATE_NEGCHANGE, was added to detect a degradation in the state of a coprocessor or accelerator. See notes in “Starting and stopping ICSF” on page 76 for more information.
- A new health check, ICSFMIG_DEPRECATED_SERV_WARNINGS, was added to detect the use of a services that will not be supported in subsequent releases. See “Migrating from the IBM @server zSeries 900” on page 70 for more information.

Changed information:

- Changes to callable services summarized in Table 4 on page 54.
- The process to reencipher the PKDS and change the RSA master key has changed for z196 systems with CEX3C coprocessors and the Sep. 2011 licensed internal code (LIC). See “Changing the RSA master key” on page 64.
- For clarity:
  - CSNBKRC and CSNEKRC, which had been referred to as the "Key Record Create" service, are now referred to as the "CKDS Key Record Create" service
  - CSNBKRC2 and CSNEKRC2, which had been referred to as the "Key Record Create2" service, are now referred to as the "CKDS Key Record Create2" service
  - CSNBKRD and CSNEKRD, which had been referred to as the "Key Record Delete" service, are now referred to as the "CKDS Key Record Delete" service
  - CSNBKRR and CSNEKRR, which had been referred to as the "Key Record Read" service, are now referred to as the "CKDS Key Record Read" service
  - CSNBKRR2 and CSNEKRR2, which had been referred to as the "Key Record Read2" service, are now referred to as the "CKDS Key Record Read2" service
  - CSNBKRW and CSNEKRW, which had been referred to as the "Key Record Write" service, are now referred to as the "CKDS Key Record Write" service
  - CSNBKRW2 and CSNEKRW2, which had been referred to as the "Key Record Write2" service, are now referred to as the "CKDS Key Record Write2" service

- CSNDKRC and CSNFKRC, which had been referred to as the "PKDS Record Create" service, are now referred to as the "PKDS Key Record Create" service
- CSNDKRD and CSNFKRD, which had been referred to as the "PKDS Record Delete" service, are now referred to as the "PKDS Key Record Delete" service
- CSNDKRR and CSNFKRR, which had been referred to as the "PKDS Record Read" service, are now referred to as the "PKDS Key Record Read" service
- CSNDKRW and CSNFKRW, which had been referred to as the "PKDS Record Write" service, are now referred to as the "PKDS Key Record Write" service

Changes made in z/OS Version 1 Release 12

This document contains information previously presented in *z/OS ICSF System Programmer's Guide*, SA22-7520-14, which supports z/OS Version 1 Release 11.

This document is for ICSF FMID HCR7780. This release runs on z/OS V1R10, z/OS V1R11, and z/OS V1R12 and only on zSeries hardware.

New information:

- Added support for IBM zEnterprise 196 (z196) servers.
- Added support for Elliptic Curve Cryptography (ECC).
- Added information on HMAC key support. HMAC key support is to be enabled with the PTF for APAR OA33260, planned for February 2011 availability.

  To accommodate HMAC keys, added a new variable length record format for the CKDS. The variable length record format is only required if HMAC keys are to be stored in the CKDS. The variable length record format can be used to store all existing symmetric keys and the new HMAC keys. Added a CKDS conversion program, CSFCNV2, that converts a fixed length record format CKDS to a variable length record format.
- Added a new, X9.24 compliant CBC wrapping method for DES keys. Added a new installation option, DEFAULTWRAP, to indicate whether the original CAA token wrapping method or the new CBC wrapping method should be used by default.
- Added new SMF type 82 subtype 29 - TKE Workstation Audit Record

Changed information:

- Changed SMF records to include an auditing header, and audit sections for server user and end user. Modified:
  - SMF type 82 subtype 1 - Initialization
  - SMF type 82 subtype 14 - Cryptographic Coprocessor Clear Master Key Entry
- Modified the Cryptographic Communication Vector Table (CCVT) and the Cryptographic Communication Vector Table Extension (CCVE)

Changes made in z/OS Version 1 Release 11

This document contains information previously presented in *z/OS ICSF System Programmer's Guide*, SA22-7520-13, which supports z/OS Version 1 Release 10.

This document is for ICSF FMID HCR7770. This release runs on z/OS V1R9, z/OS V1R10, and z/OS V1R11 and only on zSeries hardware.

New information:

- Added information for Crypto Express3 feature (CEX3C and CEX3A)
- Added new installation option FIPSMODE.
- Added support for new callable services:
  - Added new callable service PKA Key Translate (CSNDPKT and CSNFPKT). Using this callable service, applications can translate a source CCA RSA key token into a target external smart card key token.
  - Added new callable services for managing PKCS #11 tokens and objects. These additional services are:
    - PKCS #11 Derive key (CSFPDVK)
    - PKCS #11 Derive multiple keys (CSFPDMK)
    - PKCS #11 Generate HMAC (CSFPHMG)
    - PKCS #11 Generate key pair (CSFPGKP)
    - PKCS #11 Generate secret key (CSFPGSK)
    - PKCS #11 One-way hash generate (CSFPOWH)
    - PKCS #11 Private key sign (CSFPPKS)
    - PKCS #11 Pseudo-random function (CSFPPRF)
    - PKCS #11 Public key verify (CSFPPKV)
    - PKCS #11 Secret key decrypt (CSFPSKD)
    - PKCS #11 Secret key encrypt (CSFPSKE)
    - PKCS #11 Unwrap key (CSFPUWK)
    - PKCS #11 Verify HMAC (CSFPHMV)
    - PKCS #11 Wrap key (CSFPWPK)
- Added information for new SMF type 82 subtype records:
  - SMF type 82 subtype 7 - KEU Key Part Entry Section
  - SMF type 82 subtype 14 - Cryptographic Coprocessor Clear
  - SMF type 82 subtype 15 - PCI Cryptographic Coprocessor
  - SMF type 82 subtype 16 - PCI Cryptographic Coprocessor TKE
  - SMF type 82 subtype 18 - Cryptographic Coprocessor
  - SMF type 82 subtype 20 - Cryptographic Coprocessor
  - SMF type 82 subtype 27 - PKA Key Management Extensions
  - SMF type 82 subtype 28 - High Performance Encrypted Key
- Added information for using DISPLAY GRS properly to understand ICSF key data set serialization conditions, and to detect the possible ICSF client application workload that could be impeding ongoing ICSF operation.

Changed information:

- Modified token data set (TKDS) format information
- Modified the Cryptographic Communication Vector Table (CCVT) and the Cryptographic Communication Vector Table Extension (CCVE)
- Modified ICSF startup procedure to support new startup program.
- Modified the supported boundary values for the TRACEENTRY parameter in the installation options data set. The TRACEENTRY parameter now specifies a decimal value from 10000 to 500000.

This document contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Chapter 1. Introduction to z/OS ICSF

ICSF is a software element of z/OS. ICSF works with the hardware cryptographic features and the Security Server (RACF element) to provide secure, high-speed cryptographic services in the z/OS environment. ICSF provides the application programming interfaces by which applications request the cryptographic services. ICSF is also the means by which the secure cryptographic features are loaded with master key values, allowing the hardware features to be used by applications. The cryptographic feature is secure, high-speed hardware that performs the actual cryptographic functions. Your processor hardware determines the cryptographic feature available to your applications.

Hardware Features

This topic describes the cryptographic hardware features available. Information on adding and removing cryptographic coprocessors can be found in *z/OS Cryptographic Services ICSF Administrator's Guide*.

Cryptographic Hardware

This topic describes the cryptographic hardware features available. Information on adding and removing cryptographic coprocessors can be found in *z/OS Cryptographic Services ICSF Administrator's Guide*.

Crypto Express3 Feature (CEX3C or CEX3A)

The Crypto Express3 Feature is an asynchronous cryptographic coprocessor or accelerator. The feature contains two cryptographic engines that can be independently configured as a coprocessor (CEX3C) or as an accelerator (CEX3A). It is available on the IBM System z10 Enterprise Class, IBM System z10 Business Class, and the IBM zEnterprise 196.

Crypto Express2 Feature (CEX2C or CEX2A)

The Crypto Express2 Feature is an asynchronous cryptographic coprocessor or accelerator. The feature contains two cryptographic engines that can be independently configured as a coprocessor (CEX2C) or as an accelerator (CEX2A). It is available on the IBM System z9 Enterprise Class, IBM System z9 Business Class, IBM System z10 Enterprise Class, and IBM System z10 Business Class.

PCI X Cryptographic Coprocessor (PCIXCC)

The PCI X Cryptographic Coprocessor is an asynchronous cryptographic coprocessor. It is a replacement for the Cryptographic Coprocessor Feature and PCI Cryptographic Coprocessor. It is only available on a IBM @server zSeries 990 or IBM @server zSeries 890.

The PCIXCC/CEX2C DES master key is used in place of the CCF DES master key. The asymmetric-keys master key is used in place of the CCF signature and key management master keys. The PCIXCC/CEX2C supports up to 2048-bit RSA keys in all PKA services except SET services (Set Block Compose and Set Block Decompose).

This feature is in the process of being certified for Federal Information Processing Standard (FIPS) 140-2. This includes algorithmic certification under FIPS 46-2 (DES) and FIPS 180-1 (Secure Hash Standard).

CP Assist for Cryptographic Functions (CPACF)

CPACF is a set of cryptographic instructions available on all CPs of z990, z890, z9 EC, z9 BC, z10 EC and z10 BC. Use of the CPACF instructions provides improved performance. The SHA-1 algorithm is always available. Additionally, SHA-224 and SHA-256 algorithms are available on the z9 EC and z9 BC. Additionally, SHA-384 and SHA-512 algorithms are available on IBM System z10 Enterprise Class and IBM System z10 Business Class.

CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement, feature 3863, provides for clear key DES and TDES instructions. On the z9 EC and z9 BC, this feature includes clear key AES for 128-bit keys. On IBM System z10 Enterprise Class and IBM System z10 Business Class, this feature also includes clear key AES for 192-bit and 256-bit keys.

If you want to include a PCIXCC, CEX2C, PCICA (z990, z890), Crypto Express2 feature (z9 EC, z9 BC, z10 EC and z10 BC), or Crypto Express3 Coprocessor (z10 EC, z10 BC, and z196), then feature 3863 is required.

PCI Cryptographic Accelerator (PCICA)

On all systems, the PCI Cryptographic Accelerator provides support for clear keys in the CSNDPKD callable services for better performance than when executed in a cryptographic coprocessor. On z990 or z890, it also supports CSNDDSV and CSNDPKE.

PCICAs enable maximum SSL performance.

Cryptographic Coprocessor Feature (CCF)

The Cryptographic Coprocessor Feature (CCF) can have up to two cryptographic coprocessors as high-speed extensions of the central processor. Each CCF contains both DES and PKA cryptographic processing units. You can configure the processor complex to run in either single-image mode or logical partition mode.

If the Cryptographic Coprocessor Feature is in single-image mode, the same master keys must be installed on both CCFs. If you bring a second coprocessor online, ICSF verifies that the master keys are the same. If the DES master keys are different, ICSF will not use the second coprocessor. The PKA master keys must be the same on both Coprocessors in order to enable the PKA services.

This feature is currently certified for Federal Information Processing Standard (FIPS) 140-1 level 4. This includes algorithmic certification under FIPS 46-2 (DES), FIPS 180-1 (Secure Hash Standard), and FIPS 186 (Digital Signature Standard).

The possible configurations include:

- DES with PKA

  These servers are configured for full 64-bit DES keys (effective length 56 bits), 1024-bit PKA keys for DES key distribution, and 1024-bit PKA signature keys.
- Triple DES with PKA

  These servers are configured for 192-bit DES keys (effective length 169 bits), 1024-bit PKA keys for DES key distribution, and 1024-bit PKA signature keys. This configuration is available on S/390 G5 Enterprise Servers and higher.

PCI Cryptographic Coprocessor (PCICC)

The PCI Cryptographic Coprocessor, which works in conjunction with the Cryptographic Coprocessor Feature, provides the capability of generating and

retaining RSA keys in secure hardware. This capability meets a requirement to become a SET Certificate Authority. A PCI Cryptographic Coprocessor is required on a CCF system for:

- UDX capability
- Generating RSA public and private keys
- The retained key list and retain key delete callable service.

The PCICC cards are in addition to the Cryptographic Coprocessor Feature. In order for the PCI Cryptographic Coprocessor to operate, the verification pattern for the SYM-MK master key must match the verification pattern of the DES master key on the server's Cryptographic Coprocessor Feature. Before you can use the PKA services of the PCI Cryptographic Coprocessor, you must install both the KMMK and the SMK on the Cryptographic Coprocessor Feature and the RSA-MK master key on the PCI Cryptographic Coprocessor. The hash pattern of the RSA-MK master key must match the hash pattern of the SMK in order to use the PCI Cryptographic Coprocessor.

Note: For new installations, it is recommended that the installation enter the KMMK equal to the SMK master key. Existing customers should reencipher their PKDS and migrate to a system with the KMMK equal to the SMK.

Server Hardware

This topic describes the servers on which the cryptographic hardware features are available.

IBM zEnterprise 196 (z196)

The z196 provides constraint relief and addresses various customer demands. It has several cryptographic features.

- **CP Assist for Cryptographic Functions** is implemented on every processor. SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512 secure hashing is directly available to application programs.
- Feature code 3863, **CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement** – enables clear key DES and TDES instructions on all CPs. AES 128-bit, AES 192-bit and AES 256-bit support is also available.
- Feature code 0864, **Crypto Express3 Feature** – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The z10 EC and z10 BC can support a maximum of 8 features. Each feature code has two coprocessors/accelerators.

IBM System z10 Enterprise Class and IBM System z10 Business Class (z10 BC)

The IBM System z10 Enterprise Class and IBM System z10 Business Class provide constraint relief and addresses various customer demands. It has several cryptographic features.

- **CP Assist for Cryptographic Functions** is implemented on every processor. SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512 secure hashing is directly available to application programs.
- Feature code 3863, **CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement** – enables clear key DES and TDES instructions on all CPs. AES 128-bit, AES 192-bit and AES 256-bit support is also available.

- Feature code 0863, **Crypto Express2 Feature** – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The z10 EC and z10 BC can support a maximum of 8 features. Each feature code has two coprocessors/accelerators.
- Feature code 0864, **Crypto Express3 Feature** – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The z10 EC and z10 BC can support a maximum of 8 features. Each feature code has two coprocessors/accelerators.

IBM System z9 Business Class (z9 BC)

The IBM System z9 BC provides constraint relief and addresses various customer demands. It has several cryptographic features.

- **CP Assist for Cryptographic Functions** is implemented on every processor. SHA-1, SHA-224 and SHA-256 secure hashing is directly available to application programs.
- Feature code 3863, **CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement** – enables clear key DES and TDES instructions on all CPs. In addition, ICSF supports hardware implementation of AES 128-bit keys and software implementation of AES 192-bit and AES 256-bit key lengths.
- Feature code 0863, **Crypto Express2 Feature** – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM System z9 BC can support a maximum of 8 features. Each feature code has two coprocessors/accelerators.

IBM System z9 Enterprise Class (z9 EC)

The IBM System z9 EC provides constraint relief and addresses various customer demands. It has several cryptographic features.

- **CP Assist for Cryptographic Functions** is implemented on every processor. SHA-1, SHA-224 and SHA-256 secure hashing is directly available to application programs.
- Feature code 3863, **CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement** – enables clear key DES and TDES instructions on all CPs. In addition, ICSF supports hardware implementation of AES 128-bit keys and software implementation of AES 192-bit and AES 256-bit key lengths.
- Feature code 0863, **Crypto Express2 Feature** – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM System z9 EC can support a maximum of 8 features. Each feature code has two coprocessors/accelerators.

IBM @server zSeries 990 (z990)

The **IBM @server zSeries 990** provides constraint relief and addresses various customer demands. It has several cryptographic features.

- **CP Assist for Cryptographic Functions** is implemented on every processor. SHA-1 secure hashing is directly available to application programs.
- Feature code 3863, **CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement** – enables clear key DES and TDES instructions on all CPs. In addition, ICSF supports software implementation of AES.
- Feature code 0862, **PCI Cryptographic Accelerator** – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM @server zSeries 990 can support a maximum of 12 PCI Cryptographic Accelerators. Each feature code has two coprocessors.

- Feature code 0868, **PCI X Cryptographic Coprocessor** – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM @server zSeries 990 can support a maximum of 4 PCIXCCs. Each feature code has one coprocessor.
- Feature code 0863, **Crypto Express2 Coprocessor** – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM @server zSeries 990 can support a maximum of 16 CEX2Cs. Each feature code has two coprocessors.

Note: You can have a maximum of 6 PCICA features (12 cards), 4 PCIXCC features (4 cards) and 16 CEX2Cs (8 features), with maximum number of 8 installed features.

IBM @server zSeries 890 (z890)

The **IBM @server zSeries 890** provides constraint relief and addresses various customer demands. It has several cryptographic features.

- **CP Assist for Cryptographic Functions** are implemented on every processor. SHA-1 secure hashing is directly available to application programs.
- Feature code 3863, **CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement** – enables clear key DES and TDES instructions on all CPs. In addition, ICSF supports software implementation of AES.
- Feature code 0862, **PCI Cryptographic Accelerator** – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM @server zSeries 890 can support a maximum of 12 PCI Cryptographic Accelerators. Each feature code has two coprocessors.
- Feature code 0868, **PCI X Cryptographic Coprocessor** – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM @server zSeries 890 can support a maximum of 4 PCIXCCs. Each feature code has one coprocessor.
- Feature code 0863, **Crypto Express2 Coprocessor** – optional, and only available if you have feature 3863, CPACF DES/TDES Enablement installed. The IBM @server zSeries 890 can support a maximum of 16 CEX2Cs. Each feature code has two coprocessors.

Note: You can have a maximum of 6 PCICA features (12 cards), 4 PCIXCC features (4 cards) and 16 CEX2Cs (8 features), with maximum number of 8 installed features.

IBM @server zSeries 900 (z900) — Feature Code 800

You can enable these features on this server:

- **Cryptographic Coprocessor Feature** – one or two cryptographic coprocessors protected by tamper-detection circuitry and a cryptographic battery unit.
- Feature code 0861, **PCI Cryptographic Coprocessor** (PCICC) – based on the 4758 model 2 standard PCI-bus card package. You must have at least one Cryptographic Coprocessor Feature on your system with a PCICC. Note that each feature has two coprocessors.
- Feature code 0862, **PCI Cryptographic Accelerator** (PCICA). You must have at least one Cryptographic Coprocessor Feature on your system with a PCICA. Note that each feature has two coprocessors.

Note: The IBM @server zSeries 900 can support a combination of PCI Cryptographic Coprocessors (maximum of 16) or PCI Cryptographic Accelerators (maximum of 12), but the total must not exceed 16.

IBM @server zSeries 800 (z800) — Feature Code 800

These features are available on this server:

- **Cryptographic Coprocessor Feature** (CCF) – one or two cryptographic coprocessors protected by tamper-detection circuitry and a cryptographic battery unit.
- Feature code 0861, **PCI Cryptographic Coprocessor** (PCICC) – based on the 4758 model 2 standard PCI-bus card package. You must have at least one Cryptographic Coprocessor Feature on your system with a PCICC. Note that each feature has two coprocessors.
- Feature code 0862, **PCI Cryptographic Accelerator** (PCICA). You must have at least one Cryptographic Coprocessor Feature on your system with a PCICA. Note that each feature code has two coprocessors.

Note: The IBM @server zSeries 800 can support a combination of PCI Cryptographic Coprocessors (maximum of 16) or PCI Cryptographic Accelerators (maximum of 12), but the total must not exceed 16.

z/OS ICSF FMIDs

These tables explain the relationships of z/OS releases, ICSF FMIDs and servers.

Table 1. z/OS ICSF FMIDs

z/OS and z/OS.e	ICSF FMID[1]	Web deliverable name
V1.11	HCR7751	Cryptographic Support for z/OS V1R8-R10 & z/OS.e V1R8
	HCR7770	Cryptographic Support for z/OS V1R9-R11
	HCR7780	Cryptographic Support for z/OS V1R10-R12
	HCR7790	Cryptographic Support for z/OS V1R11-R13
V1R12	HCR7780	Cryptographic Support for z/OS V1R10-R12
	HCR7790	Cryptographic Support for z/OS V1R11-R13
V1R13	HCR7790	Cryptographic Support for z/OS V1R11-R13

Notes:

1. PTF information can be found in the PSP bucket '2094DEVICE'.

Refer to this chart to determine what release is associated with each ICSF FMID and what server it will run on.

Table 2. FMID and Hardware

ICSF FMID	Applicable z/OS Releases	Servers where FMID will run
HCR7751 (Base of z/OS 1.11)	1.9, 1.10, and 1.11	z800, z900, z890, z990, z9 EC, z9 BC, z10 EC and z10 BC
HCR7770 (Base of z/OS 1.12)	1.9, 1.10, and 1.11	z800, z900, z890, z990, z9 EC, z9 BC, z10 EC and z10 BC
HCR7780 (Base of z/OS 1.13)	1.10, 1.11, and 1.12	z800, z900, z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, and z196.
HCR7790	1.11, 1.12, and 1.13	z800, z900, z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, and z196.

ICSF Features

ICSF protects data from unauthorized disclosure or modification. It protects data that is stored within a system, stored in a file on magnetic tape off a system, and sent between systems. It can also be used to authenticate identities of senders and receivers and to ensure the integrity of messages transmitted over a network. It uses cryptography to accomplish these functions.

Cryptography enciphers data, using an algorithm and a cryptographic key, so the data is in an unintelligible form. Deciphering data involves reproducing the intelligible data from the unintelligible data. To encipher and decipher data, ICSF uses either the U.S. National Institute of Science and Technology Data Encryption Standard (DES) algorithm, Advanced Encryption Standard (AES), Elliptic Curve Cryptography (ECC), or the Commercial Data Masking Facility (CDMF).

Restrictions:

- The CDMF defines a scrambling technique for data confidentiality. It is a weakened form of DES and is only supported on IBM @server zSeries 900 servers.
- ECC is supported only on the z196 with a CEX3C.

ICSF supports several Public Key Algorithms (PKA), which do not require exchanging a secret key. You can use these algorithms to exchange AES, DES, or CDMF secret keys securely and to compute digital signatures for authenticating messages and users. For digital signatures, you use a pair of keys: a private (secret) key to sign a message and a corresponding public key to verify the signature. ICSF supports the RSA, ECC, and DSS algorithms. (Refer to the Federal Information Processing Standard (FIPS) Publication 186 for DSS standards.)

Restrictions:

- DSS is only supported on IBM @server zSeries 900 servers.
- ECC is supported only on the z196 with a CEX3C.

A key can be any combination of hexadecimal characters. A key determines how ICSF uses the algorithm to uniquely encipher data.

You can call an ICSF callable service from an application program to perform a cryptographic function. ICSF uses keys in cryptographic functions to:

- Protect data
- Protect other keys
- Verify that messages were not altered between sender and receiver
- Generate, protect, and verify personal identification numbers (PINs)
- Distribute AES, DES and CDMF keys
- Generate and verify digital signatures

You use ICSF callable services and programs to generate, maintain, and manage keys that are used in the cryptographic functions. A unique key performs each type of cryptographic function on ICSF. AES keys, except the AES master key, are enciphered under another key. The AES master key, which is physically secure, enciphers each AES key that is used on the system. All DES keys, except the DES master key, are enciphered under another key. The DES master key, which is physically secure, enciphers each DES key that is used on the system. AES keys are enciphered under an AES master key (AES-MK). The AES master key is 256-bits long. It is only available on the Crypto Express2 Coprocessor or Crypto Express3 Coprocessor with the Nov. 2008 or later licensed internal code (LIC). DES

keys are enciphered under the DES master key or a DES key-encrypting key. The DES master key (DES-MK or SYM-MK) is a double-length key that is used only to encrypt other DES keys. The AES and DES master keys are physically secure.

On CCF systems, the DES-MK must be the same as the DES master key on the Cryptographic Coprocessor Feature. On systems with PCI X Cryptographic Coprocessors, Crypto Express2 Coprocessors, or Crypto Express3 Coprocessor, the DES-MK verification pattern must match the hash pattern of the CKDS.

On Crypto Express2 Coprocessors and Crypto Express3 Coprocessors, the AES master key verification pattern must match the AES master key verification pattern stored in the CKDS. The AES master key (AES-MK) is a 32 byte key that is used only to encrypt AES keys.

There are two public key master keys available — the RSA master key (RSA-MK) and the ECC master key (ECC-MK). ICSF handles each master key independently. Either, both, or neither of the master keys can be set.

- RSA master keys protect RSA private keys. There are two RSA master keys on the Cryptographic Coprocessor Feature. One RSA master key, the signature master key (SMK), protects private keys that are intended for creating digital signatures. The other RSA master key, the key management master key (KMMK), protects private keys that are used in DES key distribution. Private keys that are protected by the KMMK can also be used to generate digital signatures.

  The RSA master key (RSA-MK) on the PCICC, PCIXCC, CEX2C, or CEX3C is a triple-length key used to encipher and decipher RSA keys. In order for the PCI Cryptographic Coprocessor to function, the value of the RSA-MK must have the same value as the SMK on the Cryptographic Coprocessor Feature. If the PCICC master key values are different, then the PCICC will not be made active. On systems with a PCIXCC, CEX2C, or CEX3C, the RSA-MK hash pattern must match the hash pattern of the PKDS.
- ECC master keys protect ECC keys. The ECC master key is a 256-bit AES key used to protect ECC private keys. ECC keys are supported only on the z196 with a CEX3C coprocessor. Although a CEX3C card is not necessary to load a PKDS with ECC keys, those keys will not be usable without a CEX3C.

The Cryptographic Key Data Set (CKDS)

Keys that are protected under the DES or AES master key are stored in a VSAM data set that is called the cryptographic key data set (CKDS). ICSF provides sample CKDS allocation jobs (member CSFCKDS and CSFCKD2) in SYS1.SAMPLIB. The CKDS contains individual entries for each key that is added to it. You can store all types of keys (except master keys and PKA keys) in the CKDS. Each record in the data set contains the key value encrypted under the master key and other information about the key. ICSF maintains two copies of the CKDS: a disk copy and an in-storage copy.

Notes:

1. There are two formats of the CKDS: a fixed length record (supported by all releases of ICSF) and a new, variable length record (supported by HCR7780 and later releases). The variable length record format is only required if HMAC keys are to be stored in the CKDS. The variable length record format can be used to store all existing symmetric keys and the new HMAC keys.
2. When a CKDS record is written which contains a key token with a control vector that is not supported by the Cryptographic Coprocessor Feature, a key type of CV will be placed into the CKDS record. During CKDS reencipher processing,

for any key containing a control vector which is not supported by the Cryptographic Coprocessor Feature, a key token change request will be sent to the PCI Cryptographic Coprocessor to reencipher the key. In a sysplex with a shared CKDS, the CKDS reencipher process must be invoked on a system which has a PCI Cryptographic Coprocessor installed.

Callable services use the in-storage copy of the CKDS to perform CKDS functions. For information on managing and sharing the CKDS in a sysplex environment, see *z/OS Cryptographic Services ICSF Administrator's Guide*, SA22-7521. The key generator utility program (KGUP) updates the disk copy rather than the in-storage copy. Therefore, cryptographic functions do not have to stop while KGUP updates the CKDS. The ICSF administrator can use the ICSF panels or a utility program to refresh the in-storage CKDS with the updated disk copy of the CKDS. Applications can also use the dynamic CKDS update callable services to update both the in-storage and DASD copies of the CKDS with no interruption of cryptographic function.

To add operational keys to the CKDS for z900, you can:
- Use KGUP to generate or enter keys
- Use the dynamic CKDS update callable services to create and write keys directly to the CKDS
- Use the Trusted Key Entry (TKE) workstation to load operational PIN and TRANSPORT keys. TKE is not part of the base product. It is an optional feature.

To add operational keys to the CKDS for the z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, and z196 servers, you can:
- Use KGUP to generate or enter keys or to load keys from the cryptographic coprocessor's key part registers
- Use the dynamic CKDS update callable services to create and write keys directly to the CKDS
- Use the Trusted Key Entry workstation to load operational AES or DES keys. DES keys can be loaded with TKE Version 4.1 or higher. AES keys can be loaded with TKE Version 5.3 or higher. TKE is not part of the base product. It is an optional feature.

The Public Key Data Set (PKDS)

RSA, ECC, and DSS public and private keys can be stored in a VSAM data set that is called the public key data set (PKDS). ICSF maintains the PKDS as an external data set. ICSF provides a sample PKDS allocation job (member CSFPKDS) in SYS1.SAMPLIB. ICSF maintains two copies of the PKDS: a disk copy and an in-storage copy.

You can store public key tokens or both external and internal private key tokens. Applications can use the dynamic PKDS update callable services to create, write, read, and delete PKDS records.

The PKDS must be initialized using the ICSF Master Key Management panels.

Support to reencipher and refresh the PKDS is available by using the Master Key Management Panels or the CSFPUTIL utility to reencipher the PKDS and to refresh the reenciphered PKDS. CSFPUTIL is a utility that performs the same reencipher and refresh functions available using the Master Key Management panels. Other systems with lower levels of ICSF which are sharing the PKDS would disable the dynamic PKDS access control, change the appropriate master key(s), refresh the reenciphered PKDS and enable the dynamic PKDS access control. For information

on managing and sharing the PKDS in a sysplex environment, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

Notes:

1. ECC support is available in ICSF HCR7780 and later releases. A PKDS with ECC key tokens can be shared with prior levels of ICSF. A reencipher of the PKDS with ECC tokens can only be done on systems that support ECC. If a prior level system attempts to reencipher a PKDS containing ECC tokens, it will fail with a bad token error (12/36112).
2. With ICSF release HCR7750 or later, ICSF expects the PKDS to have the longer LRECL before it will start. You can share the larger PKDS with down-level systems by installing the toleration APAR OA21807. Even with toleration APAR OA21807 installed, however, be aware that reencipherment of a larger PKDS must always be performed on an HCR7750 or later system.

The Token Data Set (TKDS)

PKCS #11 tokens and objects are stored in a VSAM data set called the token data set (TKDS). ICSF provides a sample TKDS allocation job (member CSFTKDS) in SYS1.SAMPLIB. The TKDS contains individual entries for each token and object that is added to it. ICSF maintains two copies of the TKDS: a disk copy and an in-storage copy. Only token objects are stored in the TKDS, session objects are stored in a data space.

The TKDS must be a key-sequenced data set with spanned variable length records and must be allocated on a permanently resident volume.

Additional Background Information

These topics provide some additional background information about using ICSF with other products, such as the Programmed Cryptographic Facility (PCF).

Running PCF applications on z/OS ICSF

If your installation uses PCF, you can run PCF applications on ICSF. You can use an installation option to specify whether a PCF application runs on ICSF. If you are migrating from PCF, ICSF provides a conversion program that converts a PCF CKDS to ICSF format.

You can use your own installation services and exits to customize ICSF. You can write, define, and call your own installation-defined callable service. You can also write and define exits that ICSF calls during the processing of:

- ICSF mainline
- A callable service
- The PCF CKDS conversion program
- The key generator utility program
- CKDS access

For example, most callable services in ICSF call an exit before and after processing. Such an exit can alter return codes in a service.

ICSF System SVC 143

SVC 143 (0A8F) is an ICSF system SVC that is used by CUSP and PCF macros (GENKEY, RETKEY, CIPHER, and EMK) for SVC entry into ICSF. The SVC allows

you to run a CUSP or PCF application on ICSF. See “Running PCF and z/OS ICSF on the same system” on page 173 for more information about running CUSP and PCF applications on ICSF.

SVC 143 is a type 4 SVC and does not get a lock. The General Trace Facility data is:

r141 and R0 No applicable data.

R1 Address of the parameter list. The macro that is called determines the parameter list.

Running 4753-HSP applications on ICSF

If your installation uses the IBM Network Security Processor Support Program products, you may be able to run 4753-HSP applications on ICSF without change. There are some restrictions when running both ICSF and 4753-HSP in the same z/OS (MVS) environment:

- Although both ICSF and 4753-HSP can run PCF compatibility mode, only one system can provide this service at any time.
- Because both ICSF and 4753-HSP support the Common Cryptographic Architecture (CCA) Application Programming Interface, applications need to be linked with the callable service stubs that each product provides to access the intended service.
- Internal key tokens are not interchangeable between the two products.

For more information on running ICSF and 4753-HSP in the same operating system environment, refer to “Running 4753-HSP and ICSF on the same z/OS system” on page 191.

Using RMF and SMF to monitor z/OS ICSF events

You can run ICSF in different configurations and use installation options to affect ICSF performance. While ICSF is running, you can use the Resource Management Facilities (RMF) and System Management Facilities (SMF) to monitor certain events. For example, ICSF records information in the SMF data set when ICSF changes the status of a cryptographic processor or when you enter or change the master key. ICSF also sends information and diagnostic messages to data sets and consoles.

With the availability of cryptographic hardware on an LPAR basis, RMF provides performance monitoring in the Postprocessor Crypto Hardware Activity report. This report is based on SMF record type 70, subtype 2. The Monitor I gathering options on the REPORTS control statement are CRYPTO and NOCRYPTO. Specify CRYPTO to measure cryptographic hardware activity and NOCRYPTO to suppress the gathering. In addition, overview criteria is shown for the Postprocessor in the Postprocessor Workload Activity Report - Goal Mode (WLMGL) report. Refer to *z/OS RMF Programmer's Guide*, SC33-7994, *z/OS RMF User's Guide*, SC33-7990, and *z/OS RMF Report Analysis*, SC33-7991 for additional information.

ICSF also supports enabling RMF to provide performance measurements on ICSF services (Encipher, Decipher, MAC Generate, MAC Verify, One Way Hash, PIN Translate, and PIN Verify). These measurements are of the Direct Access Crypto CCF instructions and the PCIXCCs, CEX2Cs, or CEX3Cs.

For diagnosis monitoring, use Interactive Problem Control System (IPCS) to access the trace buffer and to format control blocks.

Controlling access to ICSF

For security, you should control access to ICSF resources and services. Use a security product like the Security Server (RACF) to protect cryptographic programs, keys, and services. You should also change the value of the DES, AES and PKA master keys periodically.

Steps prior to starting installation

You use either ServerPac or CBPDO to install ICSF as part of the z/OS installation process.

When beginning installation:

1. Refer to *z/OS Planning for Installation* for installation planning information.
2. Check with your IBM center or search the IBM problem database to find any pertinent Preventative Service Planning (PSP). There may also be HOLDDATA and PSP information for ICSF on the tape.
3. Make sure that you have all needed programs and their corequisites:
   - If you use the Security Sever (RACF) and want access control and auditing services for ICSF, you need the Security Server (RACF), an optional feature of z/OS.
   - If you are a Resource Measurement Facility (RMF) user, you need the Resource Measurement Facility option available with z/OS.
4. Collect all required information. The Program Directory lists publications useful during installation.
5. Confirm you have adequate DASD storage and create SMP/E DDDEF entries for each data set. See the Program Directory for details.

Chapter 2. Installation, Initialization, and Customization

For this topic, you need to understand these terms:

installation options
: You create an installation options data set that specifies these options. They become active when you start ICSF, customizing how ICSF runs on your system.

startup procedure
: You create an ICSF startup procedure. Along with other information, this specifies the name of the installation options data set.

SYS1.SAMPLIB
: Contains samples, including an installation options data set, a CKDS allocation job, a PKDS allocation job, a startup procedure, a CICS Wait List data set, and sample JCL for SMP/E Delivery to load keys by using a pass phrase. You can update this code as necessary and generally store the updated code in SYS1.PARMLIB and SYS1.PROCLIB.

SYS1.PARMLIB
: Generally contains the installation options data set. The installation options data set can alternately be a member of a partitioned or sequential data set.

SYS1.PROCLIB
: Contains the startup procedure.

Steps for installation and initialization

Refer to the *z/OS Program Directory* for installation instructions. Several of the installation steps in the *z/OS Program Directory* refer you to this publication for details. This publication explains these installation steps.

Note: Because it is possible for ICSF control blocks like the DACC and CCVT to persist in storage across an ICSF restart, an IPL is required when installing a new release of ICSF.

1. Customize SYS1.PARMLIB. “Steps to customize SYS1.PARMLIB” on page 14 describes this task.
2. Create the Cryptographic Key Data Set (CKDS). “Steps to create the CKDS” on page 17 describes this. Create the Public Key Data Set (PKDS). “Steps to create the PKDS” on page 20 describes this task.
3. If PKCS #11 support is desired, create the TKDS. “Steps to create the TKDS” on page 24 describes this task.
4. Create the installation options data set. “Steps to create the Installation Options Data Set” on page 26 describes this task.
5. Create the startup procedure. “Steps to create the ICSF Startup Procedure” on page 28 describes this task.
6. Provide access to the ICSF panels. “Steps to provide access to the ICSF panels” on page 30 describes this task.

   Note: You only need to perform the first six steps once. If you stop ICSF and want to perform a subsequent SMP/E electronic delivery (this is optional), you need to start at Step 7 (Start ICSF for the first time).

7. Start ICSF for the first time. See “Steps to start ICSF for the first time” on page 31. Once ICSF has been started, Master Keys can be entered.

   For additional information on ICSF first time startup, refer to “Checklist for First-Time Startup of ICSF” on page 305. See *z/OS Cryptographic Services ICSF Administrator's Guide* for directions on entering Master Keys.
8. Enter Master Keys.
9. Run the JCL to set the SMP/E pass phrase for SMP/E electronic delivery (optional). “MK Initialization for SMP/E - CCF Systems Only” on page 36 describes this.

   Only complete this step if ICSF is needed for SMP/E. Do not complete this step for other applications.

Other topics in this publication and *z/OS Cryptographic Services ICSF Administrator's Guide* provide additional installation information.

For information on installing the CICS-ICSF Attachment Facility, refer to Appendix C, “CICS-ICSF Attachment Facility,” on page 299.

Steps to customize SYS1.PARMLIB

The installation options data set you will create is generally stored in SYS1.PARMLIB. If your administrator does not have access to SYS1.PARMLIB, you need to use another data set instead.

Update the data set you are using as follows:

1. Add CEE.SCEERUN and CSF.SCSFMOD0 to the LNKLST concatenation. This adds the ICSF library to the z/OS library search. This is an example of an ICSF entry to the LNKLST concatenation.

   ```
   CSF.SCSFMOD0
   ```
2. APF authorize CSF.SCSFMOD0, if LNKAUTH=APFTAB. This is an example of an ICSF entry for APF authorization.

   ```
   APF ADD DSNAME(CSF.SCSFMOD0)  VOLUME(******)
   ```
3. In the IKJTSOxx parameter, add CSFDAUTH and CSFDPKDS as a value in the AUTHPGM parameter list and in the AUTHTSF parameter list. This is an example of an ICSF entry in the IKJTSOxx member.

   ```
   AUTHPGM NAMES(        /* AUTHORIZED PROGRAMS            */  +
    ....
    ....
   CSFDAUTH              /* ICSF                           */  +
   CSFDPKDS              /* ICSF                           */  +

    ....

   AUTHTSF NAMES(        /* PROGRAMS TO BE AUTHORIZED WHEN  */  +
                         /* WHEN CALLED THROUGH THE TSO    */  +
                         /* SERVICE FACILITY               */  +
    ....
    ....
   CSFDAUTH              /* ICSF                           */  +
   CSFDPKDS              /* ICSF                           */
   ```
4. If your application programmers intend to use PKCS #11 token key objects for AES Galois/Counter Mode (GCM) encryption or GMAC generation, and have ICSF generate the initialization vectors, then you need to set ECVTSPLX or CVTSNAME to a unique value.

   This needs to be done, because, for AES GCM encryption or GMAC generation, the security of the algorithm is dependent on never repeating a key, initialization vector combination for two or more distinct sets of data. In PKCS #11,

applications can request that ICSF generate a new (unique) initialization vector each time AES GCM or GMAC is initiated. In fact, this is the only permitted way to perform AES GCM or GMAC when PKCS #11 is operating in FIPS mode. When ICSF generates initialization vectors, it uses the ECVTSPLX (sysplex mode) or CVTSNAME (non-sysplex mode) field as the cryptographic module name. The name ensures uniqueness if such keys are distributed to multiple systems, but only if each system is set with a unique name.

When setting ECVTSPLX or CVTSNAME to unique values, be aware that ICSF uses only the first (left most) 4 characters of these fields. For this reason, these 4 characters must be set to uniquely identify the system.

For example, suppose AES key value 123 is created on the current single-image system (known as System A) and is distributed to another system residing in a Sysplex (known as Sysplex B). Both systems will be performing GCM encryption where ICSF generates the initialization vectors. To ensure that unique initialization vectors are generated, set CVTSNAME=SYSA on System A and ECVTSPLX=PLXB on Sysplex B.

CVTSNAME is normally set from the SYSNAME=value statement in the IEASYSxx member of "SYS1.PARMLIB". For more information, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.

ECVTSPLX is normally set from the COUPLE SYSPLEX(value) in the COUPLExx member of "SYS1.PARMLIB". For more information, see *z/OS MVS Setting Up a Sysplex*, SA22-7625.

Notes:

1. If you will be using TKE V3.0 or higher on this host, you should also add CSFTTKE as a value in the AUTHCMD parameter list.
2. If you will only be using ICSF for SMP/E electronic delivery, this step does not need to be performed. TKE is not needed for SMP/E electronic delivery.
3. To change the active IKJTSOxx member of SYS.PARMLIB without an IPL, use the PARMLIB UPDATE command.

z/OS MVS Initialization and Tuning Guide and *z/OS MVS Initialization and Tuning Reference* provide more information.

Creating the CKDS

Installations need to understand and plan for the system resources required for managing the CKDS copy in virtual storage, particularly when the installation is deploying a very large CKDS. Refer to “ICSF System Resource Planning for the CKDS” for guidelines. Once you understand these guidelines, refer to “Steps to create the CKDS” on page 17 for step-by-step instructions.

ICSF System Resource Planning for the CKDS

Like the PKDS and TKDS, ICSF manages a mirror copy of the CKDS data set in protected, private virtual storage to optimize cryptographic workload access to symmetric keys in the normal course of workload operation. This copy is kept current as keys are dynamically added to, and removed from, the active CKDS key store. Like any set of control information maintained in virtual storage, the in-storage CKDS copy must be accommodated with sufficient system central storage and auxiliary paging space resources.

Installations need to understand and plan for the system resources required for managing the CKDS copy in virtual storage, particularly when the installation is deploying a very large CKDS. Note that “very large” is a relative assessment

depending upon the installation, and could be expressed, for example, in terms of tens or hundreds of thousands of symmetric keys in the CKDS, or perhaps even millions of keys.

An in-storage copy of a CKDS that is not experiencing significant dynamic key creation or deletion activity consumes a stable amount of virtual storage, and therefore a stable amount of system backing resource. Certain occasional but unavoidable ICSF functions such as CKDS refresh do, however, generate a significant spike in the amount of utilized virtual storage, and therefore a greater temporary demand for system resources backing that virtual storage.

Given these circumstances, it's important to calculate and plan for the system central storage and auxiliary paging space required to support an active in-storage copy. For a CKDS shared across a sysplex environment, every active ICSF in the sysplex will have an equivalent resource requirement.

Each symmetric key in the CKDS is managed with one VSAM record. Installations need to plan for the appropriate amount of combined central storage and auxiliary paging space for each VSAM record, per active ICSF. The following formula is provided to help you calculate the required system virtual storage backing resource for an active in-storage CKDS. In this formula HI-A-RBA is the allocated relative byte address for the data component of a CKDS VSAM data set. The IDCAMS LISTCAT command output for a CKDS VSAM data set can be consulted to determine the HI-A-RBA value for the data component. The `%Free Space` used in this formula represents the percentage of free space in the CKDS VSAM data set. The IDCAMS EXAMINE DATATEST command output can be consulted to determine the percentage of free space.

```
HI-A-RBA x ( ( 100 - %Free Space ) / 100 ) x 6
```

For example, the central storage and auxiliary paging space requirement for a CKDS VSAM data set with a HI-A-RBA of 481,787,904 for its data component entry and 16 percent free space can be calculated as follows.

```
481,787,904 x ( ( 100 - 16 ) / 100 ) x 6 = 2,428,211,036.16 bytes
```

This CKDS VSAM data set will require 2.26 Gigabytes of combined central storage and auxiliary paging space for system backing resource.

As is the case with all virtual storage usage, central storage is the preferred medium to optimize the workload performance, and to avoid system paging overhead. Note that excessive system paging due to any virtual storage usage can cause degradation across the workload and system operation, and an extreme shortage of central storage and auxiliary paging space can lead to a catastrophic system failure.

Note: The output from the formulas above should be added to the outputs calculated from the formulas in "ICSF System Resource Planning for the PKDS" on page 20 and "ICSF System Resource Planning for the TKDS and Session Object Memory Areas" on page 23. This will give you the required system virtual storage backing resource for all of ICSF's KDS data sets. This value represents the required amount of virtual storage for a given instance of ICSF. For a set of KDS data sets shared across a sysplex environment, every active ICSF in the sysplex will have an equivalent resource requirement.

Additional CKDS Performance Considerations: Beginning with the FMID HCR7780, ICSF support of the CKDS key store data set has been enhanced to

facilitate a CKDS that may contain millions of symmetric keys. If an installation is intending to pursue a CKDS of such a large size, then IBM recommends migrating to HCR7780 (or later) first. Prior releases of ICSF were not designed to accommodate a CKDS with millions of keys, and could experience various symptoms of degradation or failure. Note that, in a sysplex environment sharing the CKDS across multiple active ICSF instances, that all such instances should be migrated to the HCR7780 or later release level before scoping the symmetric key material to that magnitude.

IBM also recommends that installations that deploy a CKDS with millions of symmetric keys not enable CKDS MAC authentication, or disable it if it's already enabled. CKDS MAC authentication adds an additional coprocessor request for each VSAM data set read/write operation. There is a significant performance implication for CKDS MAC authentication that would be greatly magnified with such a large CKDS.

Steps to create the CKDS

The CKDS must be a key-sequenced data. There are two formats: a fixed record length of 252 bytes, and a variable record length. Allocate the CKDS on a permanently resident volume.

Attention: Ensure that this volume is not subject to data set migration. If the CKDS is migrated, message CSFM450E is issued and ICSF ends.

For detailed information about calculating space for a VSAM data set and an explanation of keyed-direct update processing and what happens when control area and control interval splits occur, see *z/OS DFSMS Access Method Services for Catalogs*.

1. Determine the amount of primary space you need to allocate for the CKDS.

   This should reflect the total number of entries you expect the data set to contain originally. Besides transport keys, PIN keys, data-encrypting keys, data-translating keys, and MAC keys, the CKDS contains a header record and system keys that ICSF uses for processing.

   CCF Systems Only: To run ICSF requires the header record and four of the system keys. The other system keys, NOCV enablement keys, ANSI enablement keys, and enhanced system keys, are optional.

   Your system needs NOCV enablement keys if it communicates with systems that do not use control vectors, supports the use of DATA keys in the MAC services, or needs to convert a PCF CKDS. Your system needs ANSI enablement keys if you distribute keys according to the ANSI X9.17 protocol. To use the SET callable services on a CDMF system, you need to install both the NOCV keys and ANSI system keys.

   Fixed length record format: Each record is 252 bytes long. Allocate space for all of the installation and system keys you expect to store in the CKDS.

   Variable length record format: The minimum size of record will be 332 bytes. Records containing HMAC keys can be up to 750 bytes long. Records containing DES and AES keys will be 332 bytes long. Allocate space for all of the installation and system keys you expect to store in the CKDS

2. Determine the amount of secondary space to allocate for CKDS.

   This should reflect the total number of entries you expect to add to the data set.

   To access keys, VSAM uses the key label as the VSAM key. This means that VSAM adds keys to the data set in collating sequence. That is, if two keys named A and B are in the data set, A appears earlier in the data set than B. As a result, adding keys to the data set can cause multiple VSAM control interval

splits and control area splits. For example, a split might occur if the data set contains keys A, B, and E and you add C. In this case, C must be placed between B and E. These splits can leave considerable free space in the data set and can affect KGUP performance.

The amount of secondary space you allocate must take into account the number of control interval and control area splits that might occur. If the disk copy of the CKDS uses a significant amount of secondary space, you can copy it into another disk copy that you created with more primary space. You can do this by using the Access Method Services (AMS) REPRO command or the AMS EXPORT/IMPORT commands.

The BUFFERSPACE parameter on the AMS DEFINE CLUSTER command (required by Step 3) lets VSAM optimize space for control area and control interval splits.

3. Create an empty VSAM data set to use as the CKDS. ICSF provides a sample job to define the CKDS in member CSFCKDS of SYS1.SAMPLIB.

   Use the AMS DEFINE CLUSTER command to define the data set and to allocate its space.

   Note: To improve security and reliability of the data that is stored on the CKDS:

   - Use the ERASE and WRITECHECK parameters on the AMS DEFINE CLUSTER command. ERASE overwrites data records with binary zeros when the CKDS cluster is deleted. WRITECHECK provides hardware verification of all data that is written to the data set.
   - Create a Security Server (RACF) data set profile for the CKDS.

   Fixed length record format: Allocate a disk copy of the CKDS by defining a VSAM cluster as in this SYS1.SAMPLIB CSFCKDS member sample:

```
//CSFCKDS JOB = JOB CARD PARAMETERS
//*******************************************************************
//*  Licensed Materials - Property of IBM                           *
//*  5694-A01                                                       *
//*  COPYRIGHT IBM CORP. 2002, 2009                                 *
//*                                                                 *
//* THIS JCL DEFINES A VSAM CKDS TO USE FOR ICSF                    *
//*                                                                 *
//*  CAUTION: This is neither a JCL procedure nor a complete JOB.   *
//*  Before using this JOB step, you will have to make the following *
//*  modifications:                                                 *
//*                                                                 *
//*  1) Add the job parameters to meet your system requirements.    *
//*  2) Be sure to change CSF to the approrpriate HLQ if you choose *
//*     not to use the default.                                     *
//*  3) Change xxxxxx to the volid where you want your CKDS to      *
//*     reside. The CKDS needs to be on a permanently resident      *
//*     volume.                                                     *
//*                                                                 *
//* NOTE: This JCL is specific for creating the CKDS. There are     *
//*       samples for each of the other key data sets.              *
//*         PKDS  - CSFPKDS JCL                                     *
//*         TKDS  - CSFTKDS JCL                                     *
//*                                                                 *
//*******************************************************************
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  DEFINE CLUSTER (NAME(CSF.CSFCKDS) -
                  VOLUMES(XXXXXX) -
                  RECORDS(100 50) -
                  RECORDSIZE(252,252) -
                  KEYS(72 0) -
                  FREESPACE(10,10) -
```

```
                SHAREOPTIONS(2 3)) -
           DATA (NAME(CSF.CSFCKDS.DATA) -
                BUFFERSPACE(100000) -
                ERASE -
                WRITECHECK) -
          INDEX (NAME(CSF.CSFCKDS.INDEX))
/*
```

Variable length record format: Allocate a disk copy of the CKDS by defining a VSAM cluster as in this SYS1.SAMPLIB CSFCKD2 member sample:

```
//CSFCKD2 JOB = JOB CARD PARAMETERS
//*******************************************************************
//* Licensed Materials - Property of IBM                            *
//* 5694-A01                                                        *
//* COPYRIGHT IBM CORP. 2010                                        *
//*                                                                 *
//* THIS JCL DEFINES A VSAM CKDS FOR VARIABLE LENGTH RECORDS        *
//* TO USE FOR ICSF                                                 *
//*                                                                 *
//* CAUTION: This is neither a JCL procedure nor a complete JOB.    *
//* Before using this JOB step, you will have to make the following *
//* modifications:                                                  *
//*                                                                 *
//* 1) Add the job parameters to meet your system requirements.     *
//* 2) Be sure to change CSF to the appropriate HLQ if you choose   *
//* not to use the default.                                         *
//* 3) Change xxxxxx to the volid where you want your CKDS to       *
//* reside. The CKDS needs to be on a permanently resident          *
//* volume.                                                         *
//*                                                                 *
//* NOTE: This JCL is specific for creating the CKDS. There are     *
//* samples for each of the other key data sets.                    *
//* PKDS - CSFPKDS JCL                                              *
//* TKDS - CSFTKDS JCL                                              *
//*                                                                 *
//*******************************************************************
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  DEFINE CLUSTER (NAME(CSF.CSFCKDS)    -
                  VOLUMES(XXXXXX)      -
                  RECORDS(100 50)      -
                  RECORDSIZE(332,1024) -
                  KEYS(72 0)           -
                  FREESPACE(10,10)     -
                  SHAREOPTIONS(2,3))   -
           DATA (NAME(CSF.CSFCKDS.DATA) -
                  BUFFERSPACE(100000)    -
                  ERASE                  -
                  WRITECHECK)            -
          INDEX (NAME(CSF.CSFCKDS.INDEX))
/*
```

You can change and use the Job Control Language according to the needs of your installation. Please note that the JCL to define the CKDS differs from the JCL that defines the PKDS (RECORDSIZE and CISZ parameters). For more information about allocating a VSAM data set, see *z/OS DFSMS Access Method Services for Catalogs*.

Creating the PKDS

Installations need to understand and plan for the system resources required for managing the PKDS copy in virtual storage, particularly when the installation is deploying a very large PKDS. Refer to “ICSF System Resource Planning for the PKDS” on page 20 for guidelines. Once you understand these guidelines, refer to “Steps to create the PKDS” on page 20 for step-by-step instructions.

ICSF System Resource Planning for the PKDS

Like the CKDS and TKDS, ICSF manages a mirror copy of the PKDS data set in protected, private virtual storage to optimize cryptographic workload access to asymmetric keys. Again, similar to the CKDS, the in-storage PKDS copy must be accommodated with sufficient system central storage and auxiliary paging space resources. The same formula used in the system resource planning section for the CKDS can be used to estimate the virtual storage requirement for an existing, stable PKDS (one that is not experiencing significant dynamic asymmetric key creation or deletion activity).

```
HI-A-RBA x ( ( 100 - %Free Space ) / 100 ) x 6
```

As described in “ICSF System Resource Planning for the CKDS” on page 15, the output from running the IDCAMS LISTCAT and EXAMINE DATATEST commands against a PKDS VSAM data set can be consulted to determine the data set's data component HI-A-RBA and the percentage of free space in the data set.

Note: The output from the formula above should be added to the outputs calculated from the formulas in “ICSF System Resource Planning for the CKDS” on page 15 and “ICSF System Resource Planning for the TKDS and Session Object Memory Areas” on page 23. This will give you the required system virtual storage backing resource for all of ICSF’s KDS data sets. This value represents the required amount of virtual storage for a given instance of ICSF. For a set of KDS data sets shared across a sysplex environment, every active ICSF in the sysplex will have an equivalent resource requirement.

Steps to create the PKDS

The PKDS must be allocated and the PKDS data set name must be specified on the PKDSN parameter of the options data set when you first start ICSF. ICSF support for the PCICC, PCIXCC, CEX2C, or CEX3C requires a PKDS. Even if not available at first time start up, a PCICC, PCIXCC, CEX2C, or CEX3C can be dynamically configured online. Since ICSF can not tell if a PCICC, PCIXCC, CEX2C, or CEX3C will be added, it requires the PKDS to be available at start up.

The PKDS must be a key-sequenced data set with variable length records. Allocate the PKDS on a permanently resident volume.

1. Determine the amount of primary space you need to allocate for the PKDS.

   This should reflect the total number of entries you expect the data set to contain originally. The PKDS will contain both public and private PKA keys. Each record has a maximum size of 2.8 KB. The average record length for a private key is 1 KB, and for a public key is 0.5 KB. Allocate space for a minimum of two private keys, one for digital signatures, and another for encipherment. In addition, allocate enough space for the number of public keys you expect to store in the PKDS. The number of public keys varies from system to system. Generally, only those keys that are received from other users or systems are stored in the PKDS. The public keys are used to send messages to the owners of the public keys.
2. Determine the amount of secondary space to allocate for the PKDS.

   This should reflect the total number of entries you expect to add to the data set. For detailed information about calculating space for a VSAM data set, see *z/OS DFSMS Access Method Services for Catalogs*.

   To access keys, VSAM uses the key label as the VSAM key. This means that VSAM adds keys to the data set in collating sequence. That is, if two keys named A and B are in the data set, A appears earlier in the data set than B. As

a result, adding keys to the data set can cause multiple VSAM control interval splits and control area splits. For example, a split might occur if the data set contains keys A, B, and E and you add C. In this case, C must be placed between B and E.

The amount of secondary space you allocate must take into account the number of control interval and control area splits that might occur. If the PKDS uses a significant amount of secondary space, you can copy it into another disk copy that you created with more primary space. You can do this by using the Access Method Services (AMS) REPRO command or the AMS EXPORT/IMPORT commands.

The BUFFERSPACE parameter on the AMS DEFINE CLUSTER command (required by Step 3) lets VSAM optimize space for control area and control interval splits. For a detailed explanation of keyed-direct update processing and what happens when control area and control interval splits occur, see *z/OS DFSMS Access Method Services for Catalogs*.

3. Create an empty VSAM data set to use as the PKDS. Use the AMS DEFINE CLUSTER command to define the data set and to allocate its space. ICSF provides a sample job to define the PKDS in member CSFPKDS of SYS1.SAMPLIB.

   Note: To improve security and reliability of the data that is stored on the PKDS:
   - Use the ERASE and WRITECHECK parameters on the AMS DEFINE CLUSTER command. ERASE overwrites data records with binary zeros when the PKDS cluster is deleted. WRITECHECK provides hardware verification of all data that is written to the data set.
   - Create a Security Server (RACF) data set profile for the PKDS.
   - The CISZ(8192) coded in this sample in the DATA section is a hardcoded requirement.

4. Allocate a disk copy of the PKDS by defining a VSAM cluster as in this SYS1.SAMPLIB CSFPKDS member sample:

```
//CSFPKDS JOB = JOB CARD PARAMETERS
//*******************************************************************
//*  Licensed Materials - Property of IBM                           *
//*  5694-A01                                                       *
//*  Copyright IBM Corp. 2002, 2009                                 *
//*                                                                 *
//* THIS JCL DEFINES A VSAM PKDS TO USE FOR ICSF                    *
//*                                                                 *
//*  CAUTION: This is neither a JCL procedure nor a complete JOB.   *
//*  Before using this JOB step, you will have to make the following *
//*  modifications:                                                 *
//*                                                                 *
//*  1) Add the job parameters to meet your system requirements.    *
//*  2) Be sure to change CSF to the appropriate HLQ if you choose  *
//*     not to use the default.                                     *
//*  3) Change xxxxxx to the volid where you want your PKDS to      *
//*     reside. The PKDS needs to be on a permanently resident      *
//*     volume.                                                     *
//*                                                                 *
//* NOTE: This JCL is specific for creating the PKDS. There are     *
//*       samples for each of the other key data sets.              *
//*         CKDS  - CSFCKDS JCL                                     *
//*         TKDS  - CSFTKDS JCL                                     *
//*                                                                 *
//*******************************************************************
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  DEFINE CLUSTER (NAME(CSF.CSFPKDS) -
```

```
                VOLUMES(XXXXXX) -
                RECORDS(100,50) -
                RECORDSIZE(350,3800) -
                KEYS(72 0) -
                FREESPACE(0,0) -
                SHAREOPTIONS(2,3)) -
         DATA (NAME(CSF.CSFPKDS.DATA) -
                BUFFERSPACE(100000) -
                ERASE -
                CISZ(8192) -
                WRITECHECK) -
       INDEX (NAME(CSF.CSFPKDS.INDEX))
/*
```

You can change and use the Job Control Language according to the needs of your installation. Please note that the JCL to define the PKDS differs from the JCL that defines the CKDS (RECORDSIZE and CISZ parameters). For more information about allocating a VSAM data set, see *z/OS DFSMS Access Method Services for Catalogs*.

Migrating to a larger PKDS: In ICSF HCR7750, the LRECL for the PKDS increased. This change allows 4096-bit RSA public and private keys to be stored in the PKDS. With ICSF release HCR7750 or later, ICSF expects the PKDS to have the longer LRECL before it will start.

If you currently have a PKDS that ICSF is using and are planning to move to

- the ICSF HCR7750 or later web deliverable
- z/OS V1.10 or later

then you need to perform the following steps before starting the new version of ICSF.

The steps take you through the tasks that must be performed to make an exact copy of the old PKDS contents. The ICSF PKA services are not available during the time the copy is made to ensure the PKDS contents are not changing. You need to schedule the change for a period of time when the applications using the PKDS keys are also not available.

The steps to migrate are:

1. If the PKDS is shared with down-level systems, install the toleration APAR on those systems to allow continued sharing of the PKDS. The toleration APAR number is OA21807.

   Note: Even with toleration APAR OA21807 installed, however, be aware that reencipherment of a larger PKDS must always be performed on an HCR7750 or later system.
2. Create the larger PKDS - use the JCL in SYS1.SAMPLIB(CSFPKDS) from the HCR7750 or later system. If the PKDS will be shared, place the VSAM data set where it can be shared.
3. Suspend activity with the PKDS:
   - On HCR7751 and lower releases, disable the PKDS READ, PKDS WRITE, PKDS CREATE, and PKDS DELETE access from the ADMINCNTL option. This will prevent any updates from being made while the migration action is performed. It will affect applications that use the PKDS services.
   - On HCR7770, disable the Dynamic PKDS access from the ADMINCNTL option. This will prevent any updates from being made while the migration action is performed. It will affect applications that use the PKDS services.

4. If the old PKDS is not empty, copy it to the larger PKDS using the JCL in SYS1.SAMPLIB(CSFPKDCP) from the HCR7750 system. If the original PKDS is an empty one, you will need to initialize the new PKDS.

```
//CSFPKDCP <JOB CARD PARAMETERS>
//*********************************************************************
//* Licensed Materials - Property of IBM                              *
//* 5694-A01                                                          *
//* Copyright IBM Corp. 2007                                          *
//*                                                                   *
//* THIS JCL COPIES ONE VSAM PKDS TO THE LARGER PKDS                  *
//*                                                                   *
//* CAUTION: This is neither a JCL procedure nor a complete JOB.      *
//* Before using this JOB step, you will have to make the following   *
//* modifications:                                                    *
//*                                                                   *
//* 1) Add the job parameters to meet your system requirements.       *
//* 2) Be sure to change CSF to the appropriate HLQ if you choose     *
//* not to use the default.                                           *
//*                                                                   *
//*********************************************************************
//STEP1 EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//INDD DD DSN=CSF.CSFPKDS.OLD,DISP=SHR
//OUTDD DD DSN=CSF.CSFPKDS,DISP=SHR
//SYSIN DD *
REPRO INFILE(INDD) OUTFILE(OUTDD)
/*
```

5. Protect the VSAM data set from use by non-authorized personnel.
6. Update the ICSF started procedures on all systems to reference the new PKDS.
7. Activate the new PKDS on each system - Refresh the PKDS from the Master Key Mgmt option on the main ICSF Administration panel.
8. Resume activity with the PKDS:
   - On HCR7751 and lower releases, enable the PKDS READ, PKDS WRITE, PKDS CREATE, and PKDS DELETE access from the ADMINCNTL option. Resume any applications that use the PKDS services.
   - On HCR7770, enable the Dynamic PKDS access from the ADMINCNTL option. Resume any applications that use the PKDS services.

Another approach is to stop ICSF, create the new PKDS, perform the copy, update the installation options data set, and restart ICSF.

Creating the TKDS

TKDS Installations need to understand and plan for the system resources required for managing the TKDS copy in virtual storage, particularly when the installation is deploying a very large TKDS. Refer to “ICSF System Resource Planning for the TKDS and Session Object Memory Areas” for guidelines. Once you understand these guidelines, refer to “Steps to create the TKDS” on page 24 for step-by-step instructions.

ICSF System Resource Planning for the TKDS and Session Object Memory Areas

Like the CKDS and PKDS, ICSF manages a mirror copy of the TKDS data set in protected, private virtual storage to optimize cryptographic workload access to persistent PKCS #11 objects (keys, certificates, and so on). Also like the CKDS and PKDS, the in-storage TKDS copy must be accommodated with sufficient system central storage and auxiliary paging space resources. Unfortunately, the variable length nature of PKCS #11 objects makes resource estimating for the TKDS difficult. The best way to estimate the virtual storage requirement for an existing,

stable TKDS (one that is not experiencing significant dynamic PKCS #11 object creation or deletion activity) is to determine the actual size of the used DATA portion of the TKDS and multiply this by 3. The following formula is provided to help you calculate the required system virtual storage backing resource for an active in-storage TKDS. In this formula HI-A-RBA is the allocated relative byte address for the data component of a TKDS VSAM data set. The IDCAMS LISTCAT command output for a TKDS VSAM data set can be consulted to determine the HI-A-RBA value for the data component. The `%Free Space` used in this formula represents the percentage of free space in the TKDS VSAM data set. The IDCAMS EXAMINE DATATEST command output can be consulted to determine the percentage of free space.

```
HI-A-RBA x ( ( 100 - %Free Space ) / 100 ) x 3
```

For example, if the DATA HI-A-RBA has the value 1622016 with 56% free space, then the virtual storage requirement estimate would be 1622016 x (44/100) x 6 = 4282122 bytes or 4182 Kilobytes.

In addition to the persistent PKCS #11 objects stored in the TKDS, applications may also make use of temporary (session) objects. These too occupy ICSF protected, private virtual storage and should be accounted for. However, since these objects are not stored in the TKDS, it is impossible to estimate their virtual storage requirements without having some knowledge of the applications that are using PKCS #11. Fortunately, most applications that use PKCS #11 use only a small number of PKCS #11 session objects and their storage requirements are already factored into the TKDS estimate above. However, some applications, such as TCP/IP's IPSec, use session objects exclusively, and may use a large number of them. Estimating the virtual storage requirements for these is beyond the scope of this document. Note that applications using PKCS #11 session objects have an overall upper limit of 128 Megabytes per application address space for session objects.

Note: The output from the formula above should be added to the outputs calculated from the formulas in "ICSF System Resource Planning for the CKDS" on page 15 and "ICSF System Resource Planning for the PKDS" on page 20. This will give you the required system virtual storage backing resource for all of ICSF's KDS data sets. This value represents the required amount of virtual storage for a given instance of ICSF. For a set of KDS data sets shared across a sysplex environment, every active ICSF in the sysplex will have an equivalent resource requirement.

Steps to create the TKDS

To enable applications to create and use persistent PKCS #11 tokens and objects using the PKCS #11 services, the TKDS must be allocated and the TKDS data set name must be specified on the TKDSN parameter of the options data set when you first start ICSF.

The TKDS must be a key-sequenced data set with variable length records. Allocate the TKDS on a permanently resident volume.

For detailed information about calculating space for a VSAM data set and an explanation of keyed-direct update processing and what happens when control area and control interval splits occur, see *z/OS DFSMS Access Method Services for Catalogs*.

1. Determine the amount of primary space you need to allocate for the TKDS.

This should reflect the total number of entries you expect the data set to contain originally. The TKDS will contain PKCS #11 tokens and objects. Each record has a maximum size of 32 KB. A record for a token will use 0.1 KB. The minimum size of a record for objects is: Data: 1 KB, Secret Key: 1.1 KB, Public Key: 1.5 KB, Private Key: 3.4 KB, Certificate: 1 KB, Domain Parameter: 1.5KB. Allocate enough space for the number of tokens to be supported and for the number of objects to be created. Note that session objects are not stored in the TKDS.

2. Determine the amount of secondary space to allocate for the TKDS.

   This should reflect the total number of entries you expect to add to the data set.

   To access tokens and objects, VSAM uses the token handle or object handle as the VSAM key. This means that VSAM adds objects to the data set in collating sequence. That is, if two objects named A and B are in the data set, A appears earlier in the data set than B. As a result, adding objects to the data set can cause multiple VSAM control interval splits and control area splits. For example, a split might occur if the data set contains objects A, B, and E and you add C. In this case, C must be placed between B and E.

   The amount of secondary space you allocate must take into account the number of control interval and control area splits that might occur. If the TKDS uses a significant amount of secondary space, you can copy it into another disk copy that you created with more primary space. You can do this by using the Access Method Services (AMS) REPRO command or the AMS EXPORT/IMPORT commands.

   The BUFFERSPACE parameter on the AMS DEFINE CLUSTER command (required by Step 3) lets VSAM optimize space for control area and control interval splits.

3. Create an empty VSAM data set to use as the TKDS. Use the AMS DEFINE CLUSTER command to define the data set and to allocate its space. ICSF provides a sample job to define the TKDS in member CSFTKDS of SYS1.SAMPLIB.

   Note: To improve security and reliability of the data that is stored on the TKDS:

   - Use the ERASE and WRITECHECK parameters on the AMS DEFINE CLUSTER command. ERASE overwrites data records with binary zeros when the TKDS cluster is deleted. WRITECHECK provides hardware verification of all data that is written to the data set.
   - Create a Security Server (RACF) data set profile for the TKDS.

4. Allocate a disk copy of the TKDS by defining a VSAM cluster as in this SYS1.SAMPLIB CSFTKDS member sample:

```
//CSFTKDS JOB = JOB CARD PARAMETERS
//*******************************************************************
//*  Licensed Materials - Property of IBM                           *
//*  5694-A01                                                       *
//*  COPYRIGHT IBM CORP. 2007, 2009                                 *
//*                                                                 *
//* THIS JCL DEFINES A VSAM TKDS TO USE FOR ICSF                    *
//*                                                                 *
//*  CAUTION: This is neither a JCL procedure nor a complete JOB.   *
//*  Before using this JOB step, you will have to make the following *
//*  modifications:                                                 *
//*                                                                 *
//*  1) Add the job parameters to meet your system requirements.    *
//*  2) Be sure to change CSF to the appropriate HLQ if you choose  *
//*     not to use the default.                                     *
//*  3) Change XXXXXX to the volid where you want your TKDS to      *
//*     reside. The TKDS needs to be on a permanently resident      *
```

```
//*     volume.                                                  *
//*                                                              *
//* NOTE: This JCL is specific for creating the TKDS. There are  *
//*       samples for each of the other key data sets.           *
//*         CKDS  - CSFCKDS JCL                                  *
//*         PKDS  - CSFPKDS JCL                                  *
//*                                                              *
//****************************************************************
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  DEFINE CLUSTER (NAME(CSF.CSFTKDS) -
                  VOLUMES(XXXXXX) -
                  RECORDS(100,50) -
                  RECORDSIZE(2200 32756) -
                  KEYS(72 0) -
                  FREESPACE(0,0) -
                  SPANNED -
                  SHAREOPTIONS(2,3)) -
            DATA (NAME(CSF.CSFTKDS.DATA) -
                  BUFFERSPACE(100000) -
                  ERASE -
                  WRITECHECK) -
           INDEX (NAME(CSF.CSFTKDS.INDEX))
/*
```

You can change and use the Job Control Language according to the needs of your installation. For more information about allocating a VSAM data set, see *z/OS DFSMS Access Method Services for Catalogs*.

Steps to create the Installation Options Data Set

The *installation options data set* is a file that you create that contains installation options. It becomes active when you start ICSF.

- The installation options data set can be a member of PARMLIB, a member of a partitioned data set, or a sequential data set.
- The format of each record in the data set must be fixed length or fixed block length.
- A physical line in the data set is 80 characters long. The system ignores any characters in positions 72 to 80 of the line.
- A logical line is one or more physical lines. You can group physical lines into a logical line by placing a comma at the end of the information. Only a comment can appear after the comma. The system ignores any other information between the comma and column 71.
- Continuation causes the next physical line to append immediately following the comma. The system removes all leading blanks on the next physical line.
- You can delimit comments by /* and */ and include them anywhere within the text. A comment cannot span physical records. The system removes comments from a logical line before parsing it. It ignores physical lines that contain only comments.
- Specify only one option setting or keyword on a logical line. (If you specify more than one, the system ignores all but the last one on the line. The system reports syntax errors, but the errors do not cause it to stop interpreting the file.)

ICSF provides two sample installation options data sets. These sample data sets use the recommended values for each option.

1. When you are starting ICSF for the first time:

a. Change the name of the data set on the CKDSN and PKDSN statements to the name of the empty VSAM datasets you created previously (in Step 3 on page 18 and Step 4 on page 21).
b. Change SSM(NO) to SSM(YES).
c. For a complete description of options you may want to change after the first start, see “Customizing ICSF after the first start” on page 37.)

2. Store the updated data set in SYS1.PARMLIB.

Note: For convenience, the installation options data set generally resides in SYS1.PARMLIB. If your cryptographic administrator does not have update access to SYS1.PARMLIB, store installation options in another data set, and RACF-protect it.

The two sample installation options data sets are as follows in SYS1.SAMPLIB:

- CSFPRM00

```
/*******************************************************************/
/*     LICENSED MATERIALS - PROPERTY OF IBM                        */
/*                                                                 */
/*     5694-A01                                                    */
/*                                                                 */
/*     COPYRIGHT IBM CORP. 1990, 2008                              */
/*                                                                 */
/*  THIS IS A SAMPLE OF THE ICSF OPTIONS DATASET                   */
/*                                                                 */
/*******************************************************************/
CKDSN(CSF.CSFCKDS)
PKDSN(CSF.CSFPKDS)
COMPAT(NO)
SSM(NO)
KEYAUTH(NO)
CKTAUTH(NO)
CHECKAUTH(NO)
TRACEENTRY(10000)
USERPARM(USERPARM)
REASONCODES(ICSF)
```

- CSFPRM01 (batch setup for SMP/E)

```
/*******************************************************************/
/*     LICENSED MATERIALS - PROPERTY OF IBM                        */
/*                                                                 */
/*     5694-A01                                                    */
/*                                                                 */
/*     COPYRIGHT IBM CORP. 1990, 2008                              */
/*                                                                 */
/*  THIS IS A SAMPLE OF THE ICSF OPTIONS DATASET                   */
/*                                                                 */
/*******************************************************************/
CKDSN(CSF.CSFCKDS)
PKDSN(CSF.CSFPKDS)
COMPAT(NO)
SSM(YES)
KEYAUTH(NO)
CKTAUTH(NO)
CHECKAUTH(NO)
TRACEENTRY(10000)
USERPARM(USERPARM)
REASONCODES(ICSF)
```

Note: See “Parameters in the installation options data set” on page 38 for descriptions of these parameters.

Use of system symbols in the options data set is supported. System symbols can be used as values for any of the parameters. System symbols must be no more than 8 characters. ICSF allows the CKDS and PKDS data set names to be a maximum of 44 characters with up to 21 qualifiers. See “Parameters in the installation options data set” on page 38 for additional information.

This example shows how system symbols could be used for the CKDS and PKDS data set names. You could use a SYS1.PARMLIB(IEASYMxx) file and modify CSFPRM01.

IEASYMxx file could contain:

```
/*-----------------------------------*/
/* SYSTEM SYMBOLS FOR ICSF CRYPTO    */
/*-----------------------------------*/
SYSDEF
 SYMDEF(&CKDSN001='CSF')
 SYMDEF(&CKDSN002='CSFCKDS')
 SYMDEF(&PKDSN001='CSF')
 SYMDEF(&PKDSN002='CSFPKDS')
```

CSFPRM01 could be modified as follows:

```
/*                                          */
/*     LICENSED MATERIALS - PROPERTY OF IBM    */
/*                                          */
/*     "RESTRICTED MATERIALS OF IBM"        */
/*     5694-A01                             */
/*                                          */
/*     (C) COPYRIGHT IBM CORP. 1990, 2008      */
/*                                          */
/*                                          */
CKDSN(&CKDSN001..&CKDSN002)
PKDSN(&PKDSN001..&PKDSN002)
COMPAT(NO)
SSM(YES)
KEYAUTH(NO)
CKTAUTH(NO)
CHECKAUTH(NO)
TRACEENTRY(10000)
USERPARM(USERPARM)
REASONCODES(ICSF)
```

When the machine or partition is IPLed, specify within the load parameter the symbol file that should be used. For example, if the previous symbol file was called IEASYM01, then within the load member, the IEASYM entry might look like `IEASYM(00,01)`; where 00 denotes the IEASYM00 file (usually the system default) and 01 denotes the IEASYM01 file.

For SMP/E, CSFPRM01 can be copied to the CPAC.PARMLIB data set. The CKDS and PKDS data set names in CSFPRM01 need to match in Server-Pac. Outside of Server-Pac, you need to copy and edit CSFPRM01.

Steps to create the ICSF Startup Procedure

ICSF provides these two job control language programs. You can use this code as the basis for your startup procedure.

- member CSF in SYS1.SAMPLIB

  ```
  //CSF PROC
  //CSF EXEC PGM=CSFINIT,REGION=0M,TIME=1440,MEMLIMIT=NOLIMIT
  //CSFPARM DD DSN=SYS1.PARMLIB(CSFPRM00),DISP=SHR
  ```

- member CSFSTART in SYS1.SAMPLIB (batch setup for SMP/E)

```
//CSFSTART PROC
//CSFSTART EXEC PGM=CSFINIT,REGION=0M,TIME=1440,MEMLIMIT=NOLIMIT
//CSFPARM DD DSN=SYS1.PARMLIB(CSFPRM01),DISP=SHR).
```

Store this startup PROC in SYS1.PROCLIB (or another suitable library). For SMP/E this startup PROC can be copied to the CPAC.PROC data set, and the data set name (SYS1.PARMLIB) can be copied to match the name you chose for the CPAC.PARMLIB data set (in which CSFPRM01 would be placed). This would work for Server-Pac. Outside of Server-Pac, you need to copy and edit CSFSTART.

1. Change or use the sample startup procedure according to your needs.
   a. In the sample code, the first line is the PROC statement. You can add one or more procedure variables to the PROC statement. For example, you can allow the system operator to decide at start time which member of the installation options data set to use. This example allows the operator to enter START CSF,M=CSFPRM01, specifying an alternate set of start-up options.

      ```
      //CSF  PROC M=CSFPRM00
      ⋮
      //CSFPARM DD DSN=MY.ICSF.PARM(&M),DISP=SHR
      ```

      You can use the same principle to change the name of a sequential data set, if you are not using a partitioned data set.
   b. The last line is the CSFPARM DD statement. The sample code specifies SYS1.PARMLIB as the data set where the installation options data set is stored. If you stored the installation options data set elsewhere, replace SYS1.PARMLIB with the name of the data set where you stored the installation options.
   c. The CSFPARM DD statement also specifies member CSFPRM00 as the name of the installation options data set. If you used a different name when you created the installation options data set (or any time you want to use other options), change this member name.
2. Store your startup procedure in SYS1.PROCLIB (or another suitable library) with a member name of your choice. (Depending on installation standards, possible names include CSF, CSFPROD, and CRYPTO.)
3. If you use Security Server (RACF), you may need to update the RACF Started Procedure Table if you define a new started task:
   a. Add the new started task name
   b. Add a RACF userid to associate with the started task. This userid requires that:
      - READ access to the data set to which the CSFPARM JCL DD statement refers
      - Define all CKDSs in every installation option data set.
      - Define all PKDSs in every installation option data set.

      See *z/OS Security Server RACF System Programmer's Guide* for more information.
   c. Optionally, you can add a RACF group name.

   Note: RACF uses the userid associated with the ICSF address space only when accessing the CKDS and PKDS named in the installation options data set and then only at ICSF startup. When you perform a CKDS or PKDS Refresh task by using the ICSF ISPF panels under TSO/E, RACF uses the TSO userid to determine access authorization. When the CKDS

Refresh and PKDS Refresh tasks are a batch job, RACF uses the userid associated with the batch address space to determine access authorization.

Steps to provide access to the ICSF panels

To provide a way for the administrator to access the ICSF panels, you can create an ICSF option on the ISPF Primary Option Menu. Access the code for the ISPF Primary Option Menu panel body and perform these steps:

1. Under the % OPTION ===> _ZCMD line, add this line:

   ```
   % <option value> - ICSF Panels
   ```

   You can specify either a letter or number for the option value. Do not use an option value that already exists in the menu.
2. On the &ZSEL= TRANS(&ZQ line, add this information:

   ```
   <option value>,''PANEL(CSF@PRIM) NEWAPPL(CSF)''
   ```

   The option value should be the same value as the option value you chose to use in the preceding step.

When you access the ISPF Primary Option Menu panel, the ICSF panels option appears on the menu. You can choose the ICSF option value to access the ICSF panels.

You must also update the logon procedure that is used by ICSF administrators who will use the ICSF panels. For example:

```
//SYSPROC DD ...
.
.
.
//        DD  DSN=CSF.SCSFCLI0,DISP=SHR
.
.
.
//ISPPLIB DD ...
.
.
.
//        DD  DSN=CSF.SCSFPNL0,DISP=SHR
.
.
.
//ISPMLIB DD ...
.
.
.
//        DD  DSN=CSF.SCSFMSG0,DISP=SHR
.
.
.
//ISPSLIB DD ...
.
.
.
//        DD  DSN=CSF.SCSFSKL0,DISP=SHR
.
.
.
// ISPTLIB
.
.
.
```

```
//        DD DSN=CSF.SCSFTLIB,DISP=SHR
.
.
.
```

An alternate method to access the ICSF panels is to use ISPF LIBDEF. Here is a sample clist.

```
/* Rexx */
/* IBMs ICSF */

address ispexec

"LIBDEF ISPPLIB DATASET ID('CSF.SCSFPNL0') STACK"
"LIBDEF ISPMLIB DATASET ID('CSF.SCSFMSG0') STACK"
"LIBDEF ISPSLIB DATASET ID('CSF.SCSFSKL0') STACK"
"LIBDEF ISPTLIB DATASET ID('CSF.SCSFTLIB') STACK"

address tso "ALTLIB ACTIVATE APPLICATION(CLIST)
            DATASET('CSF.SCSFCLI0')"
"SELECT PANEL(CSF@PRIM) NEWAPPL(CSF) PASSLIB"
address tso "ALTLIB DEACTIVATE APPLICATION(CLIST)"

"LIBDEF ISPSLIB"
"LIBDEF ISPPLIB"
"LIBDEF ISPMLIB"
"LIBDEF ISPTLIB"
```

Ensure that the latest CSFKEYS file is part of ISPTLIB; this allows scrolling of the management panels.

The *z/OS Program Directory* lists additional installation steps, and some of these steps depend on the system from which you are migrating. See *z/OS Program Directory*, other topics in this publication, and *z/OS Cryptographic Services ICSF Administrator's Guide* for details about the remaining steps.

Steps to start ICSF for the first time

Now that you have created the key data sets, the installation data set, the started procedure, and the ICSF management panels, you can start ICSF.

For additional information on starting ICSF for the first time, see Appendix D, “Helpful Hints for ICSF First Time Startup,” on page 305.

- Created an empty data set for use as a CKDS
- Specified the CKDS name in the installation options data set
- Created an empty data set for use as a PKDS
- Specified the PKDS name in the installation options data set
- If PKCS #11 support is desired, create the TKDS
- Created a startup procedure
- Installed ICSF

Steps for initializing ICSF

You must initialize ICSF and the cryptographic coprocessors:

1. Enter the START command and the startup procedure name. In this example, CSF is the name of the startup procedure.

   ```
   START CSF
   ```

   When you start ICSF, you specify the name of the ICSF startup procedure you created (see “Steps to create the ICSF Startup Procedure” on page 28). See “Starting and stopping ICSF” on page 76 for more information about starting and stopping ICSF.

Note: **CCF Systems Only**: If you start CSF using CSFSTART and then run the CSFSETMK JCL to set the master keys and initialize the CKDS, the DES master keys will be set and the PKA master keys will be set in the Cryptographic Coprocessor Feature, and the CKDS will be initialized using the appropriate pass phrase. If your environment has PCI Cryptographic Coprocessors, they will not be initialized by this process. Only the Cryptographic Coprocessor Feature is initialized. If you need to initialize the PCI Cryptographic Coprocessor, see *z/OS Cryptographic Services ICSF Administrator's Guide* for additional information on using the Pass Phrase Initialization Utility. If you re-IPL or stop ICSF and want to perform a subsequent SMP/E E-delivery, you only need to start ICSF (providing you wish to reuse the previously established options and parameters).

2. Access the ICSF panels to define a master key and initialize the CKDS and PKDS. For a description of how to use the ICSF panels to define a master key and initialize the CKDS and PKDS at first-time startup, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

   When defining a master key by specifying master key parts, **make sure the key parts are recorded and saved in a secure location.** When you are entering the key parts for the first time, be aware that **you may need to reenter these same key values at a later date to restore master key values that have been cleared.** If defining a master key using a pass phrase, realize that the same pass phrase will always produce the same master key values, and is therefore as critical and sensitive as the master key values themselves. Make sure you save the pass phrase so that you can later reenter it if needed. Because of the sensitive nature of the pass phrase, make sure you secure it in a safe place.

3. When you start ICSF for the first time, you will see different messages depending on your system hardware.
   - **z10 EC, z10 BC, and z196 with a CEX3C**:
     - First time startup messages before master keys have been loaded and the CKDS and PKDS have not been initialized:

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM124I MASTER KEY DES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY RSA ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY AES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY ECC ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CKDS IS NOT INITIALIZED.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.PKDS IS NOT INITIALIZED.
CSFM012I NO ACCESS CONTROL AVAILABLE FOR CRYPTOZ RESOURCES. ICSF PKCS11
SERVICES DISABLED.
CSFM122I PKA SERVICES WERE NOT ENABLED DURING ICSF INITIALIZATION
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM009I NO ACCESS CONTROL AVAILABLE FOR ICSF SERVICES OR KEYS
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
```

     Message CSFM124I will be issued for each CEX3C online.

Notes:

a. Message CSFM508I will not be issued if one of the Crypto Express3 Feature's cryptographic engines is configured as an accelerator (CEX3A).

b. Message CSFM122I will not be issued when your system has any CEX3C coprocessors (with the Sep. 2011 or later LIC) online. The PKA callable services control will not be active. The availability of RSA callable services will depend on the status of the RSA master key. CSFM130I is issued when the RSA master key is active and RSA callable services are available.

– First time startup messages before master keys have been loaded and sharing an initialized CKDS and PKDS:

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM123E MASTER KEY DES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, IN ERROR.
CSFM123E MASTER KEY AES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, IN ERROR.
CSFM123E MASTER KEY ECC ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, IN ERROR.
CSFM123E MASTER KEY RSA ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, IN ERROR.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM122I PKA SERVICES WERE NOT ENABLED DURING ICSF INITIALIZATION
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
```

Message CSFM123E will be issued for each CEX3C online.

Notes:

a. Message CSFM508I will not be issued if one of the Crypto Express3 Feature's cryptographic engines is configured as an accelerator (CEX3A).

b. Message CSFM122I will not be issued when your system has any CEX3C coprocessors (with the Sep. 2011 or later LIC) online. The PKA callable services control will not be active. The availability of RSA callable services will depend on the status of the RSA master key. CSFM130I is issued when the RSA master key is active and RSA callable services are available.

– Normal ICSF restart messages. Master key registers are valid and match the CKDS/PKDS.

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM129I MASTER KEY DES ON CRYPTO EXPRESS2 COPROCESSOR Epp, SERIAL
NUMBER nnnnnnnn, IS CORRECT.
CSFM129I MASTER KEY AES ON CRYPTO EXPRESS2 COPROCESSOR Epp, SERIAL
NUMBER nnnnnnnn, IS CORRECT.
CSFM129I MASTER KEY ECC ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, IS CORRECT.
CSFM129I MASTER KEY RSA ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, IS CORRECT.
```

```
CSFM129I MASTER KEY mk ON coprocessor-name cii, SERIAL
NUMBER nnnnnnn, IS CORRECT.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM400I CRYPTOGRAPHY - SERVICES ARE NOW AVAILABLE.
CSFM130I CRYPTOGRAPHY - RSA SERVICES ARE AVAILABLE.
CSFM127I CRYPTOGRAPHY - AES SERVICES ARE AVAILABLE.
CSFM130I CRYPTOGRAPHY - ECC SERVICES ARE AVAILABLE.
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
CSFM001I ICSF INITIALIZATION COMPLETE
```

Message CSFM129I will be issued for each active CEX3C.

Note: Message CSFM508I will not be issued if one of the Crypto Express3 Feature's cryptographic engines is configured as an accelerator (CEX3A).

- **z9 EC, z9 BC, z10 EC and z10 BC with a CEX2C:**
  - First time startup messages before master keys have been loaded and the CKDS and PKDS have not been initialized:

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM124I MASTER KEY DES ON CRYPTO EXPRESS2 COPROCESSOR Epp, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY AES ON CRYPTO EXPRESS2 COPROCESSOR Epp, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CKDS IS NOT INITIALIZED.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.PKDS IS NOT INITIALIZED.
CSFM012I NO ACCESS CONTROL AVAILABLE FOR CRYPTOZ RESOURCES. ICSF PKCS11
SERVICES DISABLED.
CSFM122I PKA SERVICES WERE NOT ENABLED DURING ICSF INITIALIZATION
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM009I NO ACCESS CONTROL AVAILABLE FOR ICSF SERVICES OR KEYS
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
```

Message CSFM124I will be issued for each CEX2C online.

Note: Message CSFM508I will not be issued if one of the Crypto Express2 Feature's cryptographic engines is configured as an accelerator (CEX2A).

  - First time startup messages before master keys have been loaded and sharing an initialized CKDS and PKDS:

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM123E MASTER KEY DES ON CRYPTO EXPRESS2 COPROCESSOR Epp, SERIAL
NUMBER nnnnnnnn, IN ERROR.
CSFM123E MASTER KEY AES ON CRYPTO EXPRESS2 COPROCESSOR Epp, SERIAL
NUMBER nnnnnnnn, IN ERROR.
CSFM123E MASTER KEY RSA ON CRYPTO EXPRESS2 COPROCESSOR E38, SERIAL
NUMBER nnnnnnnn, IN ERROR.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM122I PKA SERVICES WERE NOT ENABLED DURING ICSF INITIALIZATION
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
```

Message CSFM123E will be issued for each CEX2C online.

Note: CSFM508I will not be issued if one of the Crypto Express2 Feature's cryptographic engines is configured as an accelerator (CEX2A).

– Normal ICSF restart messages. Master key registers are valid and match the CKDS/PKDS.

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM124I MASTER KEY DES ON CRYPTO EXPRESS2 COPROCESSOR Epp, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY AES ON CRYPTO EXPRESS2 COPROCESSOR Epp, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM129I MASTER KEY AES ON coprocessor-name cii, SERIAL
NUMBER nnnnnnn, IS CORRECT.
CSFM129I MASTER KEY DES ON coprocessor-name cii, SERIAL
NUMBER nnnnnnn, IS CORRECT.
CSFM129I MASTER KEY RSA ON coprocessor-name cii, SERIAL
NUMBER nnnnnnn, IS CORRECT.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM400I CRYPTOGRAPHY - SERVICES ARE NOW AVAILABLE.
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
CSFM127I CRYPTOGRAPHY - AES SERVICES ARE AVAILABLE.
```

Message CSFM129I will be issued for each active CEX2C.

Note: Message CSFM508I will not be issued if one of the Crypto Express2 Feature's cryptographic engines is configured as an accelerator (CEX2A).

– ***z9 EC, z9 BC, z10 EC and z10 BC with CPACF only (no CEX2C/CEX3C or CEX2A/CEX3A)***

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM101E PKA KEY DATA SET, CSF.PKDS IS NOT INITIALIZED.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CKDS IS NOT INITIALIZED.
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM122I PKA SERVICES WERE NOT ENABLED DURING ICSF INITIALIZATION
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
```

– ***z9 EC, z9 BC, z10 EC and z10 BC with CPACF and CEX2A/CEX3A***

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.
CSFM122I PKA SERVICES WERE NOT ENABLED DURING ICSF INITIALIZATION
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM400I CRYPTOGRAPHY - SERVICES ARE NOW AVAILABLE.
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
```

You'll receive message CSFM511E for each Cryptographic Coprocessor Feature you have online.

Notes:

1. When you are starting ICSF for the first time and loading the first master key and initializing one or more CKDSs, you provide the name of the empty VSAM data set you defined previously (see step 3 on page 18) to use for the CKDS when starting ICSF.
2. While ICSF processes the data set, it requires exclusive use so that no one can make changes while the data set is read. ICSF releases the data set when it completes startup processing.
3. During CKDS initialization or refresh, ICSF reads the CKDS into extended private storage. Make sure that the region size is sufficient for reading in the entire data set. The parameter setting REGION=0M specifies the maximum available space.
4. You can add keys to the CKDS in several ways. See “The Cryptographic Key Data Set (CKDS)” on page 8 for details.
5. You can also write application programs to call services to perform cryptographic functions. See “Creating ICSF exits and generic services” on page 52 for details.

MK Initialization for SMP/E - CCF Systems Only

This task applies only to CCF systems. It does not apply to z890, z990, z9, z10, or z196 systems.

Run the JCL to set the SMP/E pass phrase for SMP/E electronic delivery only.

The JCL uses a pass phrase value to load the DES and PKA master keys. The DES and PKA master keys will be set in the Cryptographic Coprocessor Feature. *Change This Pass Phrase* is the default pass phrase. The entry point is CSFEUTIL and will have 2 or (optionally) 3 parameters. The first parameter must be the CKDS name. The second parameter (optional) is the pass phrase. The last parameter is the function PPINIT. If you do not use the default pass phrase and create your own:

- It must be sixteen to sixty-four bytes in length.
- Any EBCDIC character is allowed.
- Leading and trailing blanks will be removed.
- Embedded blanks are allowed.

Important: The same pass phrase will always produce the same master key values, and is therefore as critical and sensitive as the master key values themselves. Make sure you save the pass phrase so that you can later reenter it if needed (for example, if you need to restore master key values that have been cleared).

See this example:

```
//CSFSETMK  JOB (JOB CARD PARAMETERS)
//*******************************************************************
//*  Licensed Materials - Property of IBM                           *
//*  5694-A01                                                       *
//*  (C) Copyright IBM Corp. 2002                                   *
//*                                                                 *
//* THIS JCL USES A PASS PHRASE VALUE TO LOAD DES AND PKA MASTER KEYS*
//*                                                                 *
//*  CAUTION: This is neither a JCL procedure nor a complete JOB.   *
//*  Before using this JOB step, you will have to make the following *
```

```
//*  modifications:                                                *
//*                                                                *
//*  1) Add the job parameters to meet your system requirements.   *
//*  2) The first parameter must be the CKDS name                  *
//*  3) An optional second parameter may be used.  The second      *
//*     parameter must be 16-64 character pass phrase.             *
//*     For the pass phrase any EBCDIC character is allowed.       *
//*     Leading and trailing blanks will be removed.               *
//*     Embedded blanks are allowed.                               *
//*     It is STRONGLY recommended that the pass phrase NOT contain *
//*     any commas.  Commas are used as a delimiter for the        *
//*     parameters of the CSFEUTIL program.                        *
//*  4) The last parameter must be the function PPINIT.            *
//*  5) If the default pass phrase of "Change This Pass Phrase"    *
//*     is desired, the PARM= would look like this:                *
//*        PARM='CSF.CSFCKDS,PPINIT'                               *
//*                                                                *
//*     If a customer selected pass phrase is to be used the       *
//*     PARM= would look like this:                                *
//*        PARM='CSF.CSFCKDS,Different Pass Phrase,PPINIT'         *
//*                                                                *
//******************************************************************
//*     User supplied pass phrase of Different Pass Phrase
//STEP     EXEC PGM=CSFEUTIL,
//   PARM='CSF.CSFCKDS,Different Pass Phrase,PPINIT'
//SYSPRINT DD SYSOUT=*
//*

 OR:

//*    Using the default pass phrase of Change This Pass Phrase
//STEP     EXEC PGM=CSFEUTIL,
//   PARM='CSF.CSFCKDS,PPINIT'
//SYSPRINT DD SYSOUT=*
//*
```

In order to successfully run the CSFSETMK job, determine if these services are RACF protected in the CSFSERV class. If the services are not RACF protected in the CSFSERV class, then nothing needs to be done. If the services are protected in the CSFSERV class, then the issuer of the CSFSETMK JCL must be permitted to the profile for each service.

- CSFOWH
- CSFPMCI
- CSFCMK
- CSFREFR

Customizing ICSF after the first start

The startup procedure includes a CSFPARM DD statement, which gives the name of the installation options data set. The installation options data set includes a CKDSN option, which gives the names of the CKDS, and a PKDSN option, which gives the name of the PKDS.

After the first start, whenever you restart ICSF, the CKDS and PKDS named in the installation options data set becomes the active in-storage CKDS and PKDS.

In order for changes to the installation options dataset to take effect, **stop and restart ICSF**. To change the active in-storage CKDS or PKDS, stop and restart ICSF, or use the REFRESH option of the Master Key Management panel.

Parameters in the installation options data set

The installation options data set is an intended programming interface.

When specifying parameter values within parentheses, leading and trailing blanks are ignored. Embedded blanks may cause unpredictable results.

Support is provided for the use of system symbols in the installation options data set. System symbols can be used as values for any of the parameters. System symbols are specified in the IEASYMxx member of SYS1.PARMLIB; the IEASYM statement of the LOADxx member of SYS1.PARMLIB specifies the IEASYMxx member(s) to be used for the resolution of system symbols. This example shows the use of a system symbol for specifying the domain to be used for the start of ICSF:

```
DOMAIN(&PARDOM.)
```

When the Installation Options Data Set is processed during the start of ICSF, the value of the system symbol PARDOM will be substituted as the value of the DOMAIN parameter.

For the first start, you specified an empty VSAM data set name for the CKDS in the CKDSN option, an empty VSAM data set name for the PKDS in the PKDSN option, and SSM(YES). You may want to change these and other options for subsequent starts. Here is a complete list of installation options:

BEGIN(fmid)

Specifies that keywords following this BEGIN keyword are supported in release *fmid* and later. There must be an END statement to complete the current section. If not, an error message will be issued and ICSF will terminate.

There may be any number of BEGIN/END pairs in the data set, but they can't be nested within each other. A BEGIN must have a matching END before another BEGIN can be specified.

If the release of ICSF you're running is at this release or later, the keywords will be parsed and processed. If release of ICSF you're running is an earlier release, the keywords will be ignored.

It recommended that when your systems are all running releases that support newer keywords that the BEGIN/END pair be removed.

| The following FMIDs are supported: HCR7740, HCR7750, HCR7751,
| HCR7770, HCR7780, and HCR7790.

Here is an example of the usage of the BEGIN/END keywords.

```
keyword4       /* keyword4 is supported by all releases */
BEGIN(HCR7751)
keyword1       /* keyword1 added in HCR7751 */
keyword3       /* keyword3 added in HCR7751 */
END
BEGIN(HCR7770)
keyword2       /* keyword2 added in HCR7770 */
END
keyword5       /* keyword5 is supported by all releases */
```

CHECKAUTH(YES or NO)

| Indicates whether ICSF performs security access control checking of Supervisor State or System Key callers. If you specify CHECKAUTH(YES), ICSF issues RACROUTE calls to perform the security access control checking and the results are logged in RACF SMF records that are cut by

RACF. If you specify CHECKAUTH(NO), the authorization checks against resources in the CSFSERV class are not performed resulting in a significant performance enhancement for supervisor state and system key callers. However, the authorization checks are not logged in the RACF SMF records.

If you do not specify the CHECKAUTH option, the default is CHECKAUTH(NO).

If you configure CHECKAUTH(YES) in the ICSF options dataset, the Health Checker address space user identity must be authorized to the CSFRKL profile in class CSFSERV for the ICSFMIG7731_ICSF_RETAINED_RSAKEY migration check to successfully execute. However, you have no action to take if you choose not to run the migration check. If you configure CHECKAUTH(NO), there is no requirement to authorize the Health Checker user identity for any ICSF profiles or classes, since the check routine executes in supervisor state. This is not an implementation consideration, but rather a check deployment or activation time customer administration consideration.

CKDSN(data-set-name)
Specifies the CKDS name the system uses to start ICSF. Whenever you restart ICSF, the CKDS named in the CKDSN option becomes the active in-storage CKDS. (At first-time startup, you should specify the name of an empty VSAM data set you created to use as the CKDS.)

If you do not specify this keyword, ICSF does not become active. There is no default for this option, so you must specify a value.

CKTAUTH(YES or NO)
Decides if authentication will be performed for every CKDS record read from DASD.

Note: If the active CKDS is a variable-length record format CKDS or a fixed-length record format CKDS that does not use record level authentication, the CKTAUTH option will be ignored. It will be displayed as DISABLED on the Installation Options Display panel.

YES Indicates authentication will be performed.

NO Indicates no authentication will be performed.

COMPAT(YES, NO, or COEXIST)
Indicates whether ICSF runs in compatibility mode, non-compatibility mode, or coexistence mode with PCF.

YES Indicates **compatibility mode**.

In compatibility mode, you can run a PCF application on ICSF, because ICSF supports the PCF macros. You do not have to reassemble PCF applications to do this. You cannot start PCF at the same time as ICSF on the same operating system.

NO Indicates **non-compatibility mode**. In noncompatibility mode, you run PCF applications on PCF and ICSF applications on ICSF. You cannot run PCF applications on ICSF, because ICSF does not support the PCF macros in this mode. PCF can be started at the same time as ICSF on the same operating system. You can start ICSF and then start PCF, or you can start PCF and then start ICSF.

You should use noncompatibility mode unless you are migrating from PCF to ICSF.

COEXIST Indicates **coexistence mode**.

In coexistence mode, you can run a PCF application on PCF, or you can reassemble the PCF application to run on ICSF. To do this, you reassemble the application against coexistence macros that are shipped with ICSF. You can start PCF at the same time as ICSF on the same operating system.

If you do not specify the COMPAT option, the default value is COMPAT(NO). See "Running PCF and z/OS ICSF on the same system" on page 173 for a complete description of the COMPAT options.

When you initialize ICSF for the first time, noncompatibility mode must be active. Therefore, at first-time startup, you must specify COMPAT(NO) or allow the default to be used.

COMPENC(DES or CDMF)
This keyword is no longer supported but is tolerated.

DEFAULTWRAP(internal_wrapping_method,external_wrapping_method)
Specifies the default key wrapping for DES keys. Any token generated or updated by a service will be wrapped using the specified method unless overridden by rule array keyword or a skeleton token. The default wrapping method for internal and external tokens is specified independently.

Valid values for *internal_wrapping_method* and *external_wrapping_method* are:

ORIGINAL
Specifies the original CCA token wrapping be used: ECB wrapping for DES.

ENHANCED
Specifies the new X9.24 compliant CBC wrapping used. Note that the enhanced wrapping method is available only on the z196 with a CEX3C.

If the DEFAULTWRAP keyword is not specified, the default wrapping method will be ORIGINAL for both internal and external tokens.

DOMAIN(n)
Specifies the number of the domain that you want to use for this start of ICSF. You can specify only one domain in the options data set. Valid values are between 0 and 15 inclusive.

DOMAIN is an optional parameter. The DOMAIN parameter is only required if more than one domain is specified as the usage domain on the PR/SM panels or if running in native mode. If specified in the options data set, it will be used and it must be one of the usage domains for the LPAR.

If DOMAIN is not specified in the options data set, ICSF determines which domains are available in this LPAR. If only one domain is defined for the LPAR, ICSF will use it. If more than one is available, ICSF will issue error message CSFM409E.

The cryptographic processors support multiple sets of master key registers, which the specific domain values identify.

- The Cryptographic Coprocessor Feature has a master key register for the DES master key, the auxiliary DES master key, the signature master key and the key management master key. The auxiliary DES master key register may contain either the new or old DES master key. On the PCI Cryptographic Coprocessor, each domain has a master key register for the current, new, and old SYM-MK and RSA-MK.
- The PCIXCC/CEX2C has master key registers for the DES-MK, AES-MK and RSA-MK master keys. Each domain has a master key register for the current, new, and old DES-MK, AES-MK and RSA-MK.
- The CEX3C has master key registers for the DES-MK, AES-MK, RSA-MK, and ECC-MK master keys. Each domain has a master key for the current, new, and old DES-MK, AES-MK, RSA-MK, and ECC-MK.

For more information about partitions and running different configurations, see *z/OS Cryptographic Services ICSF Overview*.

If you run ICSF in compatibility or coexistence mode, you cannot change the domain number without re-IPLing the system. A re-IPL ensures that a program does not access a cryptographic service with a key that is encrypted under a different master key. If you are certain that no cryptographic applications are still running, you can:

1. Stop CSF
2. Start CSF in COMPAT(NO) mode with a different domain number
3. Stop CSF
4. Start CSF in compatibility or coexistence mode with a different domain number.

END Specifies the end of a section of keywords for the *fmid* from the **BEGIN(fmid)**. There must be a **BEGIN(fmid)** prior to the END. There must be an END for each **BEGIN(fmid)**. See the description for BEGIN for an example of the usage of the BEGIN and END keywords.

EXIT(ICSF-name,load-module-name,FAIL(fail-option))
Indicates information about an installation exit.

The *ICSF-name* is the identifier for each exit. Table 3 on page 42 lists all the ICSF exit names and explains when ICSF calls each exit. The load module name is the name of the module that contains the exit. The name can be any valid name your installation chooses.

Using the FAIL keyword of the EXIT statement, you specify the action ICSF, the KGUP, or the PCF conversion program takes if the exit ends abnormally. The fail action that you specify applies to subsequent calls of the exit. If an exit ends abnormally, ICSF takes a system dump. The exit is protected with an ESTAE or the ICSF service functional recovery routine (FRR).

In general, you can specify one of these values for a fail option:

NONE No action is taken. The exit can be called again and will end abnormally again.

EXIT The exit is no longer available to be called again.

SERVICE
The service or program that called the exit is no longer available to be called again.

ICSF ICSF or the key generator utility program or the PCF conversion program is ended, depending on the exit.

Some fail options are not valid for a specific exit. If you specify a fail option that is not valid, ICSF uses the next valid fail option. For example, if SERVICE is not a valid fail option for an exit, ICSF uses the EXIT option. EXIT is responsible for logging in SMF the results of any authorization checks that are made.

Table 3. Exit Identifiers and Exit Invocations

Exit Identifiers	Exit Invocations
CSFEXIT1	Gets control after the operator issues the START command, but before processing takes place. **Note:** You must not specify an EXIT statement for the first mainline exit, CSFEXIT1.
CSFEXIT2	Gets control after ICSF reads and interprets the installation options data set.
CSFEXIT3	Gets control before ICSF completes initialization.
CSFEXIT4	Gets control after the operator issues the STOP command to stop ICSF.
CSFEXIT5	Gets control when the operator issues the MODIFY command to modify ICSF.
CSFEMK	Gets control during the compatibility service for the PCF EMK macro.
CSFGKC	Gets control during the compatibility service for the PCF GENKEY macro.
CSFRTC	Gets control during the compatibility service for the PCF RETKEY macro.
CSFEDC	Gets control during the compatibility service for the PCF CIPHER macro.
CSFCKDS	Gets control when a callable service retrieves an entry from the CKDS.
CSFKGUP	Gets control during key generator utility program initialization, processing, and termination.
CSFCONVX	Gets control when you run the PCF CKDS conversion program.
CSFSRRW	Gets control when an access to a single record in the CKDS is made using the key entry hardware.
CSFAEGN	Gets control during the ANSI X9.17 EDC generate callable service.
CSFAKEX	Gets control during the ANSI X9.17 key export callable service.
CSFAKIM	Gets control during the ANSI X9.17 key import callable service.
CSFAKTR	Gets control during the ANSI X9.17 key translate callable service.
CSFATKN	Gets control during the ANSI X9.17 transport key partial notarize callable service.
CSFCKC	Gets control during the CVV key combine callable service.
CSFCKI	Gets control during the clear key import callable service.
CSFCPE	Gets control during the clear PIN encrypt callable service.
CSFCPA	Gets control during the clear PIN generate alternate callable service.
CSFCSG	Gets control during the VISA CVV service generate callable service.
CSFCSV	Gets control during the VISA CVV service verify callable service.
CSFCTT	Gets control during the ciphertext translate callable service.
CSFCTT1	Gets control during the ciphertext translate (with ALET) callable service.
CSFPGN	Gets control during the Clear PIN generate callable service.
CSFCVT	Gets control during the control vector translate callable service.
CSFCVE	Gets control during the cryptographic variable encipher callable service.
CSFDKX	Gets control during the data key export callable service.
CSFDKM	Gets control during the data key import callable service.
CSFDEC	Gets control during the decipher callable service.
CSFDEC1	Gets control during the decipher (with ALET) callable service.

Table 3. Exit Identifiers and Exit Invocations (continued)

Exit Identifiers	Exit Invocations
CSFDCO	Gets control during the decode callable service.
CSFDSG	Gets control during the digital signature generate service.
CSFDSV	Gets control during the digital signature verify callable service.
CSFDKG	Gets control during the diversified key generate callable service.
CSFENC	Gets control during the encipher callable service.
CSFENC1	Gets control during the encipher (with ALET) callable service.
CSFECO	Gets control during the encode callable service.
CSFEDH	Gets control during the ECC Diffie-Hellman callable service.
CSFEPG	Gets control during the encrypted PIN generate callable service.
CSFHMG	Gets control during the HMAC generate callable service.
CSFHMV	Gets control during the HMAC Verify callable service.
CSFPTR	Gets control during the encrypted PIN translate callable service.
CSFPVR	Gets control during the encrypted PIN verify callable service.
CSFKEX	Gets control during the key export callable service.
CSFKGN	Gets control during the key generate callable service.
CSFKGN2	Gets control during the key generate2 callable service.
CSFKIM	Gets control during the key import callable service.
CSFKPI	Gets control during the key part import callable service.
CSFKPI2	Gets control during the key part import2 callable service.
CSFKRC	Gets control during the CKDS key record create callable service.
CSFKRC2	Gets control during the CKDS key record create2 callable service.
CSFKRD	Gets control during the CKDS key record delete callable service.
CSFKRR	Gets control during the CKDS key record read callable service.
CSFKRR2	Gets control during the CKDS key record read2 callable service.
CSFKRW	Gets control during the CKDS key record write callable service.
CSFKRW2	Gets control during the CKDS key record write2 callable service.
CSFKYT	Gets control during the key test callable service.
CSFKYT2	Gets control during the key test2 callable service.
CSFKYTX	Gets control during the key test extended callable service.
CSFMDG	Gets control during the MDC generate callable service.
CSFKTR	Gets control during the key translate callable service.
CSFKTR2	Gets control during the key translate2 callable service.
CSFMGN1	Gets control during the MAC generate (with ALET) callable service.
CSFMVR	Gets control during the MAC verify callable service.
CSFMVR1	Gets control during the MAC verify (with ALET) callable service.
CSFMDG1	Gets control during the MDC generate (with ALET) callable service.
CSFMGN	Gets control during the MAC generate callable service.
CSFCKM	Gets control during the multiple clear key import callable service.
CSFSKM	Gets control during the multiple secure key import callable service.
CSFOWH	Gets control during the one-way hash generate callable service.

Table 3. Exit Identifiers and Exit Invocations (continued)

Exit Identifiers	Exit Invocations
CSFOWH1	Gets control during the one-way hash generate (with ALET) callable service.
CSFPCI	Gets control during the PCI interface callable service.
CSFPCU	Gets contol during the PIN Change/Unblock callable service
CSFPEX	Gets control during the prohibit export callable service.
CSFPEXX	Gets control during the prohibit export extended callable service.
CSFPKD	Gets control during the PKA decrypt callable service.
CSFPKE	Gets control during the PKA encrypt callable service.
CSFPKG	Gets control during the PKA key generate callable service.
CSFPKI	Gets control during the PKA key import callable service.
CSFPKT	Gets control during the PKA key translate callable service.
CSFPKTC	Gets control during the PKA key token change callable service.
CSFPKX	Gets control during the PKA Public Key Extract callable service.
CSFPKRC	Gets control during the PKDS key record create callable service.
CSFPKRD	Gets control during the PKDS key record delete callable service.
CSFPKRR	Gets control during the PKDS key record read callable service.
CSFPKRW	Gets control during the PKDS key record write callable service.
CSFPKSC	Gets control during the PKSC interface callable service.
CSFRKA	Gets control during the restrict key attribute callable service.
CSFRNG	Gets control during the random number generate callable service.
CSFRNGL	Gets control during the random number generate long callable service.
CSFRKD	Gets control during the retained key delete callable service.
CSFRKL	Gets control during the retained key list callable service.
CSFRKX	Gets control during the remote key export callable service.
CSFSKI	Gets control during the secure key import callable service.
CSFSKI2	Gets control during the secure key import2 callable service.
CSFSKY	Gets control during the secure messaging for keys callable service.
CSFSMG	Gets control during the symmetric MAC generate callable service.
CSFSMG1	Gets control during the symmetric MAC generate (with ALET) callable service.
CSFSMV	Gets control during the symmetric MAC verify callable service.
CSFSMV1	Gets control during the symmetric MAC verify (with ALET) callable service.
CSFSPN	Gets control during the secure messaging for PINs callable service.
CSFSBC	Gets control during the SET block compose callable service.
CSFSBD	Gets control during the SET block decompose callable service.
CSFSYX	Gets control during the symmetric key export callable service.
CSFSYG	Gets control during the symmetric key generate callable service.
CSFSYI	Gets control during the symmetric key import callable service.
CSFSYI2	Gets control during the symmetric key import2 callable service.
CSFT31X	Gets control during the TR-31 export callable service.
CSFT31I	Gets control during the TR-31 import callable service.
CSFTBC	Gets control during the trusted block create callable service.

Table 3. Exit Identifiers and Exit Invocations (continued)

Exit Identifiers	Exit Invocations
CSFTCK	Gets control during the transform CDMF key callable service.
CSFTRV	Gets control during the transaction validation callable service
CSFUDK	Gets control during the user derived key callable service.

See Chapter 5, “Installation Exits,” on page 111 for a detailed description of each ICSF exit, including the valid fail options.

Note: z/OS no longer ships IBM-supplied security exit routines; the security exit points remain. Users of z/OS should use the Security Server (RACF) or an equivalent product to obtain access checking of services and keys. ICSF no longer needs these exit routines.

FIPSMODE(YES or COMPAT or NO,FAIL(fail-option))

Indicates whether z/OS PKCS #11 services must run in compliance with the Federal Information Processing Standard Security Requirements for Cryptographic Modules, referred to as FIPS 140-2. FIPS 140-2, published by the National Institute of Standards and Technology (NIST), is a standard that defines rules and restrictions for how cryptographic modules should protect sensitive or valuable information. The standard is available at http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

By configuring z/OS PKCS #11 services to operate in compliance with FIPS 140-2 specifications, installations or individual applications can use the z/OS PKCS #11 services in a way that allows only the cryptographic algorithms (including key sizes) approved by the standard, and restricts access to the algorithms that are not approved. For more information, refer to *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

YES Indicates that the z/OS PKCS #11 services will operate in *FIPS standard mode*. Any application using the PKCS #11 services will be forced to use those services in a FIPS-compliant manner. Applications will not have access to the algorithms or key sizes not approved by FIPS 140-2. In addition, ICSF initialization will test that it is running on an IBM System z model type, and a version and release of z/OS, that supports FIPS. If so, then ICSF will perform a series of cryptographic known answer tests as required by the FIPS 140-2 standard. If any of these initialization tests should fail, the action the ICSF initialization process takes will depend on the *fail-option* specified.

COMPAT

Indicates that the z/OS PKCS #11 services will operate in *FIPS compatibility mode*. This mode is intended for installations where only certain z/OS PKCS #11 applications must comply with the FIPS 140-2 standard, while other applications do not. In this mode, the PKCS #11 services can be further configured so that the applications that do not need to comply with the FIPS 140-2 standard are not restricted from using any of the PKCS #11 algorithms, while applications that must comply with the standard are restricted from using the non-approved algorithms. By default, the COMPAT option will have the same effect as the YES option, and all applications using the PKCS #11 services will be forced to use those services in a FIPS-compliant manner. However, additional specifications can be made:

- at the PKCS #11 token and application level, by creating FIPSEXEMPT.*token-name* resource profiles in the CRYPTOZ class. A FIPSEXEMPT.*token-name* resource exists for each token. User IDs with READ access authority to a FIPSEXEMPT.*token-name* are exempt from FIPS compliance, while user IDs with access authority NONE can only use the PKCS #11 services in a FIPS-compliant manner.
- within applications themselves for individual keys. When an application creates a key, the application can specify that the key must be used in a FIPS 140-2 compliant fashion. The application can specify this by setting the Boolean key attribute CKA_IBM_FIPS140 to TRUE.

When the COMPAT option is specified, ICSF initialization will test that it is running on an IBM System z model type, and a version and release of z/OS, that supports FIPS. If so, then ICSF will perform a series of cryptographic known answer tests as required by the FIPS 140-2 standard. If any of these initialization tests should fail, the action the ICSF initialization process takes will depend on the *fail-option* specified.

NO Indicates that no z/OS PKCS #11 applications at the installation need to comply with the FIPS 140-2 standard, and ICSF will bypass the extra processing that is required to ensure FIPS compliance. FIPSEXEMPT.*token-name* profiles, if they exist, will not be examined. Requests to generate or use a key with CKA_IBM_FIPS140=TRUE will result in a failure return code.

The *fail-option* is either YES or NO, and indicates what action the ICSF initialization process should take if any of the initialization tests (performed when **FIPSMODE** is **YES** or **COMPAT**) should fail.

YES indicates that ICSF should end abnormally if any of the tests fail.

NO Specifies that ICSF initialization process should continue even if one or more of the tests fail. However, z/OS PKCS #11 support will be limited or nonexistent depending on the test that failed.

- If ICSF is running on an IBM system z model type or with a version of z/OS that does not support FIPS, most FIPS processing is bypassed. PKCS #11 callable services will be available, but ICSF will not adhere to FIPS 140 restrictions. Requests to generate or use a key with CKA_IBM_FIPS140=TRUE will result in a failure return code.
- If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable.

KEYAUTH(YES or NO)

Indicates whether or not ICSF authenticates a key entry after ICSF retrieves one from the in-storage CKDS. If you specify KEYAUTH(YES), ICSF authenticates the key. ICSF generates a message authentication code (MAC) for each key entry in the CKDS when you create or update the entry. If you specify KEYAUTH(YES), ICSF performs a MAC verification to ensure that the entry has not changed. If you specify KEYAUTH(NO), ICSF does not perform this authentication and gains a small performance enhancement. If you do not specify the KEYAUTH option, the default value is KEYAUTH(NO).

Note: If the active CKDS is a variable-length record format CKDS or a fixed-length record format CKDS that does not use record level authentication, the KEYAUTH option will be ignored. It will be displayed as DISABLED on the Installation Options Display panel.

MAXLEN(n)
Defines the maximum length of characters in a text string, including any necessary padding, for some callable service requests. For example, this option defines the maximum length of the text the encipher service encrypts for each call. Specify *n* as a decimal value from 1024 through 2147483647. If you do not specify the MAXLEN option, the default value is MAXLEN(65535).

The MAXLEN parameter may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647). If a value greater than this is specified, an error will result and ICSF will not start.

Note: MAXLEN is no longer displayed on the Installation Option Display panel.

PKDSCACHE
This keyword is no longer supported but is tolerated.

PKDSN(data-set-name)
Specifies the PKDS name the system uses to start ICSF. Whenever you restart ICSF, the PKDS named in the PKDSN option becomes the active PKDS. (At first-time startup, you should specify the name of an empty VSAM data set you created to use as the PKDS.)

If you do not specify this keyword, ICSF does not become active. There is no default for this option, so you must specify a value.

REASONCODES(ICSF or TSS)
Specifies which set of reason codes are to be returned from callable services. If you do not specify the REASONCODES option, the default of REASONCODES(ICSF) is used. If you specify REASONCODES(TSS), TSS reason codes will be returned. If there is a 1-to-1 mapping, the codes will be converted.

If you specified REASONCODES(ICSF) and your service was processed on a PCICC, PCIXCC, CEX2C, or CEX3C, a TSS reason code may be returned if there is no 1–1 corresponding ICSF reason code.

SERVICE(service-number,load-module-name,FAIL(fail-option))
Indicates information about an installation-defined service.

ICSF allows an installation to define its own service similar to an ICSF callable service. The *service-number* specifies a number that identifies the service to ICSF. The valid service numbers are 1 through 32767, inclusive. This set of service numbers is valid for both installation-defined services and UDX services. The service number of an installation-defined service must not be the same as the service number of a UDX service. The *load-module-name* is the name of the module that contains the service. During ICSF startup, ICSF loads this module and binds it to the *service-number* you specified.

The *fail-option* is YES or NO, indicating the action ICSF should take if loading the service ends abnormally.

YES Specifies that ICSF ends abnormally if your service cannot be loaded.

NO Specifies that ICSF continues to start if your service cannot be loaded.

If the service itself ends abnormally, ICSF does not end, but takes a system dump instead. The ICSF service functional recovery routine (FRR) protects the service.

See Chapter 6, “Installation-Defined Callable Services,” on page 159 for a description of how to write and run an installation-defined callable service.

SSM(YES or NO)

Specifies whether or not an installation can enable special secure mode (SSM) while running ICSF. SSM lowers the security of your system to let you enter clear keys and generate clear PINs. You must enable SSM for KGUP to permit generation or entry of clear keys and to enable the secure key import or clear pin generate callable services.

YES Indicates that you can enable the SSM.

NO Indicates that you cannot enable the SSM.

If you do not specify the SSM option, the default value is SSM(NO).

Note: **CCF Systems only**: When you initialize ICSF for the first time, SSM must be active. Therefore, at first-time startup, you must specify SSM(YES).

If you are running with the IBM @server zSeries 900, S/390 Enterprise Servers and S/390 Multiprise servers, you must perform these tasks to make SSM active:

- Specify SSM(YES) in the installation options data set
- Enable SSM in the cryptographic hardware
- When running under a logical partition (LPAR), enable SSM for each partition.

SSM must be enabled or disabled in ALL places or errors may be logged and functions will not work as expected.

Note: The setting of the Environment Control Mask (ECM) enables SSM. Without TKE, the supplied ECM enables SSM. With TKE, you can set the ECM directly; the supplied ECM enables SSM, but you have the ability to disable it. For details, refer to *Support Element Operations Guide* and *z/OS Cryptographic Services ICSF TKE Workstation User's Guide*.

SYSPLEXCKDS(YES or NO,FAIL(*fail-option*))

SYSPLEXCKDS(YES,FAIL(*fail-option*))

ICSF will join the ICSF sysplex group SYSICSF and this system will participate in sysplex-wide consistency for CKDS data.

SYSPLEXCKDS(YES,FAIL(YES))

Indicates ICSF initialization will end abnormally if the ICSF cross-system services environment cannot be established during ICSF initialization due to a failure creating the CKDS latch set or a failure to join the ICSF sysplex group.

SYSPLEXCKDS(YES,FAIL(NO))

Indicates ICSF initialization processing will continue

even if the request to create a CKDS latch set fails or the request to join the ICSF sysplex group fails. The system will not be notified of updates to the CKDS by other members of the ICSF sysplex group. A value of either FAIL(YES) or FAIL(NO) will be ignored with SYSPLEXCKDS(NO,...).

SYSPLEXCKDS(NO,FAIL(*fail-option***))**
CKDS update processing proceeds as it does today (i.e. no Cross-System Services task will be initialized, nor will any XCF signalling be performed when an update to a CKDS record occurs).

If you do not specify the SYSPLEXCKDS option, the default value is SYSPEXCKDS(NO,FAIL(NO)).

SYSPLEXPKDS(YES or NO,FAIL(*fail-option***))**
ICSF will join the ICSF sysplex group SYSICSF and this system will participate in sysplex-wide consistency for PKDS data.

SYSPLEXPKDS(YES,FAIL(*fail-option***))**
ICSF will join the ICSF sysplex group SYSICSFP and this system will participate in sysplex-wide consistency for PKDS data.

SYSPLEXPKDS(YES,FAIL(YES))
Indicates ICSF initialization will fail to join the sysplex if the ICSF cross-system services environment cannot be established during ICSF initialization due to a failure creating the PKDS latch set or a failure to join the ICSF sysplex group.

SYSPLEXPKDS(YES,FAIL(NO))
Indicates ICSF initialization processing will continue even if the request to create a PKDS latch set fails or the request to join the ICSF sysplex group fails. The system will not be notified of updates to the PKDS by other members of the ICSF sysplex group. A value of either FAIL(YES) or FAIL(NO) will be ignored with SYSPLEXPKDS(NO,...).

SYSPLEXPKDS(NO,FAIL(*fail-option***))**
PKDS update processing proceeds without trying to join the ICSF sysplex group.

If you do not specify the **SYSPLEXPKDS** option, the default value is **SYSPEXPKDS(NO,FAIL(NO))**.

SYSPLEXTKDS(YES or NO,FAIL(*fail-option***))**

ICSF will join the ICSF sysplex group SYSICSF and this system will participate in sysplex-wide consistency for TKDS data.

Note: TKDSN needs to be specified for this to work. See on page 50.

SYSPLEXTKDS(NO,FAIL(*fail-option***))**
Indicates no XCF signalling will be performed when an update to a TKDS record occurs.

SYSPLEXTKDS(YES,FAIL(*fail-option***))**
Indicates the system will be notified of updates made to the

TKDS by other members of the sysplex who have also specified SYSPLEXTKDS(YES,FAIL(*fail-option*)).

SYSPLEXTKDS(YES,FAIL(YES))
Indicates ICSF will terminate abnormally if there is a failure creating the TKDS latch set.

SYSPLEXTKDS(YES,FAIL(NO))
Indicates ICSF initialization processing will continue even if the request to create a TKDS latch set fails with an environment failure. This system will not be notified of updates to the TKDS by other members of the ICSF sysplex group.

If you do not specify the SYSPLEXTKDS option, the default value is SYSPLEXTKDS(NO,FAIL(NO)) is the default.

TKDSN(data-set-name)
The name of an existing TKDS or an empty VSAM data set to be used as the TKDS. To enable applications to create and use persistent PKCS #11 tokens and objects using the PKCS #11 services, this option must be specified.

TRACEENTRY(n)
Specifies the number, *n*, of trace buffers to allocate for ICSF tracing. Specify *n* as a decimal value from 10000 through 500000, inclusive. The default is 10000.

You should set this parameter to the maximum in case you ever need this trace material.

UDX(UDX-id,service-number,load-module-name,'comment_text',FAIL(fail-option))
ICSF allows the development of User Defined Extensions for the PCICC, PCIXCC, CEX2C, or CEX3C. The *UDX-id* is supplied to the installation when the UDX is developed. The *service-number* specifies a number that identifies the service to ICSF. The valid service numbers are 1 to 32767, inclusive. This set of service numbers is valid for both installation-defined services and UDX services. The service number of a UDX service must not be the same as the service number of an installation-defined service. The *load-module-name* is the name of the module that contains this service. During ICSF startup, ICSF loads this module and binds it to the service-number that was specified. A *comment* may be specified. The positional parameter is required. The comment consists of up to 40 EBCDIC characters, and may include imbedded blank characters. The comment text is enclosed by single quotes. If no comment text is desired, two contiguous single quotes should be specified.

The *fail-option* is YES or NO, indicating the action ICSF should take if loading the service ends abnormally. If the service itself ends abnormally, ICSF does not end, but takes a system dump instead.

YES Specifies that ICSF ends abnormally if your service cannot be loaded.

NO Specifies that ICSF continues to start if your service cannot be loaded.

The User Defined Extension (UDX) is responsible for logging in SMF the results of any authorization checks that were made.

USERPARM(value)

Specifies an 8-byte field for installation use. The Installation Option Display panel displays this value, which is stored in the Cryptographic Communication Vector Table (CCVT) in the *CCVT_USERPARM* field. An application program or installation exit can examine this field and use it to set system environment information. The default is eight blanks.

WAITLIST(data_set_name)

This optional parameter can be used if you have ICSF with CICS (CICS 4.1 or higher) installed. It specifies a customer modifiable data set will be used to determine names of the services to be placed into the ICSF CICS Wait List. A sample data set is provided by ICSF via member CSFWTL00 of SYS1.SAMPLIB with CCFs/PCICCs and CSFWTL01 for systems with PCIXCCs, CEX2Cs, or CEX3Cs. The sample data set contains the same entries as the default ICSF CICS Wait List (i.e., the data set contains the names of all ICSF callable services which, by default, will be driven through the CICS TRUE). The WAITLIST option should be added to the Installation Options data set under these conditions.

- Non-CICS customers will not specify a WAITLIST keyword. You must ensure, however, that if you have any existing CICS applications which invoke any of the ICSF services in the Wait List and if these applications were linked with ICSF stubs at a pre-OS/390 V2R10 level, then these applications must be re-linked with the current ICSF stubs.

  If running on a z990, z890, z9 EC or z9 BC however, you must also ensure that any existing CICS applications which invoke any of these services are re-linked to ensure that the correct version of the stub is used: CSNBCKI, CSNBCKM, CSNBDEC, CSNBENC, CSNBKYTX, CSNBMGN, CSNBMVR, CSNBPEXX, CSNBRNG
- CICS customers who do not want to make use of CICS TRUE must either not enable the TRUE or must specify a WAITLIST keyword and point to an empty wait list data set (or specify WAITLIST(DUMMY)) in the Installation Options data set.
- CICS customers who wish to modify the ICSF default CICS Wait List should modify the sample Wait List data set supplied in member CSFWTL00 or CSFWTL01 of SYS1.SAMPLIB. The WAITLIST keyword in the Installation Options Data Set should be set to point to this modified data set. If you have any existing CICS applications which invoke any of the ICSF services in the Wait List and if these applications were linked with ICSF stubs at a pre-OS/390 V2R10 level, then these applications must be re-linked with the current ICSF stubs.

  If running on a z990, z890, z9 EC or z9 BC any existing CICS applications which invoke any of these services are re-linked to ensure that the correct version of the stub is used: CSNBCKI, CSNBCKM, CSNBDEC, CSNBENC, CSNBKYTX, CSNBMGN, CSNBMVR, CSNBPEXX, CSNBRNG.

  For additional information on the CICS Attachment Facility, see Appendix C, “CICS-ICSF Attachment Facility,” on page 299.

Improving CKDS performance

To improve the performance of CKDS operations during KGUP runs, use the Batch Local Shared Resource (BLSR) with Deferred Write for all KGUP runs. See *MVS Batch Local Shared Resources* for more information.

Dispatching priority of ICSF

To avoid performance problems, the dispatching priority of ICSF should be set at least as high as that of the highest task using ICSF.

Creating ICSF exits and generic services

You need not code any exits or generic services before using ICSF productively.

Developing callable service exits and generic services requires skill in assembler programming in a cross memory environment. To help with testing, the system programmer might want to use the WTO macro with the LINKAGE=BRANCH keyword to issue console messages while in cross-memory mode. (See “service exits” on page 115 for more information.)

Chapter 3. Migration

This topic describes migration considerations.

Your plan for migrating to the new level of ICSF should include information from a variety of sources. These sources of information describe topics such as coexistence, service, hardware and software requirements, installation and migration procedures, and interface changes.

Attention: Although you are migrating to a new release, you should review the information in Chapter 2, "Installation, Initialization, and Customization," on page 13; especially review customization steps that may have changed since your last migration.

If this migration also includes a hardware upgrade be sure to have your Master Keys available. Once Migration is complete, the Master Keys may need to be loaded and set. Review Chapter 2, "Installation, Initialization, and Customization," on page 13 for information on setting Master Keys.

An IPL is required when installing a new release of ICSF (it is possible for ICSF control blocks like the DACC and CCVT to persist in storage across an ICSF restart).

Consult these documents for information on migration and installation:

- *z/OS Migration*

  This publication describes the migration tasks for z/OS at a system and element level.

  This publication, which is supplied with your product order, provides information about installing your z/OS system. In addition to specific information about ICSF, this publication contains information about all of the z/OS elements.
- *z/OS Planning for Installation*

  This publication describes the installation requirements for z/OS at a system and element level. It includes hardware, software, and service requirements for both the driving and target systems. It also describes any coexistence considerations and actions.
- *z/OS Program Directory*

  This publication, which is provided with your z/OS product order, leads you through the specific installation steps for ICSF and the other z/OS elements.
- *ServerPac Installing Your Order*

  This is the order-customized, installation publication for using the ServerPac Installation method. Be sure to review "Appendix A. Product Information", which describes data sets supplied, jobs or procedures that have been completed for you, and product status. IBM may have run jobs or made updates to PARMLIB or other system control data sets. These updates could affect your migration.

Terminology

This topic describes some terms you may need to know as you use this publication.

Migration Activities that relate to the installation of a new version or release of a program to replace a previous level. Completion of these activities

ensures that the applications and resources on your system will function correctly at the new level.

Coexistence

Two or more systems at different levels (for example, software, service or operational levels) that share resources. Coexistence includes the ability of a system to respond in these ways to a new function that was introduced on another system with which it shares resources: ignore a new function, terminate gracefully, support a new function. These are examples of multisystem configurations in which resource sharing can occur:

- A single system running multiple LPARs
- A single processor that is time-sliced to run different levels of the system (for example, during different times of the day)
- Two or more systems running separate processors
- A Parallel Sysplex configuration (also includes a basic sysplex)

Migrating from earlier software releases

These topics describe common activities and considerations that should be considered when you migrate from an earlier release of ICSF to FMID HCR7790.

Callable Services

The following table summarizes the new and changes callable services for ICSF FMID HCR7790. For complete reference information on these callable services, refer to *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Table 4. Summary of new and changed ICSF callable services

Callable service	Release	Description
Clear PIN Generate	HCR7790	**Changed:** Increased X9.8 PIN block security, stored PIN decimalization tables support.
Clear PIN Generate Alternate	HCR7790	**Changed:** Increased X9.8 PIN block security, stored PIN decimalization tables support.
Control Vector Generate	HCR7790	**Changed:** ANSI TR-31 key block support.
Coordinated KDS Administration	HCR7790	**New:** Support for a coordinated CKDS refresh or a coordinated CKDS reencipher and master key change.
CVV Key Combine	HCR7790	**New:** Double-length CVV key support
Digital Signature Verify	HCR7790	**Changed:** 4096-bit RSA clear key hardware support.
ECC Diffie-Hellman	HCR7790	**New:** Creation of: • Symmetric key material from a pair of ECC keys using the Elliptic Curve Diffie-Hellman protocol using the Static Unified Model • “Z” - The “secret” material output from D-H process
Encrypted PIN Generate	HCR7790	**Changed:** Increased X9.8 PIN block security, stored PIN decimalization tables support.
Encrypted PIN Verify	HCR7790	**Changed:** Increased X9.8 PIN block security, stored PIN decimalization tables support.
ICSF Query Algorithm	HCR7790	**Changed:** 4096-bit RSA clear key hardware support.

Table 4. Summary of new and changed ICSF callable services (continued)

Callable service	Release	Description
ICSF Query Facility	HCR7790	**Changed:** • Increased X9.8 PIN block security, stored PIN decimalization tables support. • ECC Diffie-Hellman (ECCDH) and ECC key wrapping support. • 4096-bit RSA clear key hardware support.
Key Generate2	HCR7790	**Changed:** AES key type support
Key Part Import2	HCR7790	**Changed:** AES key type support
Key Test2	HCR7790	**Changed:** • AES key type support • ANSI TR-31 key block support.
Key Token Build	HCR7790	**Changed:** ANSI TR-31 key block support.
Key Token Build2	HCR7790	**Changed:** AES key type support
Key Translate2	HCR7790	**Changed:** AES key type support
PKA Decrypt	HCR7790	**Changed:** 4096-bit RSA clear key hardware support.
PKA Encrypt	HCR7790	**Changed:** 4096-bit RSA clear key hardware support.
PKA Key Generate	HCR7790	**Changed:** Support for External ECC Keys (ECC Keys encrypted by an AES KEK)
PKA Key Import	HCR7790	**Changed:** Support for External ECC Keys (ECC Keys encrypted by an AES KEK)
PKCS #11 Derive key	HCR7790	**Changed:** Support for hardware generated “z” value.
PKCS #11 Derive multiple keys	HCR7790	**Changed:** Support for hardware generated “z” value.
PKCS #11 Private key sign	HCR7790	**Changed:** 4096-bit RSA clear key hardware support.
PKCS #11 Public key verify	HCR7790	**Changed:** 4096-bit RSA clear key hardware support.
PKCS #11 Unwrap key	HCR7790	**Changed:** 4096-bit RSA clear key hardware support.
Restrict Key Attribute	HCR7790	**Changed:** • AES key type support • ANSI TR-31 key block support.
Secure Key Import2	HCR7790	**Changed:** AES key type support
Symmetric Algorithm Decipher	HCR7790	**Changed:** AES key type support
Symmetric Algorithm Encipher	HCR7790	**Changed:** AES key type support
Symmetric Key Export	HCR7790	**Changed:** • AES key type support • Support for PKCS#1 OAEP data block formatting with the SHA-256 hash method
Symmetric Key Generate	HCR7790	**Changed:** Support for PKCS#1 OAEP data block formatting with the SHA-256 hash method
Symmetric Key Import	HCR7790	**Changed:** Support for PKCS#1 OAEP data block formatting with the SHA-256 hash method
Symmetric Key Import2	HCR7790	**Changed:** AES key type support
TR-31 Export	HCR7790	**New:** ANSI TR-31 key block support.
TR-31 Import	HCR7790	**New:** ANSI TR-31 key block support.

Table 4. Summary of new and changed ICSF callable services (continued)

Callable service	Release	Description
TR-31 Optional Data Build	HCR7790	**New:** ANSI TR-31 key block support.
TR-31 Optional Data Read	HCR7790	**New:** ANSI TR-31 key block support.
TR-31 Parse	HCR7790	**New:** ANSI TR-31 key block support.
VISA CVV Service Verify	HCR7790	**Changed:** Double-length CVV key support
VISA CVV Service Generate	HCR7790	**Changed:** Double-length CVV key support
ANSI X9.17 EDC Generate	HCR7780	**Changed:** Support for invocation in AMODE(64).
ANSI X9.17 Key Export	HCR7780	**Changed:** Support for invocation in AMODE(64).
ANSI X9.17 Key Import	HCR7780	**Changed:** Support for invocation in AMODE(64).
ANSI X9.17 Key Translate	HCR7780	**Changed:** Support for invocation in AMODE(64).
ANSI X9.17 Transport Key Partial Notarize	HCR7780	**Changed:** Support for invocation in AMODE(64).
Ciphertext Translate	HCR7780	**Changed:** Support for invocation in AMODE(64).
Clear PIN Encrypt	HCR7780	**Changed:** Support for invocation in AMODE(64).
Clear PIN Generate	HCR7780	**Changed:** Support for invocation in AMODE(64).
Clear PIN Generate Alternate	HCR7780	**Changed:** Support for invocation in AMODE(64).
Control Vector Generate	HCR7780	**Changed:** Support for invocation in AMODE(64).
Control Vector Translate	HCR7780	**Changed:** Support for invocation in AMODE(64).
Cryptographic Variable Encipher	HCR7780	**Changed:** Support for invocation in AMODE(64).
Data Key Export	HCR7780	**Changed:** Support for invocation in AMODE(64).
Data Key Import	HCR7780	**Changed:** Support for invocation in AMODE(64).
Decipher	HCR7780	**Changed:** Support for invocation in AMODE(64).
Decode	HCR7780	**Changed:** Support for invocation in AMODE(64).
Digital Signature Generate	HCR7780	**Changed:** Elliptic Curve Cryptography (ECC) support.
Digital Signature Verify	HCR7780	**Changed:** Elliptic Curve Cryptography (ECC) support.
Diversified Key Generate	HCR7780	**Changed:** • Support for invocation in AMODE(64). • New rule array keywords to support enhanced key wrapping method.
Encipher	HCR7780	**Changed:** Support for invocation in AMODE(64).
Encode	HCR7780	**Changed:** Support for invocation in AMODE(64).
Encrypted PIN Generate	HCR7780	**Changed:** Support for invocation in AMODE(64).
Encrypted PIN Translate	HCR7780	**Changed:** Support for invocation in AMODE(64).
Encrypted PIN Verify	HCR7780	**Changed:** Support for invocation in AMODE(64).
HMAC Generate	HCR7780	**New:** Support for CCA key management of HMAC keys.
HMAC Verify	HCR7780	**New:** Support for CCA key management of HMAC keys.
Key Export	HCR7780	**Changed:** Support for invocation in AMODE(64).
Key Generate2	HCR7780	**New:** Support for CCA key management of HMAC keys.
Key Import	HCR7780	**Changed:** Support for invocation in AMODE(64).

Table 4. Summary of new and changed ICSF callable services (continued)

Callable service	Release	Description
Key Part Import	HCR7780	**Changed:** • Support for invocation in AMODE(64). • New rule array keywords to support enhanced key wrapping method.
Key Part Import2	HCR7780	**New:** Support for CCA key management of HMAC keys.
Key Record Create	HCR7780	**Changed:** Support for invocation in AMODE(64).
Key Record Create2	HCR7780	**New:** Support for CCA key management of HMAC keys.
Key Record Delete	HCR7780	**Changed:** Support for invocation in AMODE(64).
Key Record Read	HCR7780	**Changed:** Support for invocation in AMODE(64).
Key Record Read2	HCR7780	**New:** Support for CCA key management of HMAC keys.
Key Record Write	HCR7780	**Changed:** Support for invocation in AMODE(64).
Key Record Write2	HCR7780	**New:** Support for CCA key management of HMAC keys.
Key Test	HCR7780	**Changed:** Support for invocation in AMODE(64).
Key Test Extended	HCR7780	**Changed:** Support for invocation in AMODE(64).
Key Test2	HCR7780	**New:** Support for CCA key management of HMAC keys.
Key Token Build	HCR7780	**Changed:** • Support for invocation in AMODE(64). • New rule array keywords to support enhanced key wrapping method.
Key Token Build2	HCR7780	**New:** Support for CCA key management of HMAC keys.
Key Translate	HCR7780	**Changed:** Support for invocation in AMODE(64).
Key Translate2	HCR7780	**New:** Support for CCA key management of HMAC keys.
MAC Generate	HCR7780	**Changed:** Support for invocation in AMODE(64).
MAC Verify	HCR7780	**Changed:** Support for invocation in AMODE(64).
MDC Generate	HCR7780	**Changed:** Support for invocation in AMODE(64).
Multiple Clear Key Import	HCR7780	**Changed:** New rule array keywords to support enhanced key wrapping method.
Multiple Secure Key Import	HCR7780	**Changed:** • Support for invocation in AMODE(64). • New rule array keywords to support enhanced key wrapping method.
One-Way Hash Generate	HCR7780	**New:** Support for invocation in AMODE(64).
PIN Change/Unblock	HCR7780	**Changed:** Support for invocation in AMODE(64).
PKA Key Generate	HCR7780	**Changed:** Elliptic Curve Cryptography (ECC) support.
PKA Key Import	HCR7780	**Changed:** Elliptic Curve Cryptography (ECC) support.
PKA Key Token Build	HCR7780	**Changed:** Elliptic Curve Cryptography (ECC) support.
PKA Key Token Change	HCR7780	**Changed:** • Elliptic Curve Cryptography (ECC) support. • Support for invocation in AMODE(64).
PKA Public Key Extract	HCR7780	**Changed:** Elliptic Curve Cryptography (ECC) support.
PKDS Key Record Create	HCR7780	**Changed:** Elliptic Curve Cryptography (ECC) support.
PKDS Key Record Delete	HCR7780	**Changed:** Elliptic Curve Cryptography (ECC) support.

Table 4. Summary of new and changed ICSF callable services (continued)

Callable service	Release	Description
PKDS Key Record Read	HCR7780	**Changed:** • Elliptic Curve Cryptography (ECC) support. • Support for invocation in AMODE(64).
PKDS Key Record Write	HCR7780	**Changed:** • Elliptic Curve Cryptography (ECC) support. • Support for invocation in AMODE(64).
Prohibit Export	HCR7780	**Changed:** Support for invocation in AMODE(64).
Prohibit Export Extended	HCR7780	**Changed:** Support for invocation in AMODE(64).
Remote Key Export	HCR7780	**Changed:** Support for invocation in AMODE(64).
Restrict Key Attribute	HCR7780	**New:** Support for CCA key management of HMAC keys.
Secure Key Import	HCR7780	**Changed:** Support for invocation in AMODE(64).
Secure Key Import2	HCR7780	**New:** Support for CCA key management of HMAC keys.
Secure Messaging for Keys	HCR7780	**Changed:** Support for invocation in AMODE(64).
Secure Messaging for PINS	HCR7780	**Changed:** Support for invocation in AMODE(64).
SET Block Compose	HCR7780	**Changed:** Support for invocation in AMODE(64).
SET Block Decompose	HCR7780	**Changed:** Support for invocation in AMODE(64).
Symmetric Key Decipher	HCR7780	**Changed:** Additional modes of operation for protecting data.
Symmetric Key Encipher	HCR7780	**Changed:** Additional modes of operation for protecting data.
Symmetric Key Export	HCR7780	**Changed:** Support for CCA key management of HMAC keys.
Symmetric Key Generate	HCR7780	**Changed:** • Support for invocation in AMODE(64). • New rule array keywords to support enhanced key wrapping method.
Symmetric Key Import	HCR7780	**Changed:** New rule array keywords to support enhanced key wrapping method.
Symmetric Key Import2	HCR7780	**New:** Support for CCA key management of HMAC keys.
Transaction Validation	HCR7780	**Changed:** Support for invocation in AMODE(64).
Transform CDMF Key	HCR7780	**Changed:** Support for invocation in AMODE(64).
Trusted Block Create	HCR7780	**Changed:** Support for invocation in AMODE(64).
User Derived Key	HCR7780	**Changed:** Support for invocation in AMODE(64).
VISA CVV Service Generate	HCR7780	**Changed:** Support for invocation in AMODE(64).
VISA CVV Service Verify	HCR7780	**Changed:** Support for invocation in AMODE(64).
PKCS #11 Derive key	HCR7770	**New:** Support for PKCS #11.
PKCS #11 Derive multiple keys	HCR7770	**New:** Support for PKCS #11.
PKCS #11 Generate HMAC	HCR7770	**New:** Support for PKCS #11.
PKCS #11 Generate key pair	HCR7770	**New:** Support for PKCS #11.
PKCS #11 Generate secret key	HCR7770	**New:** Support for PKCS #11.
PKCS #11 One-way hash generate	HCR7770	**New:** Support for PKCS #11.

Table 4. Summary of new and changed ICSF callable services (continued)

Callable service	Release	Description
PKCS #11 Private key sign	HCR7770	**New:** Support for PKCS #11.
PKCS #11 Pseudo-random function	HCR7770	**New:** Support for PKCS #11.
PKCS #11 Public key verify	HCR7770	**New:** Support for PKCS #11.
PKCS #11 Secret key decrypt	HCR7770	**New:** Support for PKCS #11.
PKCS #11 Secret key encrypt	HCR7770	**New:** Support for PKCS #11.
PKCS #11 Unwrap key	HCR7770	**New:** Support for PKCS #11.
PKCS #11 Verify HMAC	HCR7770	**New:** Support for PKCS #11.
PKCS #11 Wrap key	HCR7770	**New:** Support for PKCS #11.
PKA Key Translate	HCR7770	**New:** Support for RSA private key export.
PKA Key Generate	HCR7770	**Changed:** Support for RSA private key export.
PKA Key Token Build	HCR7770	**Changed:** Support for RSA private key export.
Symmetric Key Export	HCR7770	**Changed:** Support for invocation in AMODE(64).
Symmetric Key Import	HCR7770	**Changed:** Support for invocation in AMODE(64).
Symmetric Key Encipher	HCR7770	**Changed:** Support an encrypted key in the CKDS.
Symmetric Key Decipher	HCR7770	**Changed:** Support an encrypted key in the CKDS.
ICSF Query Algorithm	HCR7751	**New:** HCR7751
Symmetric Algorithm Decipher	HCR7751	**New:** Supports secure key AES
Symmetric Algorithm Encipher	HCR7751	**New:** Supports secure key AES
Symmetric MAC Generate	HCR7751	**New:** Supports IPv6
Symmetric MAC Verify	HCR7751	**New:** Supports IPv6
ICSF Query Function	HCR7751	**Changed:** Supports secure key AES
Key Generate	HCR7751	**Changed:** Supports secure key AES
Key Record Create	HCR7751	**Changed:** Supports secure key AES
Key Record Delete	HCR7751	**Changed:** Supports secure key AES
Key Record Read	HCR7751	**Changed:** Supports secure key AES
Key Record Write	HCR7751	**Changed:** Supports secure key AES
Key Test	HCR7751	**Changed:** Supports secure key AES
Key Token Build	HCR7751	**Changed:** Supports secure key AES
Multiple Clear Key Import	HCR7751	**Changed:** Supports secure key AES
Multiple Secure Key Import	HCR7751	**Changed:** Supports secure key AES
Symmetric Key Export	HCR7751	**Changed:** Supports secure key AES
Symmetric Key Generate	HCR7751	**Changed:** Supports secure key AES
Symmetric Key Import	HCR7751	**Changed:** Supports secure key AES
VISA CVV Service Generate	HCR7751	**Changed:** Supports PAN-14, PAN-15, PAN-17 and PAN-18
VISA CVV Service Verify	HCR7751	**Changed:** Supports PAN-14, PAN-15, PAN-17 and PAN-18

Ensure the expected master key support is available

In versions of ICSF prior to FMID HCR7780, in order for a coprocessor to become active, a DES master key needed to be set on the coprocessor. Once the coprocessor was active, DES master key support, and the support of any other master key (an AES master key or an asymetric master key) set on the coprocessor, would then be available.

Starting with ICSF FMID HCR7780, the activation procedure for non-CCF systems is designed to maximize the number of active coprocessors by selecting the set of master keys that are available on the majority of coprocessors. A DES master key is no longer required in order for a coprocessor to become active. Instead, any one of four master keys – the DES master key, the AES master key, the RSA master key (which in earlier releases was called the asymetric master key), or the ECC master key – is enough for a coprocessor to become active. However, because the goal is to select the combination of master keys that will maximize the number of active coprocessors, if a certain master key is not set on all the same coprocessors, that master key support will not be available. Before you use ICSF FMID HCR7780, you must understand the new method of coprocessor activation in order to ensure that the expected master keys will be available. You may need to set master keys on additional coprocessors to ensure the same master key support you had in earlier versions of ICSF is available in ICSF FMID HCR7780.

To further illustrate the change in coprocessor activation introduced in ICSF FMID HCR7780, let's say you have the following four coprocessors with these master keys set (as indicated by the letter “Y”).

Table 5. Coprocessor activation example

	E00	E01	E02	E03
DES	Y		Y	
AES	Y	Y	Y	Y
RSA (ASYM)	Y		Y	

In releases prior to ICSF FMID HCR7780, a DES master key was required in order for a coprocessor to become active. In our example, coprocessors E00 and E02 have the DES master key set, so they become active. The DES master key is available as well as the AES and RSA (ASYM) master keys which are set on the the E00 and E02 coprocessors. Coprocessors E01 and E03 do not have a DES master key set, and so are not active.

Starting with ICSF FMID HCR7780, any master key is enough for a coprocessor to become active and the activation procedure tries to maximize the number of active coprocessors. The AES master key is set on all four of these coprocessors, so all four of the coprocessors become active. However, since the DES and RSA master keys are not set on all four of the coprocessors, DES and RSA support is not available. As coprocessor master keys are set or changed, however, additional function may become available. In this case, setting the DES and RSA master keys on coprocessors E01 and E03 would make the DES and RSA support available.

ECC master key support is based on the existence of CEX3C coprocessors. If a mixture of CEX3C coprocessors and older coprocessors exist on a system, then ECC support will be based solely on the state of the CEX3C coprocessors. For example, let's change our example so that two of the coprocessors are CEX3C coprocessors (as identified by the G prefix in the name) that have the ECC master key set.

Table 6. Coprocessor activation example (ECC support based only on CEX3C coprocessors)

	G00	E01	G02	E03
DES	Y		Y	
AES	Y	Y	Y	Y
RSA (ASYM)	Y		Y	
ECC	Y		Y	

In this example, the AES master key is still the one set on the majority of coprocessors and makes the coprocessors active. The DES and RSA master keys are not set on all the coprocessors, and so DES and RSA support is not available. The ECC master key cannot be set on all coprocessors (because it is not available on the CEX2C). However, unlike the DES and RSA support, ECC support is still available. This is because the ECC master key is set on all CEX3C coprocessors.

After migrating from an ICSF release prior to HCR7780 to HCR7780 or later, verify the expected master key support is available.

To verify the master key support is available, select option 1, COPROCESSOR MGMT, from the ICSF Primary menu. This will display the coprocessor management panel, which shows which coprocessors are active, and the state of the master keys for coprocessors.

To ensure the master key support that you expect is available, follow these steps

1. To ensure AES support, set the AES master key on each CEX2C and CEX3C.
2. To ensure DES support, set the DES master key on each CEX2C, CEX3C, and PCIXCC.
3. To ensure RSA support, set the RSA master key on each CEX2C, CEX3C, and PCIXCC.
4. To ensure ECC support, set the ECC master key on each CEX3C.

Notes:

1. AES support is not available on PCIXCC.
2. ECC support is only available on a CEX3C on a z196.

Ensure that the CSFPUTIL utility is not used to initialize a PKDS

ICSF provides a utility program, CSFPUTIL, that performs certain functions that can also be performed using the administrator's panels. In releases of ICSF prior to FMID HCR7780, you could use the CSFPUTIL utility program to initialize a PKDS, reencipher a PKDS, and refresh the in-storage copy of the PKDS. You can still use the CSFPUTIL utility to reencipher or refresh a PKDS. However, starting with FMID HCR7780, the CSFPUTIL utility program no longer supports the function to initialize a PKDS. Instead, the ICSF panels must be used to initialize a PKDS.

Because of this change, jobs that call the CSFPUTIL utility with the INITPKDS option will no longer initialize a PKDS, and return code 4 (which indicates that supplied parameters are incorrect) will be returned. If migrating from a release of ICSF prior to HCR7780 to HRC7780 or later, you should, prior to the first IPL, make sure no jobs call CSFPUTIL with the INITPKDS option. Use the administrator panels to initialize the PKDS instead.

For more information in initializing the PKDS and on the CSFPUTIL utility, refer to *z/OS Cryptographic Services ICSF Administrator's Guide*.

Modify ICSF startup procedure to run new startup program

In ICSF FMID HCR7770, the CSFMMAIN program (which started ICSF in earlier releases) has been replaced by the new startup program CSFINIT. If migrating to HCR7770 (or later) from an earlier release, you must modify your ICSF startup procedure to run this new program. If you do not modify your ICSF startup procedure, ICSF will not start.

Member CSF in SYS1.SAMPLIB contains sample JCL code for an ICSF startup procedure to identifies the new startup program. This sample is described in "Steps to create the Installation Options Data Set" on page 26. You can also update an existing startup procedure for ICSF by doing the following.

1. Find the job step that identifies the ICSF startup program (CSFMMAIN) that was used in earlier releases. For example:

   ```
   CSFSTART EXEC PGM=CSFMMAIN,REGION=0M,TIME=1440
   ```

2. Modify the PGM parameter on this EXEC statement to identify the new startup program (CSFINIT):

   ```
   CSFSTART EXEC PGM=CSFINIT,REGION=0M,TIME=1440
   ```

3. Save your changes to the startup procedure.

If your ICSF startup procedure specifies a CSFLIST dataset, you can remove this specification. The HCR7770 level of ICSF does not utilize a CSFLIST dataset, and will ignore it if specified. Refer to *z/OS Cryptographic Services ICSF Messages* for an explanation of how CSFLIST dataset information is handled for the HCR7770 release level.

Ensure PKCS #11 applications call C_Finalize() prior to calling dlclose()

A PKCS #11 application initializes the environment by calling dlopen() to load the PKCS #11 DLL into storage, and then calling C_Initialize(). Later, when processing is complete, the application terminates processing by calling C_Finalize(), and then calling dlclose(). Reinitialization, if desired, can be achieved by calling dlopen() and C_Initialize() a second time.

In releases prior to HCR7770, z/OS PKCS #11 allowed an application to implicitly finalize the environment by calling dlclose() without first calling C_Finalize(). Starting in HCR7770, this will no longer be supported. If an application does not call C_Finalize() prior to calling dlclose(), a subsequent attempt to re-initialize PKCS #11 by calling C_Initialize() will result in error CKR_FUNCTION_FAILED being returned.

PKCS #11 application developers should scan their source code for the following sequence of calls: dlopen(), C_Initialize(), *processing functions*, dlclose(), dlopen(), C_Initialize(). Change all such sequences to insert a call to C_Finalize() before the call to dlclose().

ICSF Key Data Sets

CKDS

There are two formats of the CKDS: a fixed length record (supported by all releases of ICSF) and a new, variable length record (supported by HCR7780 and later

releases). If you are going to store the HMAC or variable-length AES keys in the CKDS, variable length format is required. See “Migrating to the variable length CKDS” for more information.

When sharing a CKDS with CCF systems and z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, and z196, the CKDS must be created on a CCF system.

Once new key types are added to the CKDS, these considerations apply when sharing the CKDS:

- once keys with non-CCF control vectors are added to the CKDS, a CKDS reencipher operation must be invoked from a system that has a PCICC, PCIXCC, CEX2C, or CEX3C installed.
- when clear DES or AES keys are added to the CKDS, RACF-protect all clear DES and AES keys by label name on all systems sharing the CKDS.
- generation of a clear DES or AES key using KGUP requires a PCIXCC, CEX2C, or CEX3C.

If you have no coprocessor, you can initialize the CKDS for use with clear AES and DES data keys. This CKDS cannot be used on a system with cryptographic coprocessors.

ICSF releases before HCR7751 do not support secure AES keys and require APAR OA26579 for toleration.

Release HCR7780 introduced the enhanced key wrapping method for DES key tokens. ICSF releases before HCR7780 do not support enhanced key wrapping and require a toleration APAR.

A CKDS with tokens wrapped with the enhanced method can only be reenciphered on a system running release HCR7780

Note: The CKDS exits (single-record, read-write and retrieval) are not enabled for the variable-length record format of the CKDS. See Chapter 5, “Installation Exits,” on page 111 for more information.

Migrating to the variable length CKDS: If HMAC or variable-length AES keys are to be stored in the CKDS, any existing CKDS must be converted to the new variable length record format. ICSF provides a conversion utility to do this. ICSF provides a utility program, CSFCNV2, that will convert a CKDS to the variable length format. Refer to Chapter 7, “Converting a CKDS from fixed length to variable length record format,” on page 169 for more information.

There is no reason to migrate a variable length record CKDS if your applications are not using HMAC keys. In the future, other symmetric keys may required the variable length record CKDS be used. You can migrate to the variable length record at any time.

Note: All systems that will share a CKDS with the variable length record format must be running ICSF HCR7780 or later.

To migrate to a variable length CKDS:

1. Install the HC7780 or later release of ICSF on all systems that will share the CKDS.
2. Allocate a new CKDS with the variable length record format. The new CKDS should be large enough to hold all key in the current CKDS.

3. Disable dynamic CKDS updates on all systems.
4. Run the CKDS Conversion2 utility to convert the existing CKDS records to the new record format
5. Refresh the new CKDS on all systems that are sharing the CKDS
6. Enable dynamic CKDS updates on all systems

PKDS

The PKDS must be initialized. Support to INITIALIZE PKDS, REENCIPHER PKDS and REFRESH PKDS is available in the Master Key Management Panels. The CSFPUTIL utility also provides support to reencipher and refresh the PKDS.

For information on managing and sharing the PKDS in a sysplex environment, see *z/OS Cryptographic Services ICSF Administrator's Guide*, SA22-7521.

The process of reenciphering the PKDS has changed for z196 systems with CEX3C coprocessors and the Sep. 2011 licensed internal code (LIC). See "Changing the RSA master key."

For additional information on ME and CRT tokens, see Appendix A, "Diagnosis Reference Information," on page 193.

TKDS

Beginning with HCR7740, these TKDS key management panels are available from the ICSF Utility panel:

- Listing tokens
- Creating a token
- Deleting a token
- Listing the objects of a token
- Viewing the attributes of an object
- Updating the attributes of an object
- Deleting an object

Access authorization of the new callable services will be determined via SAF calls. No support will be provided for invocation of an installation security exit for these new services. The CSFSERV class controls access to the ICSF PKCS #11 callable services.

Key Tokens

- On the z990, z890, z9 EC, z9 BC, z10 EC, z10 BC, and z196, all RSA internal tokens are usable on PCIXCC, CEX2C, or CEX3C if they are encrypted under the current RSA-MK.
- Existing DES internal key tokens can be used on all cryptographic coprocessors: CCF, PCICC, PCIXCC, CEX2C, or CEX3C
- AES internal key tokens can only be used on a CEX2C or CEX3C.
- ECC internal key tokens can only be used on a CEX3C.

Changing the RSA master key

The process to reencipher the PKDS and change the RSA master key has changed for z196 systems with CEX3C coprocessors and the Sep. 2011 licensed internal code (LIC). For these systems, RSA master key change will be processed in the same manner as master key change for the DES, AES and ECC master keys.

This is the current procedure for changing the RSA master key. For systems without CEX3C coprocessors and the Sep. 2011 LIC, this procedure has not changed.

1. Disable dynamic PKDS updates control (recommended)
2. Disable PKA callable services control
3. Load the new RSA master key
   - TKE: load and set RSA master key
   - ICSF panels: loading the final key part causes the current master key to be set
4. Reencipher the PKDS (old to current master key)
5. Refresh the reenciphered PKDS
6. Enable PKA callable services control
7. Enable dynamic PKDS updates control

For systems with CEX3C coprocessors (and the Sep. 2011 LIC) with the RSA master key loaded, this will be the new procedure for changing the RSA master key. See *z/OS Cryptographic Services ICSF Administrator's Guide* for more information.

1. Disable dynamic PKDS updates control (recommended)
2. Load the new RSA master key (TKE or ICSF panels)
3. Reencipher the PKDS (current to new master key)
4. Change the RSA master key (the current master key is set and the reenciphered PKDS becomes active PKDS)
5. Enable dynamic PKDS updates control

Note: When the new RSA master key change process is used:

- The PKA callable services control will not appear on the Administrator Control Functions panel.
- The availability of callable services that required the RSA master key is controlled by the state of the RSA master key. When the RSA master key is active (the master key verification pattern in the PKDS matches the verification pattern of the current RSA master key), RSA callable service are available. Message CSFM130I will be issued.
- The RSA master key cannot be set from the TKE workstation.

Installation Options Data Set

- CKTAUTH - decides if authentication will be performed for every CKDS record read from DASD. With a large CKDS, CKTAUTH(YES) could cause an impact to performance.
- PKDSCACHE - no longer supported, but tolerated.
- COMPENC - no longer supported, but tolerated.
- DOMAIN - this parameter is optional. It is required if more than one domain is specified as the usage domain on the PR/SM panels or if running in native mode.

  On a z990 or z890, if a PCI Cryptographic Accelerator, PCIXCC, CEX2C, or CEX3C is not available, or on a z9 EC, z9 BC, z10 EC, z10 BC, or z196 if a CEX2A, CEX2C, CEX3A, or CEX3C is not available, the DOMAIN parameter is not required. If a PCICA, PCIXCC, CEX2A, CEX2C, CEX3A, or CEX3C is available, the DOMAIN parameter is required if more than one domain is specified as the usage domain on the PR/SM panels. If only one usage domain is assigned to the LPAR, the DOMAIN parameter is optional.

- SYSPLEXCKDS - this parameter is optional. It specifies whether this system, if a member of a multi-system sysplex using a shared CKDS, should participate in XCF signalling in order to maintain consistency of the CKDS data across the sysplex.
- SYSPLEXPKDS - this parameter is optional. It specifies whether this system, if a member of a multi-system sysplex using a shared PKDS, should participate in XCF signalling in order to maintain consistency of the PKDS data across the sysplex.
- SYSPLEXTKDS - this parameter is optional. It specifies whether this system, if a member of a multi-system sysplex using a shared TKDS, should participate in XCF signalling in order to maintain consistency of the TKDS data across the sysplex.
- TKDSN - this parameter is the name of an existing TKDS or an empty VSAM data set to be used as the TKDS.
- TRACEENTRY - the supported boundary values for this parameter have been changed to a minimum of 10000 and a maximum of 500000. Values outside of this range are tolerated, but automatically adjusted to 10000.
- DEFAULTWRAP – this parameter is optional. The parameter defines the wrapping method of DES key tokens in ICSF. The ORIGINAL method is ECB for DES keys. The ENHANCED method is ANSI X9.24 compliant. The key value is bundled with other data and encrypted using the CBC mode. The default wrapping method is defined independently for internal and external tokens

Function Restrictions

Retained keys are RSA private keys that are stored in a cryptographic coprocessor instead of in the public key storage data set. This change does not affect retained keys that you are currently using, that is, keys that are stored on the cryptographic coprocessor. However, starting with ICSF HCR7750, the ICSF services do no allow you to store in a cryptographic coprocessor RSA keys intended for key management use. Your applications can continue to store in the cryptographic coprocessor RSA private keys intended for signature usage. The modulus length of these private keys is limited to 2048-bits. ICSF HCR7750 introduces 4096-bit modulus RSA keys support.

The 2048-bit RSA keys may have an public exponent, e, in the range of $1<e<2^{2048}$. and e must be odd. The RSA public key exponents for 2049-bit to 4096-bit RSA keys are restricted to the values 3 and 65537

ICSF delivers the migration check support for PKDS compatibility with 4096-bit RSA key support for HCR7731 and z/OS V1.9 (ICSF release HCR7740) only, but not for any releases earlier or later. This check support also requires that its delivery PTF pre-req the PTF for the PKDS 4096-bit key toleration - APAR OA21807. Doing so ensures that you have enabled the opportunity for a sufficiently allocated PKDS, and avoids the problem where you attempt to properly allocate a PKDS for 4096-bit RSA keys, but then find the current ICSF service level fails to support it.

CICS Attachment Facility

If you have the CICS Attachment Facility (CICS 4.1 or higher) installed and you specify your own CICS wait list data set, you need to modify the wait list data set to include the new callable services.

On a z900 machine, modify and include:

- HCR7780: CSFKRC2, CSFKRW2

- HCR7770: CSF1DMK, CSF1DVK, CSF1GKP, CSF1GSK, CSF1PKS, CSF1PKV, CSF1SAV, CSF1SKE, CSF1TRC, CSF1TRD, CSF1UWK, CSF1WPK, CSFTBC, CSFRKX
- HCR7751, HCR7750, HCR7740: CSF1GKP, CSF1GSK, CSF1PKS, CSF1PKV, CSF1SAV, CSF1TRC, CSF1TRD, CSF1UWK, CSF1WPK, CSFTBC, CSFRKX
- HCR7731: CSFTBC, CSFRKX

Note: If no Wait List is specified on a z900, the default wait list will be used. (See sample CSFWTL00 for the contents of the default wait list for z900.)

On a z990 or z890 (or later) class machine, modify and include:

- HCR7790: CSFEDH, CSFT31X, CSFT31I, CSFCKC
- HCR7780: CSFHMG, CSFHMG1, CSFHMV, CSFHMV1, CSFKGN2, CSFKPI2, CSFKTR2, CSFKYT2, CSFRKA, CSFSKI2, CSFSYI2, CSFKRC2, CSFKRW2
- HCR7770: CSNBSYD, CSNBSYD1, CSNBSYE, CSNBSYE1, CSFPKT, CSF1DMK, CSF1DVK, CSF1SKD, CSF1SKE, CSF1HMG, CSF1HMV, CSF1OWH, CSF1PRF, CSNBSAD, CSNBSAD1, CSNBSAE, CSNBSAE1, CSFRNGL, CSF1GKP, CSF1GSK, CSF1PKS, CSF1PKV, CSF1SAV, CSF1TRC, CSF1TRD, CSF1UWK, CSF1WPK, CSFTBC, CSFRKX
- HCR7751: CSNBSAD, CSNBSAD1, CSNBSAE, CSNBSAE1, CSFRNGL, CSF1GKP, CSF1GSK, CSF1PKS, CSF1PKV, CSF1SAV, CSF1TRC, CSF1TRD, CSF1UWK, CSF1WPK, CSFTBC, CSFRKX
- HCR7750: CSFRNGL, CSF1GKP, CSF1GSK, CSF1PKS, CSF1PKV, CSF1SAV, CSF1TRC, CSF1TRD, CSF1UWK, CSF1WPK, CSFTBC, CSFRKX
- HCR7740: CSF1GKP, CSF1GSK, CSF1PKS, CSF1PKV, CSF1SAV, CSF1TRC, CSF1TRD, CSF1UWK, CSF1WPK, CSFTBC, CSFRKX
- HCR7731: CSFTBC, CSFRKX

Note: If no Wait List is specified on a z990, z890, z9 EC, z9 BC, z10 EC and z10 BC the default wait list will be used. (See sample CSFWTL01 for the contents of the default wait list for z990, z890, z9 EC, z9 BC, z10 EC and z10 BC.)

Dynamic LPA Load

Prior to HCR7740, ICSF loaded the pre-PC routines, CICS related routines and other modules which must reside in common storage into below-the-line SQA.

As of HCR7740, ICSF uses dynamic LPA to load these modules into above-the-line ECSA. The dynamic LPA load will occur the first time that ICSF is started within an IPL, and the modules will persist across subsequent restarts of ICSF.

Special Secure Mode

Use of some ICSF services requires that ICSF be in special secure mode: CSNBPGN, CSNBSKI, CSNBSKI2, CSNBSKM, and CSNDSYG with the IM keyword (CCF only).

Note: On a CCF system, if a PCICC is available and the modulus bit length of the RSA public key is greater than or equal to 512 bits, than special secure mode is not required for CSNDSYG IM form.

Resource Manager Interface (RMF)

Support to enable RMF to provide performance measurements on these selected ICSF services and functions that use Direct Access Crypto (DAC) CCF instructions is available. On a z990, z890,z9 EC, z9 BC, z10 EC, z10 BC, and z196 with a PCIXCC, CEX2C, or CEX3C, the measurements refer to these services processing on PCIXCCs, CEX2Cs, or CEX3Cs except for one-way hash. One-way hash is processed on CPACF.

- Encipher (CSNBENC)
- Decipher (CSNBDEC)
- MAC Generate (CSNBMGN)
- MAC Verify (CSNBMVR)
- One-Way Hash (CSNBOWH)
- PIN Translate (CSNBPTR)
- Symmetric Algorithm Decipher (CSNBSAD)
- Symmetric Algorithm Encipher (CSNBSAE)
- PIN Verify (CSNBPVR)

System Abend Codes

A complete list of the reason codes for the ICSF abend (X'18F') is contained in *z/OS MVS System Codes* which is published on release boundaries. As a migration aid for HCR7790, which is not on a release boundary, new codes for FMID HCR7790 are listed here.

An 18F code indicates an abend from ICSF. Reason codes for application services routines:

Code Hex (Dec)	Meaning
2D (45)	A cryptographic coprocessor returned a bad condition code. The coprocessor has been fenced off.
6F (111)	Error from ATTACH of the CSFPLMWT task.
F3 (243)	Error from IXCQUERY request in CSFPLCMD module.
F4 (244)	Error from IXCMSGO request in CSFPLMSR module.
F5 (245)	Error from IEAVPSE2 request in CSFPLMSR module.
FA (250)	Error determining sysplex KDS cluster member list in CSFPLCMD.
19A (410)	XCF message is too big for internal ICSF buffer.
19B (411)	An IXCMSGI request failure occurred receiving an XCF message.
45A (1114)	The CSFPLMWT task received a bad RC from IEAVPSE2.
45B (1115)	The CSFPLMRT task received inconsistent internal control information.
45C (1116)	The CSFSMBTM module received inconsistent internal control information.

45D (1117)	A pause failure occurred in the CSFPLPSC module.
45E (1118)	An IXCMSGO request failure occurred sending an XCF message.
45F (1119)	An XCF message exit error occurred during ICSF sysplex processing.
460 (1120)	A sequence number error occurred during ICSF sysplex processing.
461 (1121)	A cache update failure occurred during ICSF sysplex processing.
462 (1122)	The CSFKSIDC module was called with an incorrect function code.
463 (1123)	The CSFKSIIO module was called with an invalid IO function request.
464 (1124)	The CSFKSIIO module ran out of dynamic storage.
465 (1125)	ICSF forced termination.
466 (1126)	An error occurred during ATTACH of the CSFMISTP task.

SMF Records

SMF records are documented in *z/OS MVS System Management Facilities (SMF)* and published on release boundaries. As a migration aid for ICSF FMIDs, which are often made available as Web Deliverables that are not on a z/OS release boundary, SMF record information for ICSF is also documented in Appendix B, "ICSF SMF Records," on page 283. Refer there for information on SMF records.

TKE Workstation

The Trusted Key Entry (TKE) workstation is available on the IBM @server zSeries 990 IBM @server zSeries 890, z9 EC, z9 BC, z10 EC, z10 BC, and z196. It can also be used to provide key management on the IBM @server zSeries 900.

Refer to *z/OS Cryptographic Services ICSF TKE Workstation User's Guide* for more information.

TKE Version 3.1 and Access to Callable Services

Access to services that are executed on the PCI Cryptographic Coprocessor is through Access Control Points in the DEFAULT Role. To execute callable services on the PCI Cryptographic Coprocessor, access control points must be enabled for each service in the DEFAULT Role. For systems that do not use the optional TKE Workstation, all access control points (current and new) are enabled in the DEFAULT Role with the appropriate microcode level on the PCI Cryptographic Coprocessor.

New TKE users and non-TKE users have all* access control points enabled. This is also true for brand new TKE V3.1 users (not converting from TKE V3.0).

Note: *Access control point DKYGENKY-DALL is always disabled in the DEFAULT Role for all customers (TKE and Non-TKE). A TKE Workstation is required to enable this access control point for the Diversified Key Generate service.

All of the mentioned components are required for complete access control point support.

Access to services which execute on the Cryptographic Coprocessor Feature is through SAF. Disablement through SAF is sufficient to prevent execution of a service by the Cryptographic Coprocessor Feature, the PCI Cryptographic Coprocessor, the PCI X Cryptographic Coprocessor or the Crypto Express2 Coprocessor. For functions which can be executed on the PCI Cryptographic Coprocessor or PCI X Cryptographic Coprocessor/Crypto Express2 Coprocessor, enablement of the function requires that the function be enabled through SAF and through the access control point in the DEFAULT Role. For additional details, see "TKE Version 4.x and Higher and Access to Callable Services."

These are access control points for PCICCs.

TKE Version 4.x and Higher and Access to Callable Services

Access to services that are executed on the PCI X Cryptographic Coprocessor or Crypto Express2 Coprocessor is through Access Control Points in the DEFAULT Role. To execute callable services on the PCIXCC/CEX2C, access control points must be enabled for each service in the DEFAULT Role. For systems that do not use the optional TKE Workstation, all access control points (current and new) are enabled in the DEFAULT Role with the appropriate microcode level on the PCIXCC/CEX2C.

New TKE users and non-TKE users have all* access control points enabled. If you are migrating from TKE V4.0 or TKE V4.1 or TKE V4.2 to TKE V5.x and have a PCIXCC/CEX2C, all your current access control points will remain the same and any new applicable access control points will not be enabled.

Note: *Access control points DKYGENKY-DALL and DSG ZERO-PAD unrestricted hash length and PTR enhanced PIN security are always disabled in the DEFAULT role for all customers (TKE and Non-TKE). A TKE Workstation is required to enable these access control points.

TKE Enablement from the Support Element

On z890 or systems running with May 2004 or higher version of Licensed Internal Code or a z9 EC, z9 BC and IBM System z10 Enterprise Class system running with MCL 029 Stream J12220 or higher version of Licensed Internal Code, you must enable TKE commands on each PCIXCC, CEX2C, or CEX3C card from the support element. This is true for new TKE users and those upgrading to this level of LIC. See *Support Element Operations Guide*, SC28-6820 and *z/OS Cryptographic Services ICSF TKE Workstation User's Guide*, SA23-2211 for more information.

Migrating from the IBM @server zSeries 900

Callable Services

These services are only available on the IBM @server zSeries 900. These services are not supported with a PCIXCC/CEX2C installed.

- ANSI X9.17 EDC Generate (CSNAEGN)
- ANSI X9.17 Key Export (CSNAKEX)
- ANSI X9.17 Key Import (CSNAKIM)
- ANSI X9.17 Key Translate (CSNAKTR)
- ANSI X9.17 Transport Key Partial Notarize (CSNAKTR)
- Ciphertext Translate (CSNBCTT)
- PKSC Interface Service (CSFPKSC)
- Transform CDMF Key (CSNBTCK)

- User Derived Key (CSFUDK)

A Health Check, ICSFMIG_DEPRECATED_SERV_WARNINGS, has been provided to detect the use of these services. You should migrate away from the use of these services, because support is being removed in later releases of ICSF. You should investigate applications using these services, and determine the appropriate actions to remove or replace them.

Functions Not Supported

This topic lists functions not supported with a PCIXCC/CEX2C installed.

1. There is no KMMK (key management master key).
2. The Commercial Data Masking Facility (CDMF) is no longer supported. The CDMF keyword on KGUP control statements and panels are no longer supported.
3. The Public Key Algorithm Digital Signature Standard is not supported. This affects callable services CSNDPKG, CSNDPKI, CSNDDSG, and CSNDDSV.
4. The PBVC keyword is not supported on a PCIXCC/CEX2C. This affects callable services Clear PIN Generate Alternate (CSNBCPA), PIN Translate (CSNBPTR) and PIN Verify (CSNBPVR).

Setup Considerations

This topic lists setup changes that should be considered when installing a PCIXCC/CEX2C.

Consideration should be given to:

1. CICS wait list should be updated for services now executing on PCIXCCs/CEX2Cs. The sample CICS wait list, CSFWTL01, supplied by IBM includes these services and can be used as a reference.
2. PKDS initialization is required.
3. New options data set keyword CKTAUTH.
4. A CKDS initialized on a z990, z890, z9 EC, z9 BC, z10 EC, z10 BC, and z196 cannot be used on CCF systems. However, a CKDS initialized on z900 can be used on z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, and z196.
5. If sharing a PKDS with a PCICC and PCIXCC/CEX2C, delete the PKDS records for labelnames of retained keys on PCICCs no longer in use.
6. Customers who run CSFEUTIL to setup ICSF for automated electronic delivery process no longer need to execute CSFEUTIL on a z990, z890, z9 EC, z9 BC, or z196 system. SHA-1 is available on z990, z890, z9 EC, z9 BC, z10 EC, z10 BC, and z196 without entering ICSF master keys.
7. Options data set keyword SYSPLEXCKDS.
8. Options data set keyword SYSPLEXTKDS.
9. Options data set keyword SYSPLEXPKDS.
10. The PKDSCACHE function is replaced with an in-storage copy of the PKDS. No installation option is required.

Programming Considerations

This topic lists programming changes that should be considered when installing a PCIXCC/CEX2C.

Consideration should be given to:

1. The DATAC key type can not be used on the IBM @server zSeries 900.

2. The PIN block format checking on PCIXCC/CEX2C is more rigorous than with a CCF.

   For CSNBPVR, CSNBPTR and CSNBCPA services, the input PIN block must have the correct format as specified in the PIN Profile parameter. On a CCF system, the PIN block format checking is incomplete.

   For example, the REFORMAT processing mode of PIN Translate (CSNBPTR) may now fail on a PCIXCC/CEX2C when it was previously successful on a CCF. On a CCF, if input to the PIN verify service (CSNBPVR) is a malformed encrypted PIN block, the service will fail with return code 4, reason code 3028 (verification failed); on a PCIXCC/CEX2C, the service may fail with return code 8 and some appropriate reason code for invalid PIN format.
3. 512 to 2048 bit modulus for RSA keys is supported in all PKA services except SET services (Set Block Compose and Set Block Decompose).
4. All CCF functions are now executed on the PCIXCC/CEX2C. This may cause some impact on the performance of customer applications.
5. Reason codes from the PCIXCC/CEX2C may be different from previous cryptographic hardware.
6. With PCIXCCs/CEX2Cs, the requirement that caller must be in supervisor state to use NOCV tokens is lifted for the CKDS Key Record Write (CSNBKRW) service.
7. The z/OS SCHEDULE and IEAMSCHD macros are used to schedule SRBs. On the IBM @server zSeries 990, IBM @server zSeries 890, IBM System Enterprise Class or IBM System Business Class, since there are no CCFs on the system, applications should delete FEATURE=CRYPTO on the SCHEDULE and IEAMSCHD macros or the SRB being scheduled will not run.
8. External tokens that are export prohibited are imported differently on a z990, z890, z9 EC, z9 BC, z10 EC, z10 BC, and z196 system with PCIXCCs/CEX2C/CEX3Cs. The imported internal token will have the same control vector as the external token with export prohibited. These tokens will only be usable on a z990, z890, z9 EC, z9 BC, z10 EC, z10 BC, and z196 with a PCIXCC/CEX2C or on CCF systems with PCICCs. On previous hardware (CCF systems) the imported internal token had a control vector that allowed export, and export prohibition was enforced by the export flag in the token.
9. Prohibit Export service can now be used for MAC and MACVER keys.
10. New rule array keyword TDES-MAC added to the MAC Generate and MAC Verify services.
11. New rule array keywords, CFB and PKCS-PAD added to the Symmetric Key Decipher and Symmetric Key Encipher services.
12. A RACF check is added to the Key Generation Utility (CSFKGUP).
13. The CSFKGUP utility exit control block has been changed for AES. See Chapter 5, “Installation Exits,” on page 111 for the new format.

Migrating from 4753-HSP

ICSF provides key management callable services that are identical to the 4753-HSP verbs of the same name. Key management applications that are developed for the 4753-HSP and use these common verbs can be run on OS/390 ICSF or z/OS ICSF without reassembly. You will, however, need to relink them.

If your installation is currently using the 4753-HSP and you are migrating to OS/390 ICSF or z/OS ICSF, consider:

- **4753-HSP cryptographic key storage**

  Internal key tokens for ICSF and the 4753-HSP are not interchangeable. Key token migration for the 4753 exists through the optional TKE Version 3 Workstation for the S/390 G5, G6, and z900 servers and TKE Version 4 or higher for z900, z890, z990, z9 EC, z9 BC and IBM System z10 Enterprise Class servers. TKE Version 3, Version 4, and Version 5 supply a 4753 Migration Utility. It allows you to migrate internal DES key tokens from the 4753 to ICSF. Key exchange between the two systems is through the external key token. To migrate keys from the 4753-HSP to ICSF, you must first establish an exporter/importer key relationship between the 4753-HSP and ICSF. You can then write an application to export keys from the 4753-HSP key storage and import them into the ICSF CKDS. You can perform this type of key exchange only with CCA-defined keys, which have the same control vectors on key-encrypting keys. If your 4753-HSP installation includes non-CCA key types in key storage, you need to generate a special exporter/importer key-encrypting key pair on the 4753-HSP. The exporter key-encrypting key nullifies the CV value that is used on the 4753-HSP, and the importer key-encrypting key includes the CV value that is needed at ICSF.

- **Key labels**

  ICSF/MVS Version 1 Release 2 and above supports an extended key label of up to 64 bytes. Although the 4753-HSP also supports a 64-byte key label, there are additional key label formatting restrictions that do not apply to ICSF. The 4753-HSP key label consists of one to five name tokens that are separated by periods. Each name token includes one to eight alphanumeric or national string characters. ICSF, therefore, can accept all 4753-HSP key labels, but the 4753-HSP cannot accept all ICSF key labels. For more information on key label formatting restrictions, refer to *IBM Transaction Security System: Concepts and Programming Guide: Volume I, Access Controls and DES Cryptography*.

  ICSF/MVS Version 1 Release 2 and above, like the 4753-HSP, requires unique key labels for data-encrypting keys, data-translation keys, and MAC keys. To maintain compatibility with ICSF/MVS Version 1 Release 1, however, KGUP will continue to allow multiple key types per label for importer, exporter, and PIN keys under these conditions. Use either KGUP or the KEU to enter the keys, and ensure that the key labels do not conflict with other unique label restrictions.

- **UDX (User Defined Extension) support**

  Beginning with OS/390 V2 R10 ICSF, ICSF support is provided for UDX capabilities. UDX routines are developed by special contract with IBM and are only distributed to authorized customers.

  The UDX function is invoked by an "installation-defined" or generic callable service. The callable service is defined in the Installation Options data set (UDX parameter) and the service stub is link-edited with the application. The application program calls the service stub which accesses the UDX installation-defined service.

  There is a one-to-one correspondence between a specific generic service in ICSF and a specific UDX command processor in the PCICC, PCIXCC, or CEX2C. The administrator, through ISCF panels, performs UDX authorization processing on each PCI Cryptographic Coprocessor. Authorization is not LPAR specific. See Managing User Defined Extensions in *z/OS Cryptographic Services ICSF Administrator's Guide*, SA22-7521 for additional information.

  Support for writing your own UDX for a PCI Cryptographic Coprocessor is available. Development of a UDX for a PCIXCC or CEX2C requires a special contract with IBM. See the *UDX Reference and Guide* and the *4758 Custom Software Developer's Toolkit Guide* for additional information. These, and other

publications related to the IBM 4758 Coprocessor can be obtained in PDF format from the Library page located at http://www.ibm.com/security/cryptocards.

See the UDX parameter and Installation-Defined Callable Services in *z/OS Cryptographic Services ICSF System Programmer's Guide*, SA22-7520 for additional details.

Chapter 4. Operating ICSF

You use certain commands to operate ICSF. Also, there are different conditions for operating ICSF that you should consider. This topic describes the ICSF operating tasks.

Starting and stopping ICSF

To start ICSF, issue the operator START command. You must issue the START command after each IPL. When you issue the START command, verification tests check that the master key in each coprocessor is the same as the master key that enciphered the cryptographic key data set (CKDS) and that the hash patterns in each coprocessor is the same as the hash pattern of the master key that enciphered the PKA key data set (PKDS).

Releases of ICSF prior to FMID HCR7780 required an RSA master key on all systems. For non-CCF systems running HCR7780 or later, an RSA master key is no longer required on all systems. This requirement, however, still exists for CCF systems. For non-CCF systems, any combination of master keys can be loaded. The activation procedure for non-CCF systems selects the combination of master keys that will maximize the number of active coprocessors. ICSF checks the master keys available on the system (AES, DES, ECC and RSA) and determines validity based on the master keys used for the CKDS and PKDS. The master key verification patterns (MKVPs) contained in the header of the CKDS and PKDS are compared to the MKVPs of the master keys on the coprocessors. If they match, the master key is valid. After determining the valid master keys for the system, it then selects the set of available master keys that will maximize the number of active coprocessors.

- On **CCF Systems**, verification tests are also performed to ensure that the PCI Cryptographic Coprocessor SYM-MK on all PCI Cryptographic Coprocessors is the same as the DES master key on the Cryptographic Coprocessor Features, and that the PCI Cryptographic Coprocessor RSA-MK on all PCI Cryptographic Coprocessors is the same as the SMK on the Cryptographic Coprocessor Features.

  If a master key does not match the CKDS, this occurs:

  – ICSF starts.
  – A message that indicates the verification failed for the indicated coprocessor and CPU appears on the console.
  – The Cryptographic Coprocessor Feature on the CPU for which the verification failed is not active.

- On **PCIXCC Systems**, verification tests are also performed to ensure that the PCIXCC DES-MK is the same on all PCIXCC coprocessors, and that the PCIXCC RSA-MK is the same on all PCIXCC coprocessors.

  If a DES-MK master key verification pattern does not match the verification pattern in the CKDS, then: ICSF starts and a message that indicates the verification failed for the indicated coprocessor appears on the console. The PCIXCCs will not be active.

  If the RSA-MKs do not match, or if they match but the hash pattern does not match the hash pattern in the PKDS, a message indicates that the PKA hash pattern in the PKDS does not match the system PKA hash pattern. PKA callable services are not enabled.

  If the RSA-MKs do match the hash pattern in the PKDS but the DES-MK is not valid, then PKA callable services are not enabled. Once the DES-MK become valid, the user will have to enable the PKA services or stop and restart ICSF.
- On **CEX2C Systems**, verification tests are also performed to ensure that the DES-MK, AES-MK, and RSA-MK are the same on all CEX2C coprocessors.

  If DES-MK or AES-MK master key verification patterns do not match the verification patterns in the CKDS, then: ICSF starts and a message that indicates the verification failed for the indicated coprocessor appears on the console. In order for the coprocessor to become active, either the DES-MK or the AES-MK (or both) verification patterns must match those in the CKDS. If neither match, the coprocessor will not be active.

  If the RSA-MKs do not match, or if they match but the hash pattern does not match the hash pattern in the PKDS, a message indicates that the RSA hash pattern in the PKDS does not match the system RSA hash pattern. PKA callable services are not enabled.

  If the RSA-MKs do match the hash pattern in the PKDS but both the DES-MK and AES-MK are not valid, then PKA callable services are not enabled. Once the DES-MK or AES-MK becomes valid, the user will have to enable the PKA services or stop and restart ICSF.
- On **CEX3C Systems**, verification tests are also performed to ensure that the DES-MK, AES-MK, RSA-MK, and ECC-MK are the same on all CEX3C coprocessors.

  If DES-MK or AES-MK master key verification patterns do not match the verification patterns in the CKDS, then: ICSF starts and a message that indicates the verification failed for the indicated coprocessor appears on the console.

  If the RSA-MKs do not match or the ECC-MKs do not match, or if they match but do not match the hash pattern in the PKDS, a message indicates that the hash pattern in the PKDS does not match the system hash pattern.

  In order for a coprocessor to become active, at least one master key on the coprocessor must be valid.

  PKA callable services are enabled if the RSA-MK matches the hash pattern in the PKDS. PKA callable services are disabled if the RSA-MK does not match the hash pattern.

When ICSF successfully starts, a message that indicates that initialization is complete appears on the console.

This example shows the format of the START command to start ICSF, assuming that CSF is the name of the start procedure:

```
START CSF
```

You can start ICSF only as a started task.

To stop ICSF, issue the operator STOP command. After you issue the command, all ICSF processing stops. If ICSF stops successfully, a message that states that ICSF is stopped appears on the console.

This example shows the format of the STOP command to stop ICSF, assuming that CSF is the name of the started procedure:

```
STOP CSF
```

Notes:

1. If a problem is detected with a cryptographic coprocessor or with an accelerator during initialization, then a CSFM540I message is generated and the device is bypassed.
2. A Health Check, ICSF_COPROCESSOR_STATE_NEGCHANGE, monitors the state of the coprocessors and accelerators on a daily basis to detect a negative change in state.
3. If ICSF is unresponsive to the STOP command, be aware that you will not be able to use the CANCEL command to stop ICSF processing. Instead, use the force command:

   ```
   FORCE csfproc,arm
   ```

Modifying ICSF

When you issue the MODIFY command, ICSF gives control to the installation exit CSFEXIT5, if it exists. Your installation can write an exit routine for CSFEXIT5 that changes ICSF operations. For example, you might have the installation exit change the CHECKAUTH and KEYAUTH installation options without having to stop and restart ICSF. See Chapter 5, “Installation Exits,” on page 111 for a description of the installation exits.

If your installation does not write an exit routine for CSFEXIT5, no action occurs when you enter the MODIFY command.

Using different configurations

A central processor complex can have multiple cryptographic features of various types. This topics describes some of the different configurations available with the various servers.

Configuring the z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, and z196

There is only LPAR mode on a z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, and z196. You can divide your processor complex into PR/SM logical partitions. When you create logical partitions on your processor complex, you use the usage domain index on the Support Element Customize Image Profile page only if you have, or plan to add, PCICAs, PCIXCCs, CEX2Cs, CEX3Cs, CEX2As, or CEX3As.

The DOMAIN parameter is optional. The number that is specified for the usage domain index must correspond to the domain number you specified with the DOMAIN(n) keyword in the installation options data set – if you specified one. The DOMAIN keyword is required if more than one domain is specified as the usage domain on the PR/SM panels.

A PCICA, PCIXCC, CEX2C, CEX3C, CEX2A, or CEX3A can be configured and shared across multiple partitions.

Note: The domain assigned to the TKE Host LPAR must be unique if TKE is to control all the coprocessor cards in the environment. No other LPAR can use the domain assigned to the TKE Host.

With the z990, z890, z9 EC, z9 BC, z10 EC, z10 BC, and z196, there is support for up to 30 LPARs. On previous systems, where the maximum number of LPARs was 16, a domain was unique to an LPAR. With more than 16 LPARs to support, the domain may not be unique across LPARs but the same domain may be assigned to different LPARs if they are accessing different PCICAs, PCIXCCs, CEX2Cs, or CEX3Cs. This is illustrated by LPAR 1 and LPAR 3 in Figure 1. They are both assigned to usage domain 0 but on two different PCICAs.

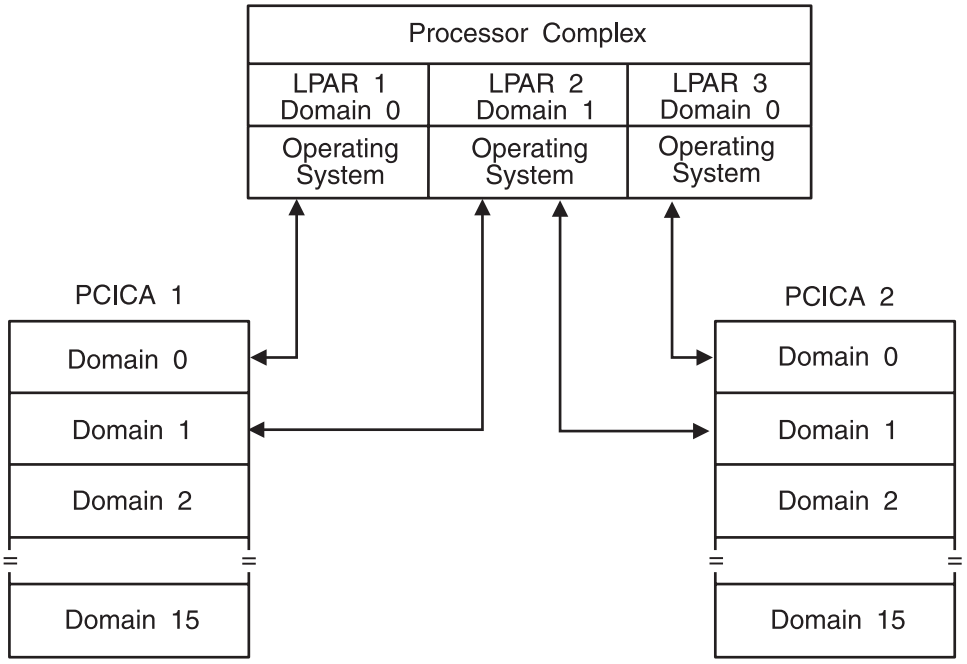

Figure 1. Two Crypto PCICAs on a Processor Complex Running in LPAR Mode

The example in Figure 1 shows that LPAR 2 has assigned access to Domain 1 on both PCICA 1 and PCICA 2. If you were to add another PCICA, PCIXCC, CEX2C, or CEX3C to LPAR 2, Domain 1 on the new PCICA, PCIXCC, CEX2C, or CEX3C would also be assigned.

A PCIXCC and PCICA configuration with domain sharing is illustrated by LPAR 1 and LPAR 3 in Figure 2 on page 80.

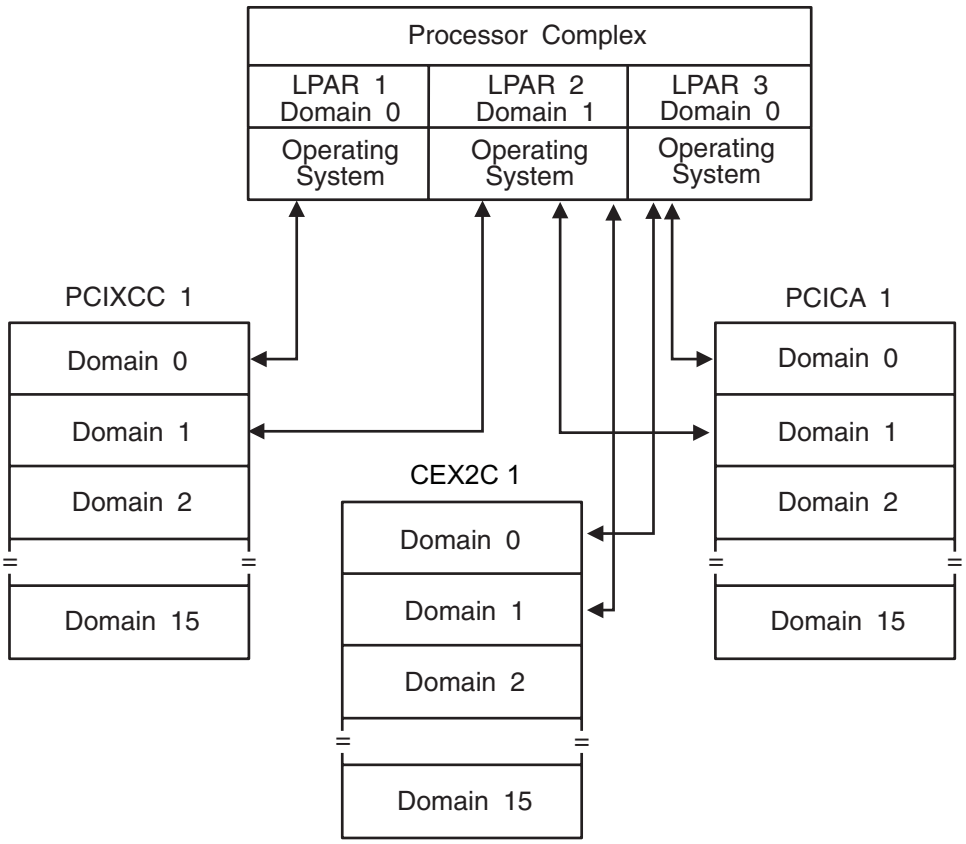

Figure 2. Multiple Crypto Coprocessors on a Complex Running in LPAR Mode

The example in Figure 2 shows that LPAR 2 has assigned access to Domain 1 on PCIXCC 1, CEX2C 1, and PCICA 1. LPAR 3 has assigned access to Domain 0 on CEX2C 1 and PCICA 1.

Configuring the IBM @server zSeries 900

The Cryptographic Coprocessor Feature can include up to two cryptographic coprocessors, each of which is attached to a central processor within the central processor complex. Each cryptographic coprocessor has sixteen master key register sets, referred to as domains. ICSF uses the domain usage index to access each domain. You can configure the complex to run in one of these modes:

- Single image mode
- Logical partition mode

Single Image Mode

In single image mode, the processor complex has access to the same domain on both Crypto CP 0 and Crypto CP 4. The domain is specified in the installation options data set. The DOMAIN parameter is optional. It is required if more than one domain is specified as the usage domain on the PR/SM panels or if running in native mode. See *z/OS Cryptographic Services ICSF System Programmer's Guide* for additional information on the DOMAIN parameter. The accessed domain on both coprocessors must have the same master key. Figure 3 on page 81 shows an example single image mode configuration.

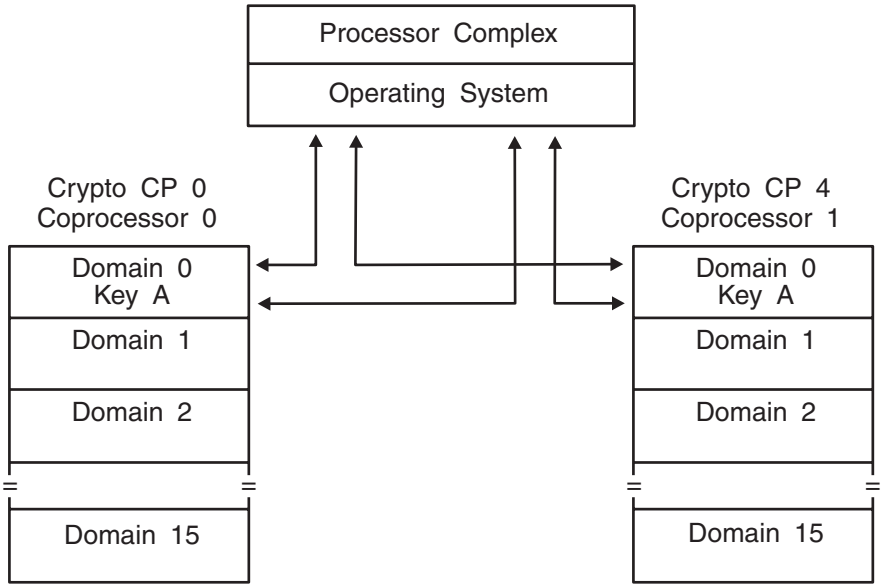

Figure 3. Two Crypto CPs on a Processor Complex Running in Single Image Mode

Logical Partition (LPAR) Mode

You can divide your processor complex into PR/SM logical partitions (LPARs). When you create logical partitions on your processor complex, you use the usage domain index on the Support Element Customize Image Profile page to enable access to a Crypto CP domain. The number that is specified for the usage domain index must correspond to the domain number you specify with the DOMAIN(n) keyword in the installation options data set. The DOMAIN parameter is optional. It is required if more than one domain is specified as the usage domain on the PR/SM panels or if running in native mode. See *z/OS Cryptographic Services ICSF System Programmer's Guide* for additional information on the DOMAIN parameter.

You can assign more than one domain to an LPAR, but you must use a unique installation options data set to define each domain. Assigning more than one domain to an LPAR makes it possible to have separate master keys for different purposes. For example, you might have one master key for production operations and a different master key for test operations.

The PCI Cryptographic Coprocessor can be configured like a Cryptographic Coprocessor Feature. It can be dedicated or shared across multiple partitions with each card supporting up to 16 domains.

The PCI Cryptographic Accelerator can be configured like a Cryptographic Coprocessor Feature. It can be dedicated or shared across multiple partitions with each card supporting up to 16 domains.

When using logical partitions, there is no domain sharing unless TKE is being used. The 'HOST' LPAR can control the domains of the other LPARS if the control domain for the first LPAR is setup for it. The example in Figure 4 on page 82 shows that LPAR 1 has access to Domain 0 on Crypto CP 0, Crypto CP 1, and PCICC. LPAR 2 has access to Domain 1 and Domain 2 on both Crypto CPs and on the PCICC. LPAR 1 cannot access Domain 1 or Domain 2 on the PCICC or on either of the Crypto CPs. Likewise, LPAR 2 cannot access Domain 0 on either Crypto CP or the

PCICC. The ICSF system running on the operating system in LPAR 2 has access to only one domain at any particular time, as specified in the installation options data set.

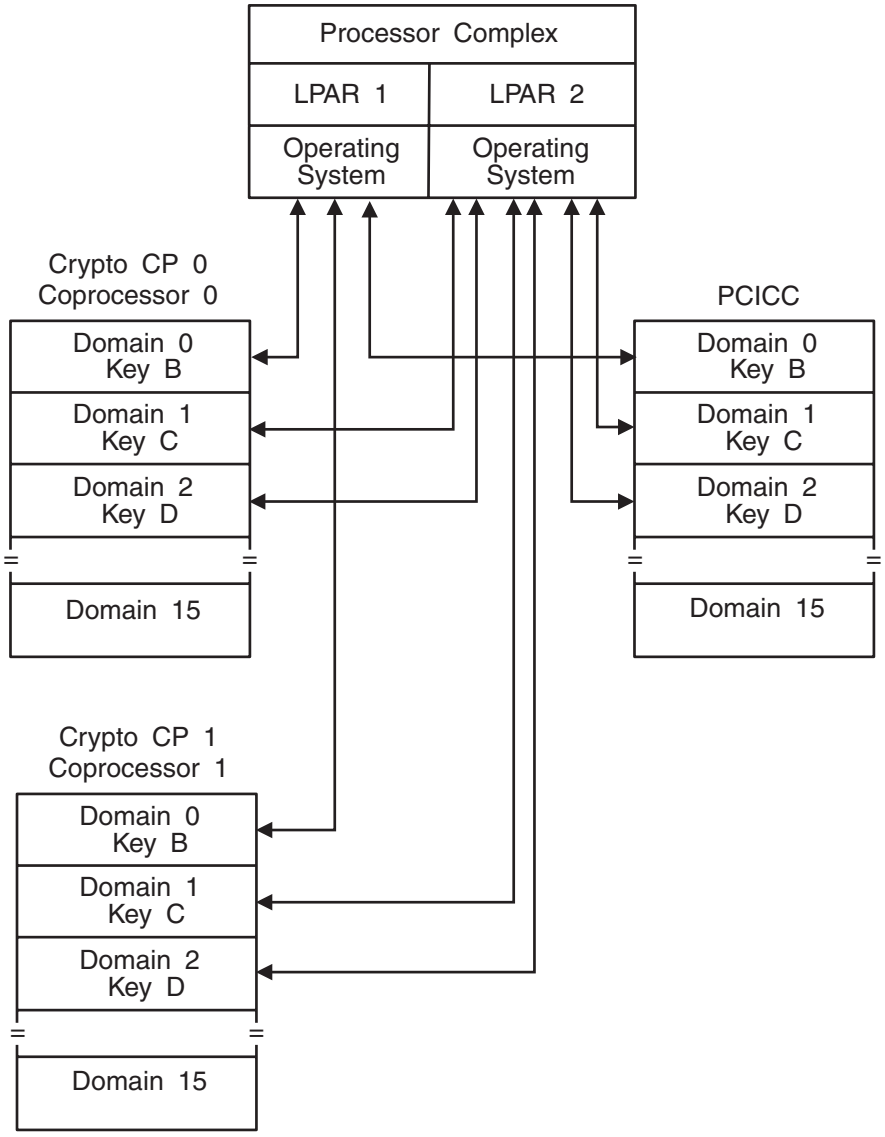

Figure 4. Two Crypto Coprocessors and one PCICC on a Processor Complex Running in LPAR Mode

Adding and Removing Cryptographic Coprocessors

It may become necessary for your installation to add or remove cryptographic coprocessors. This topic gives you a brief overview of the hardware implications. For more detailed information, refer to the *zSeries PR/SM Planning Guide* and the *zSeries Hardware Management Console Operations Guide (OS/2)*.

There are several terms associated with removing the cards. Use the Support Element (SE) panel to configure cryptographic coprocessors online and offline (standby). Use the ICSF Coprocessor Management panel from your TSO user ID to activate and deactivate cryptographic coprocessors. Use the TKE workstation to enable and disable cryptographic coprocessors.

Adding Cryptographic Coprocessors

You can dynamically add these cryptographic coprocessors: PCICA, PCICC, PCIXCC, CEX2C, CEX2A, CEX3C, and CEX3A. If you are adding a PCIXCC, CEX2C, or CEX3C, ensure that feature 3863 is installed on your z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, or z196.

The cryptographic coprocessor number must be in the Candidates list of the LPAR Activation panel. **Configure On** the card. Each coprocessor or accelerator will display as `ONLINE`. Once the master keys are entered, they become `ACTIVE`. The PCICA, CEX2A, and CEX3A will automatically become `ACTIVE`.

Steps for activating/deactivating cryptographic coprocessors

From your TSO userid, select option 1, Coprocessor Mgmt.

```
CSF@PRIM ----- Integrated Cryptographic Service Facility  ---------
OPTION ===> 1

Enter the number of the desired option.

  1  COPROCESSOR MGMT    -  Management of Cryptographic Coprocessors
  2  MASTER KEY MGMT     -  Master key set or change, CKDS/PKDS processing
  3  OPSTAT              -  Installation options
  4  ADMINCNTL           -  Administrative Control Functions
  5  UTILITY             -  ICSF Utilities
  6  PPINIT              -  Pass Phrase Master Key/CKDS Initialization
  7  TKE                 -  TKE Master and Operational key processing
  8  KGUP                -  Key Generator Utility processes
  9  UDX MGMT            -  Management of User Defined Extensions

       Licensed Materials - Property of IBM

      5694-A01 (C) Copyright IBM Corp. 1990, 2008. All rights reserved.
      US Government Users Restricted Rights - Use, duplication or
      disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Press ENTER to go to the selected option.
Press END   to exit to the previous menu.
```

Figure 5. Primary Panel

On the Coprocessor Management panel, you can select the coprocessors you want to activate or deactivate.

```
CSFGCMP0 ---------------- ICSF Coprocessor Management -------------
COMMAND ===>

Select the coprocessors to be processed and press ENTER.
Action characters are: A, D, E, K, R, and S. See the help panel for details.

                Serial
 CoProcessor    Number         Status   AES   DES   ECC   RSA
 -----------   ---------       ------   ---   ---   ---   ----
__ H00                         ACTIVE
A  G01          00000001       ONLINE    U     U     U     U
__ G02          00000002       ACTIVE    C     U     U     C
__ G03          00000003       ACTIVE    C     U     A     C
D  G04          00000004       ACTIVE    C     C     A     C
__ G05          00000005       ONLINE    U     C     E     U
__ E06          00000006       ACTIVE    C     C     -     C
__ G07          00000007       OFFLINE   -     -     -     -
```

Figure 6. Coprocessor Management Panel

Note: The coprocessors available for z9 EC and z9 BC are Exx and Fxx. The coprocessors available for z10 EC and z10 BC are Exx, Fxx, Gxx, and Hxx. The coprocessors available on the z196 are Gxx and Hxx.

When a PCICC, PCICA, PCIXCC, CEX2C, CEX2A, CEX3C, or CEX3A is deactivated through the Coprocessor Management Panel, the card is only deactivated for that one LPAR.

Steps to configure on/off cryptographic coprocessors

To configure the PCICC, PCICA, PCIXCC, CEX2C, CEX2A, CEX3C, or CEX3A cards online and offline, you must use the support element (SE) panel.

Before configuring a card offline, it is strongly recommended that you deactivate the card first from the ICSF Coprocessor Management panel. You need to 'deactivate' the card in ALL partitions that are using that card. This allow jobs to complete before the card is varied offline. You use the **Configure On/Off** service on the Support Element panel to take the card offline (standby).

After you configure the card offline from the SE panel, hit enter on the coprocessor management panel to verify that the card is offline. This configuring is done to remove and replace cards or to load new card code for the PCIXCC, CEX2C, CEX3C, and PCICC cards.

To bring a card back online, use the SE panel again. If a card was deactivated and then brought offline (configured off), you will need to activate it again through the Coprocessor Management panel.

There are no z/OS operator commands to vary these devices.

Steps for enabling/disabling cryptographic coprocessors (PCICC, PCIXCC, CEX2C, and CEX3C)

With TKE 3.0 or higher you can disable/enable the PCICCs. With TKE V4.0 or higher, you can disable/enable the PCIXCCs/CEX2Cs. With TKE V6.0, you can disable/enable the CEX3Cs.

When a PCICA, PCICC, PCIXCC, CEX2C, CEX2A, CEX3C, or CEX3A is deactivated through the Coprocessor Management Panel, the card is only

deactivated for that one LPAR. When a PCICC, PCIXCC, CEX2C, or CEX3C is disabled by TKE, the card is disabled for the entire system, not just the LPAR that issued the disable.

Intrusion Latch on the PCICC, PCIXCC, CEX2C, or CEX3C

Under normal operation, the intrusion latch on a PCICC, PCIXCC, CEX2C, or CEX3C is tripped when the card is removed. This causes all installation data, master keys, retained keys, roles and authorities to be zeroized in the card when it is reinstalled.

If a situation arises where a PCIXCC, CEX2C, or CEX3C needs to be removed, for example, you need to remove your card for service, and you do not want the installation data to be cleared, perform this procedure to disable the PCIXCC, CEX2C, or CEX3C before removing.

There is no similar procedure for the PCICC.

This process will require you to switch between the TKE application, the ICSF Coprocessor Management panel, and the Support Element.

1. Open an Emulator Session on the TKE workstation and logon to your TSO userid on the Host System where the PCIXCC, CEX2C, or CEX3C will be removed.
2. From the ICSF Primary Option Menu on TSO, select Option 1 for Coprocessor Management.
3. Leave the Coprocessor Management panel displayed during the rest of this procedure. You will be required to hit ENTER on the Coprocessor Management panel at different times. DO NOT EXIT this panel.
4. Open the TKE Host where the PCIXCC, CEX2C, or CEX3C will be removed. Open the PCIXCC, CEX2C, or CEX3C. Click on Disable Crypto Module.
5. After the PCIXCC, CEX2C, or CEX3C has been disabled from TKE, hit ENTER on the Coprocessor Management panel. The status should change to DISABLED.

   Note: You do not need to deactivate a disabled card.
6. **Configure Off** the PCIXCC, CEX2C, or CEX3C from the Support Element.
7. After the card has been taken Offline, hit ENTER on the Coprocessor Management panel. The status should change to OFFLINE.
8. Remove the PCIXCC, CEX2C, or CEX3C. Perform whatever operation needs to be done. Replace the PCIXCC, CEX2C, or CEX3C.
9. **Configure On** the PCIXCC, CEX2C, or CEX3C from the Support Element.
10. When the initialization process is complete, hit ENTER on the Coprocessor Management panel. The status should change to DISABLED.
11. From the TKE Workstation Crypto Module General page, click on Enable Crypto Module.
12. After the PCIXCC, CEX2C, or CEX3C has been enabled from TKE, hit ENTER on the Coprocessor Management panel. The Status should return to its original state. If the Status was ACTIVE in step 2, when the PCIXCC, CEX2C, or CEX3C is enabled it should return to ACTIVE.

All installation data; master keys, retained keys, roles, and authorities should still be available. The PCIXCC, CEX2C, or CEX3C data was not cleared with the card removal because it was Disabled first via the TKE workstation.

Steps for enabling/disabling cryptographic coprocessors (CCF)

With TKE V3.0 and higher, you can enable and disable the Cryptographic Coprocessor Feature.

Cryptographic functions can be disabled through the ECM Domains Controls function. This places the Cryptographic Coprocessor Feature in standby mode. The functions can be brought out of standby mode by enabling the cryptographic function bit in the ECM through TKE.

Note: Status displayed on the coprocessor management panel will show DISABLED, not STANDBY.

Performance considerations for using installation options

You specify installation options in the installation options data set. Three installation options, CHECKAUTH, KEYAUTH, and CKTAUTH provide additional security checking, but affect performance.

In ICSF, the Security Server (RACF) always checks non-Supervisor State callers. The CHECKAUTH option allows you to specify whether CSF performs access control checking of Supervisor State and System Key callers. Specify CHECKAUTH(NO) if you do not want CSF to check Supervisor State and System Key callers. Specify CHECKAUTH(YES) if you want CSF to check Supervisor State callers. Checking Supervisor State and System Key callers significantly affects performance.

The KEYAUTH option allows you to specify whether ICSF should authenticate an entry in the CKDS whenever ICSF accesses the entry. ICSF creates a message authentication code (MAC) for each entry in the CKDS and stores the MAC with the entry. Whenever ICSF retrieves an entry from the CKDS, ICSF uses the MAC to authenticate the entry. When ICSF authenticates the entry, ICSF verifies that the entry was not inadvertently changed or damaged. If the authentication fails, ICSF returns either a return code with a reason code or message.

You specify KEYAUTH(NO) for ICSF not to authenticate an entry or KEYAUTH(YES) for ICSF to authenticate an entry. The authentication can have a significant impact on performance when using the Crypto Express2 or Crypto Express3 feature. The chance of an error occurring in the in-storage CKDS is minimal. However, the authentication might be useful for diagnostic purposes if an error occurs.

The CKTAUTH option allows you to specify whether ICSF should authenticate an entry in the CKDS whenever ICSF reads the record from DASD. Customers with a large CKDS may experience a performance impact as each authentication requires a request to the PCIXCC, CEX2C, or CEX3C. CKTAUTH has no effect on the KEYAUTH option.

The SYSPLEXCKDS, SYSPLEXPKDS and SYSPLEXTKDS options specify whether sysplex-wide data consistency for the CKDS, PKDS, and TKDS is desired. For a description of the subkeywords, see “Parameters in the installation options data set” on page 38.

Dispatching priority of ICSF

To avoid performance problems, the dispatching priority of ICSF should be set at least as high as that of the highest task using ICSF.

VTAM session-level encryption

ICSF supports VTAM session-level encryption. VTAM session-level encryption provides protection for messages within SNA sessions, that is, between pairs of logical units that support their respective end users. When this method of protection is in effect, data is enciphered by the originating logical unit and deciphered only by the destination logical unit. Thus, the data never appears in the clear while passing through the network.

ICSF places no restrictions on the addressing mode of calling programs. In particular, when VTAM session-level encryption is used with ICSF, VTAM can use storage greater than 16 megabytes.

System SSL encryption

ICSF supports System SSL encryption on all servers.

On the z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, or z196, a PCICA, PCIXCC, CEX2C, or CEX3C is required.

On CCF systems that also have PCICAs, you can run system SSL encryption without entering master keys.

Access method services cryptographic option

In compatibility mode, ICSF supports the Access Method Services Cryptographic Option. The option enables the user of the Access Method Services REPRO command to use the Data Encryption Algorithm to encipher data.

The Access Method Services user can use REPRO to encipher data that is written to a data set, and then store the enciphered data set offline. When desired, you can bring the enciphered data set back online, and use REPRO to decipher the enciphered data. You can decipher the data either on the host processor on which it was enciphered, or on another host processor that contains the Access Method Services Cryptographic Option and the same cryptographic key that was used to encipher the data. You can either use ICSF to create the cryptographic keys, or use keys that the Access Method Services user supplies.

With the exception of catalogs, all data set organizations that are supported for input by REPRO are eligible as input for enciphering. Similarly, with the exception of catalogs, all data set organizations supported for output by REPRO are eligible as output for deciphering. The resulting enciphered data sets are always sequentially organized (SAM or VSAM entry-sequenced data sets).

See Appendix E, “Using AMS REPRO Encryption,” on page 317 for more information in using this method.

Remote Key Loading

The process of remote key loading is loading DES keys to automated teller machines (ATMs) from a central administrative site. Because a new ATM has none of the bank's keys installed, getting the first key securely loaded is currently done manually by loading the first key-encrypting key (KEK) in multiple cleartext key parts. A new standard ANSI X9.24-2 defines the acceptable methods of doing this using public key cryptographic techniques, which will allow banks to load the initial KEKs without having to send anything to the ATMS. This method is quicker, more reliable and much less expensive.

Once an ATM is in operation, the bank can install new keys as needed by sending them enciphered under a KEK it installs at a previous time. Cryptographic architecture in the ATMs is not Common Cryptographic Architecture (CCA) and it is difficult to export CCA keys in a form understood by the ATM. Remote key loading will make it easier to export keys to non-CCA systems without compromising security.

In order to use ATM Remote Key Loading, TKE users will have to enable the access control points for these functions:

- Trusted Block Create - API Keyword = Inactive
- Trusted Block Create - API Keyword = Active
- Public Key Import - Source Key Token = Trusted Block
- Public Key Import - Source Key Token = PKA96 Key Token
- Remote Key Export

Event Recording

ICSF records certain ICSF events in the System Management Facilities (SMF) data set. ICSF also sends messages that are generated during processing to the ICSF job log and consoles. The SMF recording and messages help you detect problems and track events. This topic describes the events that ICSF records in the SMF record and describes where ICSF sends certain messages.

These records can be used with RACF SMF type 80 record to audit use of the callable services and the keys. The RACF type 80 records are extracted and formatted using the RACF SMF Unload Utility. See *z/OS Security Server RACF Auditor's Guide* for information on how to use this utility. For information about the formatted SMF records see *z/OS Security Server RACF Macros and Interfaces*.

System Management Facilities (SMF) Recording

ICSF uses SMF record type 82 to record certain ICSF events. Record type 82 contains:

- a **fixed header / self-defining section**. This section contains the common SMF record headers fields and the triplet fields (offset/length/number), if applicable, that locate the other sections on the record.
- **ICSF event specific (subtype) section**. Each subtype contains information about the event that caused ICSF to write to the SMF record. For subtypes that log state changes, the SMF record will contain additional auditing sections.
- an **auditing header section**. This section is present in the record for subtypes that log state changes. It describes the number and overall length of the auditing sections that follow.

- a **server user section** and, optionally, an **end user section**. If both sections are present, they can appear in either order.

You can map record type 82 by using the CSFSMF82 macro.

ICSF records information in the SMF data set when these events occur:

- ICSF starts
- ICSF status changes on a processor
- ICSF handles error conditions for Cryptographic Coprocessor Feature failure or tampering
- You enable or disable special secure mode
- You enter a master key part
- You use the ICSF panels to process an operational key part or key part register loaded using the TKE workstation
- TKE commands and responses are all audited through SMF 82 (TKE commands on the Cryptographic Coprocessor Feature use CSFPKSC. TKE commands on the PCICC, PCIXCC, CEX2C, and CEX3C use CSFPCI.)
- The in-storage cryptographic key data set (CKDS) is refreshed
- A dynamic change is made to the PKDS
- The in-storage PKDS is refreshed
- Duplicate tokens were detected
- A key store policy check resulted in a 'warning'
- You use the ICSF panels to update the new master key register on a PCICC, PCIXCC, CEX2C, or CEX3C
- You create or delete a retained key on a PCICC, PCIXCC, CEX2C, or CEX3C
- The TKE workstation issues a PCICC, PCIXCC, CEX2C, or CEX3C command request or receives a reply response from a PCICC, PCIXCC, CEX2C, or CEX3C
- ICSF records processing times for PCICCs, PCIXCCs, CEX2Cs, CEX3Cs, PCICAs, CEX2As, and CEX3As.
- A PCICA, PCICC, PCIXCC, CEX2C, CEX3C CEX2A, or CEX3A is either brought online or taken offline
- ICSF issues IXCJOIN to join the ICSF sysplex group or issues IXCLEAVE to leave the sysplex group.
- The trusted block create callable service is used to create or activate a trusted block.

Each of these events causes ICSF to record information in a separate subtype in the SMF record.

Recording and Formatting type 82 SMF Records in a Report - Sample jobs are available (in SYS1.SAMPLIB) to assist in the recording and formatting of type 82 SMF data:

- **CSFSMFJ** - JCL that executes the code to dump and format SMF type 82 records for ICSF. Before executing the JCL, you need to make modifications to the JCL (see the prologue in the sample for specific instructions). After the JCL has been modified, terminate SMF recording of the currently active dump dataset (by issuing I SMF) to allow for the unloading of SMF records. After SMF recording has been terminated, execute the JCL. The output goes into the held queue. This is an example of CSFSMFJ.

```
//CSFSMFJ  JOB <JOB CARD PARAMETERS>
//*******************************************************************
//*  LICENSED MATERIALS - PROPERTY OF IBM                           *
//*  5694-A01                                                       *
//*  (C) COPYRIGHT IBM CORP. 2002                                   *
//*                                                                 *
//*  This JCL reads Type 82 SMF records and formats them in a report.*
//*                                                                 *
//*  CAUTION: This is neither a JCL procedure nor a complete JOB.   *
//*  Before using this JOB step, you will have to make the following *
//*  modifications:                                                 *
//*                                                                 *
//*   1) Add the job parameters to meet your system requirements.   *
//*   2) Change the DUMPIN DSN=hlq.smfdata.input to be the name of  *
//*      the dataset where you currently have SMF data being        *
//*      recorded.                                                  *
//*   3) Change the STEPLIB VOL=SER=ttttt1 and VOL=SER=ttttt2 to    *
//*      be the volumes where these sort datasets reside.           *
//*   4) Change the SYSPROC DSN=hlq.rexx.dataset to be the name of  *
//*      the dataset where you have placed the CSFSMFR REXX sample. *
//*                                                                 *
//*   Prior to executing this job, you need to terminate SMF        *
//*   recording of the currently active dump dataset for allow the  *
//*   unload of SMF records.                                        *
//*                                                                 *
//*******************************************************************
//*
//*-----------------------------------------------------------------*
//*   UNLOAD SMF 82 RECORDS FROM VSAM TO VBS                        *
//*-----------------------------------------------------------------*
//SMFDMP   EXEC  PGM=IFASMFDP
//DUMPIN   DD    DISP=SHR,DSN=hlq.smfdata.input
//DUMPOUT  DD    DISP=(NEW,PASS),DSN=&&VBS,UNIT=3390,
//         SPACE=(CYL,(1,1)),DCB=(LRECL=32760,RECFM=VBS,BLKSIZE=4096)
//SYSPRINT DD    SYSOUT=*
//SYSIN    DD    *
    INDD(DUMPIN,OPTIONS(DUMP))
    OUTDD(DUMPOUT,TYPE(82))
//*
//*-----------------------------------------------------------------*
//*   COPY VBS TO SHORTER VB AND SORT ON DATE/TIME                  *
//*-----------------------------------------------------------------*
//COPYSORT EXEC PGM=SORT,REGION=6000K
//STEPLIB  DD DISP=SHR,DSN=SYS1.SORTLPA,VOL=SER=ttttt1,UNIT=3390
//         DD DISP=SHR,DSN=SYS1.SICELINK,VOL=SER=ttttt2,UNIT=3390
//SYSOUT   DD SYSOUT=*
//SORTWK01 DD UNIT=3390,SPACE=(CYL,10)
//SORTIN   DD DISP=(OLD,DELETE),DSN=&&VBS
//SORTOUT  DD    DISP=(NEW,PASS),DSN=&&VB,UNIT=3390,
//         SPACE=(CYL,(1,1)),DCB=(LRECL=3000,RECFM=VB)
//SYSIN    DD *
 SORT FIELDS=(11,4,A,7,4,A),FORMAT=BI,SIZE=E4000
//*
//*-----------------------------------------------------------------*
//*   FORMAT TYPE 82 RECORDS                                        *
//*-----------------------------------------------------------------*
//FMT     EXEC PGM=IKJEFT01,REGION=5128K,DYNAMNBR=100
//SYSPROC  DD DISP=SHR,DSN=hlq.rexx.dataset
//SYSTSPRT DD SYSOUT=*
//INDD     DD DISP=(OLD,DELETE),DSN=&&VB
//OUTDD    DD SYSOUT=*
//SYSTSIN DD *
  %CSFSMFR
```

- **CSFSMFR** - An EXEC that formats the SMF type 82 records into a readable report.

ICSF Initialization (Subtype 1)

When ICSF starts, ICSF writes to subtype 1 after initialization is completed. Subtype 1 describes the values of installation options that are specified in the installation options data set.

Subtype 1 contains this information:

- Special secure mode (SSM) option
- Key authentication (KEYAUTH) option
- Security Server (RACF) checking of Supervisor State and System Key callers (CHECKAUTH) option
- Compatibility mode with CUSP or PCF (COMPAT) option
- Cryptographic domain number (DOMAIN) option
- Number of trace entries (TRACEENTRY) option
- CKDS name (CKDSN) option
- Maximum length for data in a callable service (MAXLEN) option

  Beginning with z/OS V1 R2, the MAXLEN parameter may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647). If a value greater than this is specified, an error will result and ICSF will not start.
- CKDS record authentication (CKTAUTH) option
- User parameter (USERPARM) option
- PKDS name (PKDSN) option
- TKDS name (TKDSN) option

SMF records for this subtype will also contain a server user audit section.

ICSF Status Change (Subtype 3)

ICSF writes to subtype 3 when processors are verified at initialization, after a master key is set or changed, when ICSF switches from stand-by mode to normal mode, or when a processor comes online or offline. When processor status changes, subtype 3 gives the status of the processors still online.

Subtype 3 contains this information:

- Processor number
- Coprocessor number
- Cryptographic domain number
- Master key version number

If a master key change or set occurs, subtype 3 also contains this information:

- Master key verification pattern
- Old master key verification pattern, if an old master key exists
- New master key verification pattern, if a new master key exists

Error Handling for Cryptographic Coprocessor Feature (Subtype 4)

ICSF writes to subtype 4 when the Coprocessor is in standby mode or when the Cryptographic Coprocessor Feature detects tampering.

Subtype 4 contains this information:

- Status word from the Cryptographic Coprocessor Feature
- Processor number
- Cryptographic domain number

Special Secure Mode Change (Subtype 5)

Subtype 5 contains special secure mode status bit. ICSF writes to subtype 5 when the status of special secure mode changes. ICSF also updates subtype 5 when the Cryptographic Coprocessor Feature indicates that special secure mode was required for an instruction, but was not enabled.

Master Key Part Entry (Subtype 6)

ICSF writes to subtype 6 when master key parts are entered using TKE workstation and are processed using the TKE master key entry ICSF panels. Subtype 6 contains this information:

- The verification pattern for the master key part
- The verification pattern for the new master key
- A bit indicating whether the verification pattern is valid
- The Coprocessor number
- The cryptographic domain number

If you enter the final master key part, the record also contains the verification pattern for the entire master key and a bit indicating whether the verification pattern is valid.

Operational Key Part Entry (Subtype 7)

ICSF writes to subtype 7 when key parts are entered using the TKE workstation and are processed using the operational key entry ICSF panels. Subtype 7 contains this information:

- The ENC-ZERO verification pattern of the completed key for a PCIXCC, CEX2C, or CEX3C or the CCF verification pattern
- A bit indicating whether the verification pattern is valid
- The cryptographic coprocessor domain number
- The cryptographic coprocessor number
- The name of the CKDS that contains the entry with the key part
- The label of the CKDS entry that contains the key part

SMF records for this subtype will also contain server user and end user audit sections.

CKDS Refresh (Subtype 8)

ICSF writes to subtype 8 when the in-storage CKDS is successfully refreshed. ICSF refreshes the in-storage CKDS by reading a disk copy of a CKDS into storage. Subtype 8 contains this information:

- Name of the current in-storage CKDS that ICSF refreshes
- Name of the disk copy of the CKDS that ICSF read into storage to replace the current CKDS

SMF records for this subtype will also contain server user and end user audit sections.

Dynamic CKDS Update (Subtype 9)

ICSF writes to subtype 9 when an application uses the dynamic CKDS update services to write to the CKDS. Subtype 9 contains this information:

- Name of the changed CKDS
- An indication of the operation performed.
- The CKDS entry (which includes the label name and key type) that was changed

SMF records for this subtype will also contain server user and end user audit sections.

PKA Key Part Entry (Subtype 10)

ICSF writes to subtype 10 when you use the ICSF panels to enter PKA master key parts. Subtype 10 contains this information:

- An indication of which PKA Master key is changing; the Signature Master Key (SMK), or the Key Management Master Key (KMMK)
- An indication of whether the hash pattern of the PKA master key register is valid (It is valid when the final key part is entered.)
- The hash pattern (MDC-4) of the PKA master key register
- The hash pattern of PKA key part
- The Coprocessor number
- Current cryptographic domain

If no DES master key has been validated, the key part entries do not contain a hash pattern. The record for the final key contains the hash pattern of the complete key.

Clear New Master Key Part Entry (Subtype 11)

ICSF writes to subtype 11 when you use the ICSF panels to enter new master key parts. Subtype 11 contains this information:

- An indication of whether the hash patterns for the new master key register and new master key part are valid. (The new master keys register hash pattern is only valid once the final key part in entered.)
- An indication of whether the verification patterns for the new master key register and key part are valid. (The new master key verification pattern is valid only after the final key part is entered.)
- The hash pattern of the new master key register
- The verification pattern of the new master key register
- The hash pattern of new master key part
- The verification pattern of new master key part
- The Coprocessor number
- Current cryptographic domain

If no DES master key has been validated, the key part entries do not contain a verification pattern and hash pattern. The record for the final key contains the verification pattern and hash pattern of the complete key.

PKSC Commands (Subtype 12)

ICSF writes to subtype 12 for every PKSC command entered through the CSFPKSC interface. Subtype 12 contains this information:

- The complete PKSC request
- The corresponding PKSC response

Dynamic PKDS Update (Subtype 13)

ICSF writes to subtype 13 when an application uses the dynamic PKDS update services to change the PKDS. Subtype 13 contains this information:

- The name of the changed PKDS
- An indication of the operation performed.
- The name of the changed entry in the PKDS

SMF records for this subtype will also contain server user and end user audit sections.

Cryptographic Coprocessor Clear Master Key Entry (Subtype 14)

ICSF writes to subtype 14 whenever you use ICSF panels to update AES-MK, DES-MK, ECC-MK, or RSA-MK in the new master key register in a PCICC, PCIXCC, CEX2C, or CEX3C. Subtype 14 contains this information:

- The master Key valid indicator
- The indicator for a PCICC, PCIXCC, CEX2C, or CEX3C
- The new master key verification pattern
- The key part verification pattern
- The cryptographic coprocessor processor number
- The cryptographic coprocessor serial number
- The cryptographic coprocessor domain index

SMF records for this subtype will also contain server user and end user audit sections.

Cryptographic Coprocessor Retained Key Create or Delete (Subtype 15)

ICSF writes to subtype 15 whenever you create or delete a retained private key in a PCICC, PCIXCC, CEX2C, or CEX3C. Subtype 15 contains this information:

- The operation performed (created, deleted from PCI, deleted from PKDS)
- The indicator for a PCICC, PCIXCC, CEX2C, or CEX3C
- The retained key label
- The cryptographic coprocessor processor number
- The cryptographic coprocessor serial number
- The domain index

SMF records for this subtype will also contain server user and end user audit sections.

Cryptographic Coprocessor TKE Command Request or Reply (Subtype 16)

ICSF writes to subtype 16 whenever a TKE workstation either issues a command request to, or receives a reply response from, a PCICC, PCIXCC, CEX2C, or CEX3C. Subtype 16 contains this information:

- The indicator for request or reply
- The indicator for a PCICC, PCIXCC, CEX2C, or CEX3C
- The cryptographic coprocessor processor number
- The cryptographic coprocessor serial number
- The cryptographic coprocessor domain index
- The request command block or reply response block length
- The request command data block or reply response data block length
- The request or reply CPRB
- The length of the fixed audit data
- The number of relocate sections
- The function id
- The function return code
- The function description - describes the function id.

SMF records for this subtype will also contain server user and end user audit sections.

PCI Cryptographic Coprocessor Timing (Subtype 17)

ICSF periodically records processing times for PCI Cryptographic Coprocessor operations in subtype 17. Subtype 17 contains this information:

- The time immediately before the operation begins
- The time immediately after the operation ends
- The time immediately after the results of the operation have been communicated to the caller address space
- The number of processes waiting to submit work to the same PCI Cryptographic Coprocessor, domain, and reference slot used by this operation

- The function code for this operation
- The PCI Cryptographic Coprocessor processor number
- The PCI Cryptographic Coprocessor serial number
- The PCI Cryptographic Coprocessor domain
- A reference number that identifies an internal ICSF queue element

Cryptographic Coprocessor Configuration (Subtype 18)

ICSF writes subtype 18 when a PCICA, PCICC, PCIXCC, CEX2C, CEX2A, CEX3C, or CEX3A is brought online or taken offline. Subtype 18 contains this information:

- The operation performed (coprocessor brought online, taken offline)
- The coprocessor number
- The PCICC, PCIXCC, CEX2C, or CEX3C serial number, or a PCICA, CEX2A, or CEX3A number

PCI X Cryptographic Coprocessor Timing (Subtype 19)

ICSF periodically records processing times for PCIXCC operations in subtype 19. Subtype 19 contains this information:

- The time immediately before the operation begins
- The time immediately after the operation ends
- The time immediately after the results of the operation have been communicated to the caller address space
- The number of processes waiting to submit work to the same PCIXCC, domain, and reference slot used by this operation
- The function code for this operation
- The PCIXCC processor number
- The PCIXCC serial number
- The PCIXCC domain
- A reference number that identifies an internal ICSF queue element

Cryptographic Coprocessor Timing (Subtype 20)

ICSF periodically records processing times for PCIXCC, CEX2C, CEX3C, CEX2A, and CEX3A operations in subtype 20. Subtype 20 contains this information:

- The device type
- The time immediately before the operation begins
- The time immediately after the operation ends
- The time immediately after the results of the operation have been communicated to the caller address space
- The number of processes waiting to submit work to the same coprocessor, domain, and reference slot used by this operation
- The function code for this operation
- The PCIXCC, CEX2C, CEX3C, CEX2A, or CEX3A processor number
- The PCIXCC, CEX2C, CEX3C, CEX2A, or CEX3A serial number
- The PCIXCC, CEX2C, CEX3C, CEX2A, or CEX3A domain
- A reference number that identifies an internal ICSF queue element

ICSF Sysplex Group (Subtype 21)

ICSF writes subtype 21 when ICSF joins or leaves the ICSF sysplex group. Subtype 21 contains this information:

- The name of the ICSF sysplex group
- The name of the sysplex member
- An indication of whether the member joined or left the sysplex group
- An indication of whether the join or leave was due to normal initialization/termination processing
- An indication of whether the leave was due to error recovery processing
- The time of the join or leave

- The name of the active CKDS

Trusted Block Create (Subtype 22)

ICSF writes subtype 22 when the Trusted Block Create callable services are invoked. Subtype 22 contains this information:

- Type of call, Active or Inactive
- If a Public Key Section was present in the Trusted Block Token
- ASID of the Caller
- If Input Trusted Block Token is in the PKDS, save it's Label
- If Output Trusted Block Token is in the PKDS, save it's Label
- If the Transport Key Token is in the CKDS, save it's Label

SMF records for this subtype will also contain server user and end user audit sections.

Token Data Set (TKDS) (Subtype 23)

ICSF writes subtype 23 when the Token Data Set (TKDS) record is updated (created, modified, deleted) of PKCS #11 tokens or token objects. Token Data Set callable services are invoked. Subtype 23 contains this information:

- The name of the changed TKDS
- An indication of the operation performed
- The name of the changed entry in the TKDS

SMF records for this subtype will also contain server user and end user audit sections.

Duplicate Key Tokens (Subtype 24)

ICSF writes subtype 24 when the security administrator has indicated that duplicate key tokens must be identified. Subtype 24 contains this information:

- The data set name
- The number of key labels
- The key labels

Key Store Policy (Subtype 25)

ICSF writes subtype 25 when a callable service checks the key store policy. Subtype 25 contains this information:

- The list information (incomplete, from CKDS, from PKDS)
- The number of key labels
- The unauthorized duplicate key label and key type

SMF records for this subtype will also contain server user and end user audit sections.

PKDS Data Space Refresh (Subtype 26)

ICSF writes to subtype 26 when the in-storage PKDS is successfully refreshed. ICSF refreshes the in-storage PKDS by reading a disk copy of a PKDS into storage. Subtype 26 contains this information:

- Name of the current in-storage PKDS that ICSF refreshes
- Name of the disk copy of the PKDS that ICSF read into storage to replace the current PKDS

SMF records for this subtype will also contain server user and end user audit sections.

PKA Key Management Extensions (Subtype 27)

When PKA Key Management Extensions are enabled, ICSF writes to subtype 27 to record operational and error information related to PKA Key Management Extensions. A subtype 27 record is written:

- when a CSF.PKAEXTNS.ENABLE or CSF.PKAEXTNS.ENABLE.WARNONLY profile in the XFACILIT class uses the APPLDATA field to specify a trusted certificate repository, an SMF record is cut to indicate if the trusted certificate repository was successfully changed, or whether there was an error. The APPLDATA field and the repository it specifies will be checked at startup and whenever the XFACILIT class is RACLISTed. ICSF will write a subtype 27 record if the certificate repository is changed, or if there is an error. In this case, subtype 27 will indicate if:
  - the trusted certificate repository was changed
  - the specified trusted certificate repository is empty
  - an error was detected while extracting the APPLDATA
  - the specified repository was not found
  - one or more certificates could not be parsed
- when an application calls a service attempting to use a key in a way that is not allowed by the ICSF segment specifications within the CSFKEYS or XCSFKEY profile that covers the key. The SMF record will be written at the completion of the callable service, which, depending on whether PKA Key Management Extensions had been enabled in warning or fail mode, may or may not allow the requested operation on the key. Subtype 27 contains this information. In this case, subtype 27 will indicate if:
  - an asymmetric key may not be used for the requested function
  - a symmetric key cannot be exported by the provided asymmetric key

SMF records for this subtype will also contain server user and end user audit sections.

High Performance Encrypted Key (Subtype 28)

Symmetric Key Encipher (CSNBSYE, CSNBSYE1, CSNESYE and CSNESYE1) and Symmetric Key Decipher (CSNBSYD, CSNBSYD1, CSNESYD and CSNESYD1) callable services exploit CP Assist for Cryptographic Functions (CPACF) for improved key management performance. A CKDS encrypted key can be used in these services, but only when SYMCPACFWRAP(YES) is specified in the ICSF segment of the CSFKEYS class profile that covers the key. ICSF writes to subtype 28 at the completion of functions that attempt to wrap an encrypted key under the CPACF wrapping key. Subtype 28 will indicate if the rewrapping operation is:

- permitted for this symmetric key
- not permitted for this symmetric key

SMF records for this subtype will also contain server user and end user audit sections.

TKE Workstation Audit Record (Subtype 29)

If you have the optional TKE Workstation, you can use the TKE Audit Record Upload Configuration Utility to send Trusted Key Entry workstation security audit records to a System z host, where they will be saved in the z/OS System

Management Facilities (SMF) dataset. Each TKE security audit record is stored in the SMF dataset as a type 82 subtype 29 record. For more information on the TKE Audit Record Upload Configuration Utility, refer to the *z/OS Cryptographic Services ICSF TKE Workstation User's Guide*.

Message Recording

ICSF writes messages to the job log, and to the security console and the operator console.

ICSF writes most of its messages to the job log. Messages that demand action from the master console operator will display on the operator console, and messages related to system security will display on the security console. Some of these console messages will appear only on the console, and some will also be written to the job log. Messages that are not displayed on either the operator or security console are written to the job log.

For a description of each ICSF message, see *z/OS Cryptographic Services ICSF Messages*.

Security Considerations

You can provide enhanced security on ICSF by controlling access to resources and changing the values of your keys periodically. This topic describes these aspects of security:

- Controlling access to utility programs - KGUP, CSFDUTIL
- Controlling access to the callable services
- Controlling access to cryptographic keys
- Controlling access to tokens
- Scheduling changes for cryptographic keys
- Controlling access to panel functions
- Controlling access to RACF SMF log records

Controlling the program environment

Some programs or applications which use ICSF require that the environment be program controlled. In a program controlled environment, programs within the address space are defined to the Security Server (RACF). Defining a program to RACF requires the program name and the name of the data set which contains the program.

The commands to define the ICSF load module to RACF are:

```
RDEFINE PROGRAM * ADDMEM('CSF.SCSFMOD0'//NOPADCHK) UACC(READ)
SETROPTS WHEN(PROGRAM) REFRESH
```

Additional details on program control may be found in the "Program Control" topic of the *z/OS Security Server RACF Security Administrator's Guide*.

Controlling access to KGUP

Anyone running the key generator utility program can read and alter an unprotected cryptographic key data set (CKDS). Therefore, only authorized users should have access to the key generator utility program. To make it difficult for an unauthorized person to execute the key generator utility program, store the program in an APF-authorized library that is protected by the Security Server (RACF). Additionally, a security administrator can define a CSFKGUP profile in the CSFSERV class and permit or deny users access to the utility.

Controlling access to CSFDUTIL

CSFDUTIL reads through a CKDS or PKDS and generates a report for duplicate secure key tokens. Only authorized users should have access to the CSFDUTIL utility program. To make it difficult for an unauthorized person to execute the CSFDUTIL utility program, store the program in an APF-authorized library that is protected by the Security Server (RACF). Grant ICSF administrators access to the CSFDUTIL resource in the CSFSERV class.

Controlling access to the callable services

Unauthorized persons should not perform the cryptographic or key management functions that the callable services provide. The security administrator should be the only one able to access some services like those used in managing keys. The security administrator can give access to some services, such as enciphering and deciphering data, to persons who are authorized on the system.

You can use the Security Server (RACF) to control which users can use ICSF callable services. For example, you can use the key export service to export any type of key. Your installation may want only the security administrator to be able to use the key export function.

ICSF provides security exit points that you can use to control access to a callable service. (In ICSF/MVS Version 2 Release 1 the IBM-supplied security exit routines were removed, but the exit points still remain.) For information about the security exit points, see “Security installation exits” on page 143.

Your installation may want other users to just be able to export data keys, because sending encrypted data between systems is a common function. The data key export callable service permits the export of data keys only. Your security administrator can have access to the key export service and can use the Security Server (RACF) to give other users access to the data key export service. For more information on controlling who can use ICSF callable services, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

Access control points for specific functions may be enabled/disabled through the TKE workstation. See the*z/OS Cryptographic Services ICSF TKE Workstation User's Guide* for additional information.

Controlling access to cryptographic keys

Besides the key generator utility program and services, your installation should also control access to the cryptographic keys. First, it is highly recommended that you store cryptographic keys in data sets that are protected by RACF or an equivalent product. You should limit access to authorized persons or applications. Second, you can use RACF to control access to keys in the in-storage cryptographic key data set. For more information on protecting cryptographic keys, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

When clear DES or AES keys are added to the CKDS, RACF-protect all clear keys by label name on all systems sharing the CKDS.

ICSF also provides security exit points that you can use to control access to keys in the in-storage CKDS and in the PKDS. For information about the security exit points, see “Security installation exits” on page 143.

Controlling access to secure key tokens

You and your installation have the option of controlling access to a secure tokens that have the same token value and different key labels. To do this, define a key store policy. Key store policy are a system wide setting, using RACF profiles to define the policy. Because key store policy makes use of additional RACF checks, careful planning should occur before implementing the support.

For details on key store policy, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

Scheduling changes for cryptographic keys

You should periodically change the value of cryptographic keys to reduce the possibility of exposing a key value. It is recommended that you change the DES or AES master key at least every 12 months.

The security administrator can use the key generator utility program (KGUP) to change the cryptographic keys. KGUP updates keys in the disk copy of the cryptographic key data set while the callable services access keys in the in-storage copy of the cryptographic key data set. Therefore, you can change the keys without affecting cryptographic operations. For more information on using KGUP, refer to *z/OS Cryptographic Services ICSF Administrator's Guide*.

Controlling access to administrative panel functions

You can perform many ICSF administration functions by using the TSO panels. RACF can protect access to these functions. The functions include:

- Refreshing the CKDS
- Setting the master key
- Changing the master key

Additionally, for S/390 Enterprise Servers, S/390 Multiprise servers, and IBM @server zSeries, these functions are also protected:

- Clear key entry (access can also be controlled through the TKE workstation, domain controls)
- Pass phrase MK/KDS initialization
- Administrative control functions (enabling and disabling dynamic CKDS access, PKA callable services, PKDS read access, and PKDS write, create, and delete access)

These functions are treated the same way as callable services. See 'Viewing and Changing System Status' in the *z/OS Cryptographic Services ICSF Administrator's Guide*, SA22-7521 for more information.

Obtaining RACF SMF log records

For information on how to capture SMF log records for RACF access events, see *z/OS Security Server RACF Auditor's Guide* and *z/OS Security Server RACF Command Language Reference*.

You can extract RACF log records from the SMF data set that can be correlated to the ICSF log records. For more information on how to obtain RACF log records from the SMF data set, see *z/OS Security Server RACF Auditor's Guide*.

Debugging Aids

This topic contains information you can use when diagnosing problems on ICSF. This topic describes:

- Component trace
- Abnormal endings
- Using the IPCS formatting routine
- Detecting ICSF serialization contention conditions

Component Trace

The ICSF component trace is written to a single buffer that is addressed from the ICSF CCVE. This buffer contains the number of entries you specify in the installation options data set TRACEENTRY parameter. If you do not specify this installation option, the default is 10000. Each entry is 96 bytes long, and the maximum number of entries allowed is 500,000. When the buffer is full, the trace is wrapped.

In addition to the service and exit trace entry information that is always provided, you can also activate ICSF component tracing for:

- instruction trace entries
- Sysplex CKDS trace entries
- Sysplex PKDS trace entries
- Sysplex TKDS trace entries

To turn tracing on for all these additional entries, use the TRACE ON command:

```
TRACE CT,ON,COMP=CSF
```

Follow the TRACE ON command with this reply:

```
R nn,END
```

To deactivate tracing for these additional entries, use this TRACE OFF command:

```
TRACE CT,OFF,COMP=CSF
```

Examining the Trace Entry Buffer

To examine the trace buffer, use the CTRACE facility of the Interactive Problem Control System (IPCS) in either batch or online mode.

CSF TRACE Common Header: These items are present in every trace entry:

ASCB@ Address of the (application) ASCB
TCB@ Address of the (application) TCB
ASID Application's address space identifier

These items are omitted in the IPCS trace SUMMARY output, and the type-specific data appears after this common header.

Service and Exit Trace Entry Types: ICSF component trace is always active, and these types of trace entries are always written to the buffer:

MISC: Miscellaneous ICSF internal call
BSERVICE: Before the call to service
ASERVICE: After the call to service
BEXIT: Before the call to exit
AEXIT: After the call to exit

Type-Specific Data for Misc Trace Entries: These items are traced for MISC entries:

Module: Name of the ICSF module which issued this entry
X'8': Returned PSMID
X'4A': CCPA pointer
X'4B' A 4 byte field.
ID: Internal ICSF identifier

These items appear in both IPCS SUMMARY and IPCS FULL output.

Type-Specific Data for Service Trace Entries: These items are traced for BSERVICE and ASERVICE entries:
Module: Name of the service called
Rcode: Return code from the service
Reason: Reason code from the service

Rcode and Reason are meaningless for BSERVICE.

These items appear in both IPCS SUMMARY and IPCS FULL output, and the exit trace entries are not called.

Instruction Trace Entry Types: If you have activated ICSF component tracing, instruction trace entries are written to the buffer in addition to the service and exit trace entry information that is always provided:
BCRYPTO: Before the cryptographic instruction
ACRYPTO: After the cryptographic instruction

Type-Specific Data for Instruction Trace Entries: These items are traced for BCRYPTO and ACRYPTO entries:
GPRnn: General Registers 0–13
ARnn: Access Registers 3 and 8
Instruction: The cryptographic instruction

These items appear in both IPCS SUMMARY and IPCS FULL output.

The precise meanings of the register differ for each cryptographic instruction. Indeed, the registers GPR10-GPR13 are not used by any cryptographic instruction. However, the more common registers are:

GPR00 Function called (for example, for CMD, encipher or decipher).

GPR01 Cryptographic status (only valid for ACRYPTO).

GPR02 Address of the local instruction parameter block. The length and usage of the parameter block differ from instruction to instruction and the usage from function to function within the instruction.

GPR03 Address of output text when this is of variable length (for example, ciphertext for encipher command).

GPR08 Address of input text when this is of variable length (for example, plain text for encipher command).

AR03 Access Register 3 (only useful if GPR03 is useful).

AR08 Access Register 8 (only useful if GPR08 is useful).

Sysplex CKDS Entry Types: If you have activated ICSF component tracing, these trace entries are written to the buffer in addition to the service and exit trace entry information that is always provided.
XCFMSGS: Traces the Send of an XCF message relating to an update to a CKDS in a sysplex

XCFMSGR: Traces the Receipt of an XCF message relating to an update to a CKDS in a sysplex
XCFENQX: Traces the return of control to the CKDS I/O subtask following the request for an exclusive ENQ on the SYSZCKDS.ckdsdsn resource. (The difference in the timestamp values between the XCFENQX entry and the XCFMSGS entry can be used to determine the delay due to multi-system effects of using the SYSPLEXCKDS option for sysplex-wide consistency of CKDS data.)

Type-Specific Data for Sysplex CKDS Trace Entries: These items are traced for XCFMSGS, XCFMSGR, and XCFENQX entries:
ASCB: ASCB address
TCB: TCB address
ASID: Address Space ID
GPRnn: General Registers 0-12
GPRLEN: The length of the General Registers (8 for AMODE(64) operation, 4 otherwise)
CSS: Address of ICSF Cross-System Services (CSS) block
SYSID: The System ID of the system originating the XCF message
MsgInfo: The message type and CKDS action (create, update, delete)
IOStatus: The status of the CKDS I/O operation

These items appear in both IPCS SUMMARY and IPCS FULL output.

Sysplex PKDS Entry Types: If you have activated ICSF component tracing, these trace entries are written to the buffer in addition to the service and exit trace entry information that is always provided.
XCFPMSGS: Traces the broadcast of an XCF message related to PKDS I/O.
XCFPMSGR: Traces the Receipt of an XCF message relating to an update to a PKDS I/O.
XCFPENQ: Traces the return of control to the PKDS I/O subtask following the request for an exclusive ENQ on the SYSZTKDS.PKDSdsn resource.

Type-Specific Data for Sysplex PKDS Trace Entries: These items are traced for XCFTMSGS, XCFTMSGR, and XCFTENQ entries:
ASCB: ASCB address
TCB: TCB address
ASID: Address Space ID
GPRnn: General Registers 0-12
GPRLEN: The length of the General Registers (8 for AMODE(64) operation, 4 otherwise)
CSS: Address of ICSF Cross-System Services (CSS) block

These items appear in both IPCS SUMMARY and IPCS FULL output.

Sysplex TKDS Entry Types: If you have activated ICSF component tracing, these trace entries are written to the buffer in addition to the service and exit trace entry information that is always provided.
XCFTMSGS: Traces the broadcast of an XCF message related to TKDS I/O.
XCFTMSGR: Traces the Receipt of an XCF message relating to an update to a TKDS I/O.
XCFTENQ: Traces the return of control to the TKDS I/O subtask following the request for an exclusive ENQ on the SYSZTKDS.TKDSdsn resource.

Type-Specific Data for Sysplex TKDS Trace Entries: These items are traced for XCFTMSGS, XCFTMSGR, and XCFTENQ entries:

ASCB: ASCB address
TCB: TCB address
ASID: Address Space ID
GPRnn: General Registers 0-12
GPRLEN: The length of the General Registers (8 for AMODE(64) operation, 4 otherwise)
CSS: Address of ICSF Cross-System Services (CSS) block

These items appear in both IPCS SUMMARY and IPCS FULL output.

Abnormal Endings

ICSF has an abnormal ending in these cases only:
- When an error occurs during ICSF initialization
- When you specify FAIL(ICSF) in the callable service exit installation option
- When the setting of a cryptographic domain index fails

If an abnormal end occurs in any other cases, your application or unit of work ends; however, ICSF is still available.

ICSF has an abnormal end code unique to ICSF. Errors specific to ICSF result in an abnormal end code of X'18F' and a unique reason code. In general, all abnormal ends occurring within ICSF result in an appropriate system dump, user dump, or LOGREC recording.

Review the reason code to see if the abnormal end was an installation or user error. For a list of the reason codes for abnormal end code X'18F', refer to *z/OS MVS System Codes*. If you cannot resolve the problem, save the dump and contact the IBM Support Center.

IPCS Formatting Routine

There is a CTrace filter exit for ICSF. You can now issue these IPCS commands:

```
CTRACE  COMP(CSF)  OPTIONS((COUNTS,FAILURES))
CTRACE  COMP(CSF)  OPTIONS((COUNTS))
CTRACE  COMP(CSF)  OPTIONS((FAILURES))
```

COUNTS
Produces a list of services called and how often they were called.

FAILURES
Produces output for each failed ICSF service trace entry.

There is a formatter for ICSF called CSFDATA. It is an IPCS VERBEXIT. To run it, enter:

```
VERBX  CSFDATA  'options'
```

The supported options are:
- CELL
- CCPV
- CCPP
- CCPA
- CCPS

- CACB
- CCPD

If no options are specified you get VERBX CSFDATA Output:

```
No valid options were specified on VERBX CSFDATA.
Valid options are CELL,CCPV,CCPP,CCPA,CCPS,CACB,CCPD
```

Sample output:

```
COMPONENT TRACE FULL FORMAT
COMP(CSF)
OPTIONS((FAILURES))
**** 08/15/2006

SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  -------------

SYSAK     ASERVICE  00000006  12:58:13.679197  After  call to service

         ASCB@.... 00FCD980  TCB@..... 007BCA38  AS_id.... 0022
         Module... CSFNENC   RCode.... 00000008  Reason... 0000002F
         User..... 00000000  00000000  00000000  00000000  00000000  00000000
                   00000000  00000000  00000000  00000000  00000000  00000000
                   0000

SYSAK     ASERVICE  00000006  13:07:02.162950  After  call to service

         ASCB@.... 00FCD980  TCB@..... 007BCA38  AS_id.... 0022
         Module... CSFNSYI   RCode.... 00000008  Reason... 00000042
         User..... 00000000  00000000  00000000  00000000  00000000  00000000
                   00000000  00000000  00000000  00000000  00000000  00000000
                   0000


=================================================================
  COMPONENT TRACE FULL FORMAT
  COMP(CSF)
  OPTIONS((COUNTS))


 ICSF COUNTS FROM CTRACE:
 SERVICE CALLS_FOUND = 00000070
 FAILING SERVICES    = 00000035
 SERVICE   #SUCCESS   #FAILED
 CSFSSTSW  00000001   00000000
 CSFNPCV   00000001   00000000
 CSFNPCM   00000001   00000000
 CSFSTRL   00000002   00000000
 CSFNENC   00000003   00000025
 CSFNDEC   00000017   00000000
 CSFNSYI   00000003   00000004
 CSFNPKD   00000007   00000006


=====================================================================
COMPONENT TRACE FULL FORMAT
COMP(CSF)
OPTIONS((COUNTS,FAILURES))
**** 08/15/2006

SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  -------------

SYSAK     ASERVICE  00000006  12:58:13.679197  After  call to service

         ASCB@.... 00FCD980  TCB@..... 007BCA38  AS_id.... 0022
```

```
         Module... CSFNENC   RCode.... 00000008  Reason... 0000002F
         User..... 00000000  00000000  00000000  00000000  00000000  00000000
                   00000000  00000000  00000000  00000000  00000000  00000000
                   0000

SYSAK     ASERVICE  00000006  13:11:04.023615  After  call to service

         ASCB@.... 00FCD980  TCB@..... 007BCA38  AS_id.... 0022
         Module... CSFNPKD   RCode.... 00000008  Reason... 00000041
         User..... 00000000  00000000  00000000  00000000  00000000  00000000
                   00000000  00000000  00000000  00000000  00000000  00000000
                   0000

ICSF COUNTS FROM CTRACE:
SERVICE CALLS_FOUND = 00000010
FAILING SERVICES    = 00000002
SERVICE   #SUCCESS   #FAILED
CSFNENC   00000003   00000001
CSFNPKD   00000007   00000001


============================================================

Sample output of ?CSFTRCE TYPE(MISC)

COMPONENT TRACE FULL FORMAT
COMP(CSF)
**** 08/15/2006

SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  -------------


SYSAK     MISC      00000013  15:20:29.471700  Miscellaneous

         ASCB@.... 00F58100  TCB@..... 007D4A70  AS_id.... 001E
         Module... CSFACCPD  HEX8..... 00297D54  F800804D
         Hex4A.... 7F44B3B0  Hex4B.... 00E7F1F0  ID....... X10
SYSAK     MISC      00000013  15:20:29.471702  Miscellaneous

         ASCB@.... 00F58100  TCB@..... 007D4A70  AS_id.... 001E
         Module... CSFACCPD  HEX8..... 00297D54  F800804D
         Hex4A.... 7F44B468  Hex4B.... ACD7F0F1  ID....... P01
SYSAK     MISC      00000013  15:20:29.471731  Miscellaneous

         ASCB@.... 00FB1780  TCB@..... 007D54F8  AS_id.... 0029
         Module... CSFGCCPN  HEX8..... 00297D54  F800804D
         Hex4A.... 7F44B468  Hex4B.... B8D7F0F8  ID....... P08

SYSAK     MISC      00000013  15:20:29.476769  Miscellaneous

         ASCB@.... 00FB0D00  TCB@..... 007D9138  AS_id.... 0030
         Module... CSFASEND  HEX8..... 00307D91  3800804F
         Hex4A.... 7F44B3B0  Hex4B.... 00E7F1F1  ID....... X11
SYSAK     MISC      00000013  15:20:29.476769  Miscellaneous

         ASCB@.... 00FB0D00  TCB@..... 007D9138  AS_id.... 0030
         Module... CSFASEND  HEX8..... 00307D91  3800804F
         Hex4A.... 7F44B468  Hex4B.... A4D7F1F0  ID....... P10
```

You can use the Interactive Problem Control System (IPCS) to format and display the certain ICSF control blocks. The IPCS CBFORMAT command displays the control block's eye-catcher name, its location in the address space, and its field names with their offsets. You specify a symbol with the command to identify the

control block. Table 7 lists the control blocks you can display, the symbol IPCS recognizes for each control block, and a reference for the control block format.

Table 7. IPCS Symbols and Format References for the ICSF Control Blocks

Control Block	Symbol	Format Reference
Installation-defined Service Table	CSFMGST	Varies for each installation.
CSF Exit Name Table	CSFENT	See Table 13 on page 121.
Cryptographic Communication Vector Table	CSFCCVT	See Table 88 on page 267.
Cryptographic Communication Vector Table Extension	CSFCCVE	See Table 89 on page 273.
Secondary Parameter Block	CSFASPB	See Table 17 on page 131.

For example, to format and display the ICSF Exit Name table issue this command:

```
CBFORMAT CSFENT
```

Instead of using a symbol to identify the control block, you can provide an address. Find and specify the address of the control block in the address space at the time of the dump. When you specify an address, you must also specify the STRUCTURE keyword with the control block symbol.

Note: To format the secondary parameter block, you must provide an address to identify the control block.

For example, if the address of the secondary parameter block is F632D0, issue this command to format the secondary parameter block.

```
CBFORMAT F632D0. STRUCTURE(CSFASPB)
```

In the example, the secondary parameter block is located at address F632D0 in the address space at the time of the dump. On the command, you must put a period after the address. With this control block, you also specify the structure keyword with the symbol CSFASPB.

For more information about using the CBFORMAT command, see *z/OS MVS IPCS User's Guide*.

Detecting ICSF Serialization Contention Conditions

If a user task or address space holds an ENQ or latch for an extended period of time, it is likely hung and needs to be cancelled so that other work can obtain the ENQ or latch. Some applications might provide controls or document procedures for addressing situations in which the application appears to be gating the rest of the workload. The ICSF system programmer should consult the application's system programmer or administrator regarding actions to take for or against the application. Such action could include stopping or canceling the application.

ICSF requires Global Resource Serialization (GRS) ENQ resources to manage concurrent operations involving the key data sets (CKDS, PKDS and TKDS), and the ICSF ENQ scheme has ICSF itself obtaining any necessary data set ENQ, in a proxy fashion, on behalf of an application unit of work driving an ICSF API request requiring an ENQ. ICSF also manages any set of additional, different application requests that may be waiting for that same ENQ resource. For this reason, GRS always perceives only ICSF as a key data set ENQ resource owner or waiter, and a DISPLAY GRS,CONTENTION command would not illustrate key data set ENQ

contention between two or more competing application requests within a single system scope. For sysplex scope ENQ contention, DISPLAY GRS,CONTENTION would, without any internal assistance, illustrate only ICSF itself as an ENQ holder or waiter, and would not reflect any client application identity or information associated with ICSF's ENQ resource usage.

ICSF provides an internal capability to embellish the DISPLAY GRS command output to illustrate the ICSF client applications for which ICSF is holding an ENQ resource, and on the general conditions involving client waiters for an ENQ resource. This enhanced capability is transparently provided and requires no additional ICSF or GRS installation or configuration action. The ICSF support to enhance the DISPLAY GRS output is relevant on a DISPLAY GRS,CONTENTION command only if GRS can detect contention, which is not the case when two or more ICSF client application requests are competing for the same ENQ resource within a single system scope. The ICSF support is relevant on a DISPLAY GRS,RES=(*qname-rname*) command whenever the ENQ resource specified in the *qname-rname* option is currently held, regardless of whether or not contention exists. For this reason, the DISPLAY GRS,RES=() command version is recommended as the reliable technique for obtaining information about ICSF key data set ENQ serialization conditions. The DISPLAY GRS command syntax for the various ICSF key data set ENQ resources can be summarized as follows:

Table 8. DISPLAY GRS command syntax ICSF key data set ENQ resources

This command:	Displays ENQ information for the:
DISPLAY GRS,RES=(SYSZCKT.*)	CKDS
DISPLAY GRS,RES=(SYSZPKT.*)	PKDS
DISPLAY GRS,RES=(SYSZTKT.*)	TKDS

Here is sample command output for the DISPLAY GRS,RES=(SYSZCKT.*) command:

```
ISG343I 12.01.33 GRS STATUS 360
S=SYSTEM  SYSZCKT  SYSZCKT
SYSNAME        JOBNAME         ASID     TCBADDR   EXC/SHR

SY1       CSFJM70 /APPL107   0040/0045  007D8E88 EXCLUSIVE

ADDITIONAL RESOURCE INFORMATION FROM:  ICSF Managed ENQ
Owner: APPL107  TTOKEN: 00000120000000030000000003007FF050 Waiters: 005
```

In this example, the display command result illustrates that ICSF on system SY1 started under jobname CSFJM70 and executing in ASID 40, has obtained the CKDS ENQ resource exclusively on behalf of the client application running with a jobname of APPL107 and executing in ASID 45. Furthermore, the APPL107 application unit of work that caused ICSF to obtain this ENQ was the task identified by task token 00000120000000030000000003007FF050, and there are five additional application requests on system SY1 that are awaiting access to this ENQ resource.

The DISPLAY GRS,RES=() command must be executed on (or routed to) all of the systems within the scope of a sysplex to obtain the comprehensive understanding of an ICSF key data set ENQ resource.

ICSF also exploits Global Resource Serialization (GRS) latches for serializing resources that are managed within the scope of a single system. In the case of ICSF latches, whenever a client application request requires an ICSF latch for serialization, the latch is obtained under the application's unit of work (not proxied

like the ENQ), and therefore the DISPLAY GRS,CONTENTION command will always illustrate the application information for the current latch owner(s).

The following operational steps are recommended when ICSF serialization contention is suspected as a cause for a workload slowdown or hang:

1. Issue the DISPLAY GRS,CONTENTION command to illustrate sysplex scope contention on ICSF ENQ serialization resources, or system level contention on ICSF latch serialization resources. If the command result demonstrates latch contention, go to step 3. If the command result demonstrates ICSF key data set ENQ contention and discloses the ENQ owner client application information, go to step 3. If the command result does not demonstrate contention, or does not disclose the ENQ owner client application information, proceed to the next step.
2. Issue the following commands as needed (depending on the key data sets you are using):

   ```
   DISPLAY GRS,RES=(SYSZCKT.*)
   DISPLAY GRS,RES=(SYSZPKT.*)     Issue this command only if you are utilizing a PKDS
   DISPLAY GRS,RES=(SYSZTKT.*)     Issue this command only if you are utilizing a TKDS
   ```

   The commands need to be executed either on all systems within a sysplex, or on the local system where the ENQ resource is known to be owned. The command result should disclose the ENQ owner client application information.
3. Initiate an action for or against the client application to end the unit of work on behalf of which ICSF has obtained the ENQ resource. Such action could include stopping or canceling the application.

Chapter 5. Installation Exits

Your installation can define exit routines to supplement the Integrated Cryptographic Service Facility (ICSF), the key generator utility program (KGUP), and the PCF conversion program. Exit routines are programs that programmers at your installation write to allow you to "customize" an application. Your installation may need to perform specific functions with the data that your cryptographic application manipulates. At various points in processing, ICSF, KGUP, and the PCF conversion program release control to an exit routine.

Some common uses for installation exits include:
- Identifying and verifying users
- Accessing alternate data sets
- Manipulating input commands
- Manipulating output data

This topic describes the various types of exit points in ICSF and the functions that your exits can perform.

Attention: Only an experienced system programmer should use the ICSF installation exits. Writing an exit routine and installing a new exit are tasks that require a thorough knowledge of system programming in an OS/390 and z/OS environment. An unknowledgeable programmer who attempts to write exit routines or to install new exit points, runs the risk of seriously degrading the performance of your system and causing complete system failure.

Types of exits

ICSF provides several types of exit points:
- Exits that are called during initialization, stopping, and modification of ICSF itself, which are known as the mainline exits
- Exits that are called from the services
- An exit called from the PCF conversion program
- An exit called when you update the CKDS with a key that is entered through the key entry hardware or during conversion program processing
- An exit called when records are retrieved from the in-storage CKDS
- Security exits that are called during initialization and stopping of ICSF, during a call to a service, and when accessing a CKDS entry
- An exit called at various points during KGUP processing

These topics briefly describe the different types of exits available in ICSF.

Note: Although IBM no longer supplies security exit routines, the exit points still remain.

Mainline exits

You can supply three exits that are called during ICSF initialization. You can also define an exit routine to run after an operator issues the STOP command and another exit to run after the MODIFY command. Thus, mainline exits can run at these five different points:
- Initialization points
  - Before ICSF initialization

  - After ICSF reads and interprets the installation options
  - Before the completion of ICSF initialization
- When an operator issues a STOP ICSF command
- When an operator issues a MODIFY ICSF command

You can use a mainline exit to alter values in the Cryptographic Communication Vector Table, to end ICSF, or to change ICSF installation options. For more information about the mainline exits, see “Mainline installation exits” on page 116.

Exits for the services

Each of the services in ICSF calls an exit before and after processing. *z/OS Cryptographic Services ICSF Application Programmer's Guide* describes the services in greater detail.

You can use a service exit to change, augment, or replace processing or to bypass the IBM-supplied processing for the service entirely. “Services installation exits” on page 123 gives further details about exits for the services.

The PCF CKDS conversion program exit

The PCF conversion program changes a CKDS from PCF to ICSF CKDS format. See Chapter 8, “Migration from PCF to z/OS ICSF,” on page 173 for more information about the conversion program.

ICSF provides three exit points for the same exit routine:
- During the initialization of the conversion program
- While the conversion program is processing individual records
- During the ending of the conversion program

See “PCF conversion program installation exit” on page 136 for more information about the conversion program installation exit (CSFCONVX).

The Single-record, Read-write exit

Certain ICSF processes read records from or write records to the CKDS. These processes include running a conversion program, refreshing and reenciphering the CKDS, and using the key entry hardware to enter a key. When these processes read or write CKDS records, they call the exit. You can customize the processing of a CKDS record read-write with the single-record, read-write exit (CSFSRRW). See “Single-record, Read-write installation exit” on page 139 for more information about the single-record, read-write exit.

When application programs write records to or read records from the PKDS, ICSF calls the single-record, read-write exit.

Note: This exit is given control only for a fixed-length record CKDS. The exit does not work with the variable-length record format of the CKDS.

The cryptographic key data set entry retrieval exit

You can use certain services to manage keys on ICSF. A service can access a key in the in-storage CKDS by specifying a key label. For more information about the services, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

When a service requests a record from the in-storage CKDS by label, ICSF calls the CKDS entry retrieval exit. For instance, you can use this exit to perform a

specific search of the installation data field in the record. See “Cryptographic key data set entry retrieval installation exit” on page 134 for more information about the CKDS entry retrieval exit.

Note: This exit is given control only for a fixed-length record CKDS. The exit does not work with the variable-length record format of the CKDS.

Security exits

You can supply four different exits to control access to resources on ICSF. ICSF calls the security exits at these points:

- During CSF initialization
- During CSF termination
- When an application calls an ICSF service
- When an entry in the in-storage CKDS is accessed

See “Security installation exits” on page 143 for more information about the security exits.

The KGUP exit

You use KGUP to generate and maintain keys in the CKDS. KGUP creates key values that systems can use in key exchanges. The ICSF administrator uses job control language to start KGUP and specifies information to KGUP through the use of a control statement.

As opposed to the five different mainline exits, ICSF provides one exit for KGUP processing that is called at four different points. ICSF calls the KGUP exits at these points:

- During KGUP initialization
- Before KGUP processes a key that is identified by a control statement
- Before KGUP updates the CKDS
- During KGUP termination

The KGUP exit receives a parameter that identifies the exit's calling point. Thus, the installation exit can perform different functions at each of the calls.

You can use the KGUP exit to change key values, make a copy of a CKDS entry, or end KGUP. “Key generator utility program installation exit” on page 147 gives a more detailed description of the KGUP exit.

Entry and return specifications

All of the exits described in “Types of exits” on page 111 use standard linkage conventions on entry and return from the exits.

Registers at entry

The mainline exits have these register contents on entry:

Register	Contents
0	Address of the exit parameter block (EXPB)
1	Address of a parameter list
2–12	Not applicable
13	Address of register save area

14	Return address
15	Entry point address

The service exits have these register contents on entry:

Register	Contents
0	Address of the exit parameter block (EXPB)
1	Address of a parameter list
2–13	Not applicable
14	Return address
15	Entry point address

The CKDS entry retrieval installation exit has these register contents on entry:

Register	Contents
0	Not applicable
1	Address of a parameter list
2–12	Not applicable
13	Address of register save area
14	Return address
15	Entry point address

The conversion program, single-record, read-write, and KGUP exits have these register contents on entry:

Register	Contents
0	Not applicable
1	Address of a control block (CVXP, RWXP, or KGXP, depending on the exit)
2–12	Not applicable
13	Address of register save area
14	Return address
15	Entry point address

The particular control blocks that are passed through register 0 or register 1 are described with each exit.

Registers at return

Registers for all exits must contain the original contents on entry with the exception of register 15 which must contain a valid return code. See each exit for a list of valid return codes. The registers should contain this information on return.

Register	Contents
0–14	Same as entry contents
15	Valid return code

Exits environment

ICSF calls different types of exits in distinct environments. The exits differ regarding the mode in which they run and how they address data.

Mainline exits

ICSF mainline exits run in task mode in the ICSF address space. All the passed storage pointers specify addresses in the ICSF address space and are not ALET qualified. There are essentially no restrictions on the use of z/OS services for these exits.

service exits

ICSF calls the service exits in cross memory mode after a space switch PC. The exits run in the ICSF address space, which is the primary address space. The exits need to address parameters in the caller's address space, which is the secondary address space. In general, user-passed parameters, including the parameter list itself, are in the secondary address space. An exit that is running in access register (AR) mode using an ALET of 1 can access these parameters. For information about cross memory mode and AR mode, see *z/OS MVS Programming: Extended Addressability Guide*.

CKDS entry retrieval exit

The exit runs in cross memory mode. The addresses of the CKDS records that are used by the exit are ALET-qualified. The exit receives both the current CKDS record address and the record's associated ALET as parameters in the exit parameter list. The exit must run in AR mode, and must use the information passed in the exit parameter list to access CKDS entries. For information about cross memory mode and AR mode, see *z/OS MVS Programming: Extended Addressability Guide*.

KGUP, Conversion Programs, and Single-record, Read-write exits

The exits run in task mode in the caller's home address space. The exits do not run in cross memory mode and are not passed ALET-qualified storage pointers. There are essentially no restrictions on the use of z/OS services for these exits.

Security exits

The initialization and termination security exits run in task mode in the ICSF address space. The passed storage pointers specify an address in the ICSF address space and are not ALET-qualified. There are essentially no restrictions on the use of z/OS services for these exits.

ICSF calls the security service exit and the security keys exit in cross memory mode after a space switch PC. The security service exit runs in the ICSF address space, which is the primary address space. The security key exit runs in cross memory and AR mode.

Exit recovery

An ESTAE routine provides recovery for the mainline exits; the single-record, read-write exit; and the security initialization and termination exits. If an exit ends abnormally, the ESTAE routine intercepts the abnormal ending code and schedules a system dump. If the conversion program exit ends abnormally, the conversion

program ends abnormally. If the KGUP exit ends abnormally, KGUP also ends abnormally. ESTAE routines provide recovery for the conversion program and KGUP.

The ICSF Functional Recovery Routine (FRR) provides recovery for the service exits, the CKDS entry retrieval exit, and the security service and key exits. If an exit ends abnormally, the FRR intercepts the abnormal ending code and schedules a system dump.

There are times during ICSF processing that ICSF suppresses dumps. For example, ICSF does not schedule dumps when integrity checking user data. This action avoids the possibility of user errors that can severely affect system performance. However, ICSF does write a record to SYS1.LOGREC if the error occurs.

When writing exits, you may also want to suppress dumps under certain circumstances. You can suppress dumps by setting a bit on in the SPB. This bit, the SPBTERM bit, is the third bit of the flag byte at offset 18 in the SPB. An exit might want to suppress dumps whenever the exit writes user storage. The exit can turn the bit on before the WRITE instruction and turn the bit off again after the instruction.

Mainline installation exits

ICSF begins when an operator issues a START command from the operator console. When ICSF issues this command, the initialization process begins.

After ICSF starts, operators can issue the MODIFY or STOP commands. You can define installation exits to customize ICSF at the initialization, stopping, and modification points.

Purpose and use of the exits

ICSF calls the mainline exits during the startup, modification, and shutdown stages. The exits allow your installation to change the initialization options, issue special messages, and bypass operator commands. This is a description of each point at which ICSF calls mainline exit routines.

CSFEXIT1

ICSF calls this exit after an operator issues a START command, but before any processing takes place. You can use this exit to change the allocation of the installation options data set.

ICSF always calls the exit. If this exit does not exist, ICSF continues normal processing. If this exit exists, ICSF starts it.

CSFEXIT2

ICSF calls this exit during the initialization process after the installation options data set is read and interpreted. You can use this exit to change certain installation options.

CSFEXIT3

ICSF calls this exit just before ICSF initialization is complete. You can use this exit to issue commands to start other cryptographic work.

CSFEXIT4

ICSF calls this exit when an operator issues a STOP command. You can use this exit to decide to allow or disallow the STOP command.

CSFEXIT5

CSFEXIT5 receives the command input block (the string that is entered by the operator), so you can customize CSFEXIT5 to perform any processing you require. ICSF calls this exit when an operator issues a MODIFY command. ICSF provides the MODIFY command exit to allow each installation the flexibility of defining its own command. ICSF does no processing when an operator uses the MODIFY command. The MODIFY command is simply a call to CSFEXIT5.

Environment of the exits

The exits receive control with these characteristics:

- Supervisor state
- Key 0
- APF-authorized
- TCB mode
- Address Space Control mode=access register mode
- AMODE(31) orAMODE(64)

  The exit receives control in AMODE(64) if the service was invoked in AMODE(64); otherwise the exit receives control in AMODE(31). If you have a callable service exit for a service which supports invocation by an AMODE(64) caller, once HCR7720 is installed, you should recode your exit to be sure it can handle being invoked in AMODE(64).
- RMODE(ANY)

The exits can change the characteristics during their processing. However, the exits must return to ICSF with the same characteristics as on entry.

Installing the exits

Because ICSF calls CSFEXIT1 before any initialization occurs, the exit is not defined in the same way as the other exits. For all the mainline exits, install the load module that contains the exit into an APF-authorized library. ICSF uses this normal OS/390 search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

You must define CSFEXIT2, CSFEXIT3, CSFEXIT4, and CSFEXIT5 in the installation options data set. However, you *must not* define CSFEXIT1 in the installation options data set, and the load module name for the exit must be CSFEXIT1.

To define the exits in the installation options data set, define the ICSF exit point name and load module name on the EXIT keyword in the installation options data set. For information about the installation options data set, see “Parameters in the installation options data set” on page 38. The EXIT keyword has this syntax:

EXIT (*ICSF exit point name*, *load module name*, **FAIL (***options*))

The **ICSF exit point name** portion of the keyword refers to the ICSF name for each exit, CSFEXIT2, CSFEXIT3, CSFEXIT4, and CSFEXIT5. The **load module name** is the name of the load module that contains the exit. The name can be any valid

name your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded. The valid FAIL options are:

NONE Initialization continues even if exits cannot be loaded.
SERVICE Initialization continues even if exits cannot be loaded.
EXIT Initialization continues even if exits cannot be loaded.
ICSF End ICSF if exits cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, abnormally ends, and generates an SVC dump when attempting to load the exit.

Input

All mainline exits receive the address of an exit parameter block (EXPB) passed in register 0. Each exit receives the address of an address list passed in register 1. Each address in the list points to a parameter.

Figure 7 illustrates the contents of register 0 and EXPB for the mainline exits.

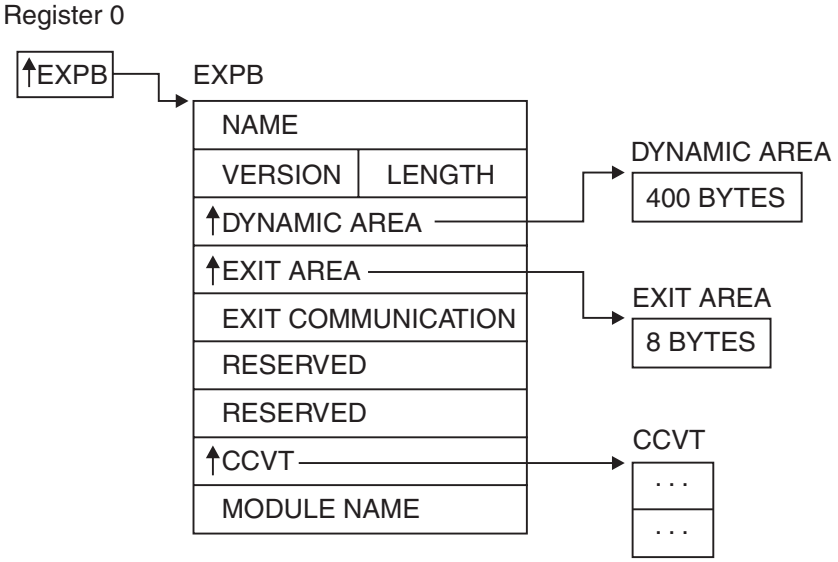

Figure 7. EXPB Control Block for Mainline Exits

Both the mainline exits and the services exits receive the address of EXPB in register 0. Some of the fields in EXPB are used only by the service exits and are reserved fields for the mainline exits.

The Exit Parameter Block

Table 9 describes the contents of the exit parameter block.

Table 9. EXPB Control Block Format for Mainline Exits

Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. This field contains the character string EXPB.

Table 9. EXPB Control Block Format for Mainline Exits (continued)

Offset (Dec)	Number of Bytes	Description
4	2	Version. The version of the control block. This field contains the character string `01`.
6	2	Length. The length of the control block. The value of this field is 40 in decimal.
8	4	Dynamic area address. The address of a 400-byte area that the exit can use as a dynamic area.
12	4	Exit area address. The address of an 8-byte area the exits can use to communicate with each other. ICSF does not check or change this field.
16	4	Exit communication area. A character string that can be used for communication between the exits. The field is initialized to zero before CSFEXIT1 is called, and ICSF does not modify this field.
20	4	Flags. **Reserved**. The flag field is used only by the exits for the services. The field contains binary zeros for the mainline exits.
24	4	Secondary parameter block (SPB) address. **Reserved**. The SPB is used only by the exits for the services. The field contains binary zeros for the mainline exits.
28	4	CCVT address. Address of the Cryptographic Communication Vector Table (CCVT). “The Cryptographic Communication Vector Table (CCVT)” on page 266 describes the CCVT in greater detail.
32	8	Module name. The installation exit's load module name. The field contains the value of the load module name you specified on the EXIT keyword in the installation options data set. The field is 8 bytes of characters, and the value is left-justified and padded with blanks.

Parameters

All mainline exits receive an address list that uses standard entry linkage. Register 1 points to the address list. Each address in the list points to a parameter. Tables in the next four topics describe the parameters for each of the mainline exits.

CSFEXIT1: This table describes the parameters for CSFEXIT1:

Table 10. CSFEXIT1 Parameters

Parameter	Number of Bytes	Description
1	8	The data set name (DDNAME) of the installation options data set.
2	Variable	The command input block for the START command. The command control block is mapped by IEZCIB.

When ICSF calls this, the Cryptographic Communication Vector Table exists, but the table is not yet complete.

CSFEXIT2 and CSFEXIT3: Both CSFEXIT2 and CSFEXIT3 receive the same parameters. Table 11 describes these parameters.

Table 11. CSFEXIT2 and CSFEXIT3 Parameters

Parameter	Number of Bytes	Description
1	44	A character string that is the CKDS name specified in the CKDSN installation option.
2	4	A decimal value that is the maximum length permitted for data passed to services specified in the MAXLEN installation option. Beginning with z/OS V1 R2, the MAXLEN parameter may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647). If a value greater than this is specified, an error will result and ICSF will not start.
3	4	ICSF environmental options. **Note:** Do not change bits 2, 4, and 5. Byte 1: **Bit** — **Meaning When Set On** **0** — Special secure mode allowed. **1** — Special secure mode enabled. **2** — Reserved and must be zero. **3** — Key authentication required. **4** — The hardware has gone from active to inactive. **5** — First start of ICSF during this IPL. **6** — Security Sever (RACF) checking required for authorized callers. **7** — PCF coexistence. Bytes 2–4: Reserved
4	4	Address of the exit name table. Table 13 on page 121 describes the exit name table.

CSFEXIT4 and CSFEXIT5: Both CSFEXIT4 and CSFEXIT5 receive the same parameters. Table 12 on page 121 describes these parameters.

Table 12. CSFEXIT4 and CSFEXIT5 Parameters

Parameter	Number of Bytes	Description
1	44	A character string that is the CKDS name specified in the CKDSN installation option.
2	4	A decimal value that is the maximum length permitted for data passed to services specified in the MAXLEN installation option. Beginning with z/OS V1 R2, the MAXLEN parameter may still be specified in the options data set, but only the maximum value limit will be enforced (2147483647). If a value greater than this is specified, an error will result and ICSF will not start.
3	4	ICSF environmental options. **Note:** Do not change bits 2, 4, and 5. Byte 1: **Bit** — **Meaning When Set On** **0** — Special secure mode allowed. **1** — Special secure mode enabled. **2** — Reserved and must be zero. **3** — Key authentication required. **4** — The hardware has gone from active to inactive. **5** — First start of ICSF during this IPL. **6** — Security Server (RACF) checking required for authorized callers. **7** — PCF coexistence. Bytes 2–4: Reserved
4	4	Address of the exit name table. Table 13 describes the exit name table.
5	Variable	The command input block. You can use the IEZCIB mapping macro to map the control block.

The Exit Name Table: The exit name table contains a list of all of the exits and their load module names. Table 13 describes the format of the exit name table.

Table 13. Format of the Exit Name Table

Offset (Dec)	Number of Bytes	Description
0	4	Exit name table ID. The value is always the character string `ENT`.
4	2	Exit name table version. The value is always the character string `01`.
6	2	Length of the exit name table. This value is in decimal.
8	4	Number of entries in the array which is the number of exits ICSF supplies. This value is in decimal.
12	4	Subpool that the exit name table is in.

Table 13. Format of the Exit Name Table (continued)

Offset (Dec)	Number of Bytes	Description
16	4	**Reserved.**
20	4	**Reserved.**
24	4	**Reserved.**
28	4	**Reserved.**
32	8	ICSF exit name 1. This value is a character string.
40	8	Installation load module name 1. This value is a character string.
48	4	Flags. Flag bytes. Only the first two bytes are used; bytes 3 and 4 are reserved. Byte 1: **Bit** — **Meaning When Set On** **0** — Exit has been requested by the installation. **1** — Exit has been loaded. **2** — Exit is active. **3** — If exit fails, end ICSF. **4** — If exit fails, do not call the exit again. **5** — If exit fails, fail the service. **6** — If exit fails, do nothing. **7** — Exit has failed previously. Byte 2: **Bit** — **Meaning When Set On** **0** — The exit should be called. **1** — The exit is available to the installation. **2** — If the security exit fails, fail the service. **3–7** — **Reserved.**
52	4	Address of the exit.
56	4	**Reserved.**
60	4	**Reserved.**
64	8	ICSF exit name 2. This value is a character string.
72	8	Installation load module name 2. This value is a character string.
80	4	Flags. See offset +48 for flag byte definitions.
84	4	Address of the exit.
88	4	**Reserved.**
92	4	**Reserved.**

x	8	ICSF exit name a.
x+8	8	Installation load module name a.
x+16	4	Flags. See offset +48 for flags.
x+20	4	Address of the exit.
x+24	4	**Reserved.**
x+28	4	**Reserved.**

Return Codes

All mainline exits can pass back a return code in register 15. CSFEXIT1, CSFEXIT2, and CSFEXIT3 support these decimal return codes:

Return Code	Description
0	Proceed with initialization.
16	End ICSF.

CSFEXIT4 supports these decimal return codes:

Return Code	Description
0	Proceed with the STOP command.
4	Do not allow the STOP command to proceed.

CSFEXIT5 supports these decimal return codes:

Return Code	Description
0	Continue processing.
4	End ICSF.

Any return codes other than those listed cause ICSF to end abnormally.

Services installation exits

ICSF provides services that you can use to perform various cryptographic functions. Examples of these functions include enciphering and deciphering data, generating and verifying message authentication codes, generating and verifying PINs, and dynamically updating the CKDS and PKDS. You can define an installation exit for each of the services to customize processing. For a detailed description of the services, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Use this general format to request a service:

```
CALL CSNBxxx (
              return_code
             ,reason_code
             ,exit_data_length
             ,exit_data
             ,parameter_5
             ,parameter_6
               .
               .
               .
             ,parameter_N)
```

Table 14 on page 125 lists the ICSF exit names for each of the services. The parameters that the application passes to a service are known as the service parameter list, and the parameters vary from service to service. “Parameters” on page 133 describes the services parameter lists in more detail.

Purpose and use of the exits

Each of the services has an installation exit. Each installation exit for a service has two exit points:

- **The Preprocessing exit point.** This exit point occurs after an application program calls a service, but before the service starts processing. For example, you can use this exit point to check or change the parameters that the application passes on the call, or to end the call. You can also perform additional security checks.
- **The Postprocessing exit point.** This exit point occurs after the service has finished processing, but before the service returns control to the application program. For example, you can use this exit point to check and change the return code from the service or perform cleanup processing.

Environment of the exits

The exits receive control with these characteristics:

- Supervisor state
- Key 0
- APF-authorized
- TCB or SRB mode
- Cross memory mode
- AR mode
- AMODE(31) orAMODE(64)

  The exit receives control in AMODE(64) if the callable service was invoked in AMODE(64); otherwise the exit receives control in AMODE(31). If you have a callable service exit for a service which supports invocation by an AMODE(64) caller, you must recode your exit to be sure it can handle being invoked in AMODE(64) when running with ICSF HCR7720 or later.
- RMODE(ANY)

The exits can change the characteristics during their processing. However, the exits must return to their caller with the same characteristics as on entry.

You must write the exits in assembler, because you are in AR and cross memory mode and the addresses of some of the parameters you may access are ALET-qualified. In particular, parameters passed into a service are in the user's address space which you can access with an ALET of 1.

For information about cross memory and AR mode, see *z/OS MVS Programming: Extended Addressability Guide*.

Installing the exits

You install an exit for a service by installing the load module that contains the exit into an APF-authorized library. ICSF uses this normal search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and the load module name as a value on the EXIT keyword in the installation options data set. For more information about the installation options data set, see “Parameters in the installation options data set” on page 38. The EXIT keyword has this syntax:

EXIT (*ICSF name*, *load module name*, **FAIL (***options*))

The **ICSF name** portion of the keyword refers to the ICSF name for each service exit. Note that the ICSF name for each service exit is the same as its name. Table 14 lists the ICSF names for each of the service exits. Table 15 on page 128 lists the ICSF names for each of the compatibility service exits. The **load module name** is the name of the load module that contains the exit. The name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded or it ends abnormally. The valid FAIL options are:

NONE No action is taken. The exit can be called again and will end abnormally again.

EXIT The exit is no longer available to be called again.

SERVICE The service or program that called the exit is no longer available to be called again.

ICSF ICSF or the key generator utility program or the PCF conversion program is ended, depending on the exit.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit. If the exit ends abnormally, the service call fails regardless of the fail option you specified. Fail options apply only to subsequent requests for the service.

Note: In this table, CSFPKSC (PKSC interface) and CSFPCI (PCI interface), are a part of the product-sensitive programming interface.

Table 14. Services and Their ICSF Names

Service	***ICSF Name***
ANSI X9.17 EDC Generate	**CSFAEGN**
ANSI X9.17 Key Export	**CSFAKEX**
ANSI X9.17 Key Import	**CSFAKIM**
ANSI X9.17 Key Translate	**CSFAKTR**
ANSI X9.17 Transport Key Partial Notarize	**CSFATKN**
Ciphertext Translate	**CSFCTT**
Ciphertext Translate (with ALET)	**CSFCTT1**
CKDS Key Record Create	**CSFKRC**
CKDS Key Record Create2	**CSFKRC2**
CKDS Key Record Delete	**CSFKRD**
CKDS Key Record Read	**CSFKRR**
CKDS Key Record Read2	**CSFKRR2**
CKDS Key Record Write	**CSFKRW**
CKDS Key Record Write2	**CSFKRW2**
Clear Key Import	**CSFCKI**
Clear PIN Encrypt	**CSFCPE**
Clear PIN Generate	**CSFPGN**
Clear PIN Generate Alternate	**CSFCPA**

Table 14. Services and Their ICSF Names (continued)

Service	***ICSF Name***
Control Vector Translate	**CSFCVT**
Coordinated KDS Administration	**CSFCRC**
Cryptographic Variable Encipher	**CSFCVE**
CVV Key Combine	**CSFCKC**
Data Key Export	**CSFDKX**
Data Key Import	**CSFDKM**
Decipher	**CSFDEC**
Decipher (with ALET)	**CSFDEC1**
Decode	**CSFDCO**
Digital Signature Generate	**CSFDSG**
Digital Signature Verify	**CSFDSV**
Diversified Key Generate	**CSFDKG**
ECC Diffie-Hellman	**CSFEDH**
Encipher	**CSFENC**
Encipher (with ALET)	**CSFENC1**
Encode	**CSFECO**
Encrypted PIN Generate	**CSFEPG**
Encrypted PIN Translate	**CSFPTR**
Encrypted PIN Verify	**CSFPVR**
HMAC Generate	**CSFHMG**
HMAC Verify	**CSFHMV**
Key Export	**CSFKEX**
Key Generate	**CSFKGN**
Key Generate2	**CSFKGN2**
Key Import	**CSFKIM**
Key Part Import	**CSFKPI**
Key Part Import2	**CSFKPI2**
Key Test	**CSFKYT**
Key Test2	**CSFKYT2**
Key Test Extended	**CSFKYTX**
Key Translate	**CSFKTR**
Key Translate2	**CSFKTR2**
MAC Generate	**CSFMGN**
MAC Generate (with ALET)	**CSFMGN1**
MAC Verify	**CSFMVR**
MAC Verify (with ALET)	**CSFMVR1**
MDC Generate	**CSFMDG**
MDC Generate (with ALET)	**CSFMDG1**
Multiple Clear Key Import	**CSFCKM**
Multiple Secure Key Import	**CSFSKM**

Table 14. Services and Their ICSF Names (continued)

Service	***ICSF Name***
One Way Hash Generate	**CSFOWH**
One Way Hash Generate (with ALET)	**CSFOWH1**
PCI Interface	**CSFPCI**
PIN change/unblock	**CSFPCU**
PKA Decrypt	**CSFPKD**
PKA Encrypt	**CSFPKE**
PKA Key Generate	**CSFPKG**
PKA Key Import	**CSFPKI**
PKA Key Translate	**CSFPKT**
PKA Key Token Change	**CSFPKTC**
PKA Public Key Extract	**CSFPKX**
PKDS Key Record Create	**CSFPKRC**
PKDS Key Record Delete	**CSFPKRD**
PKDS Key Record Read	**CSFPKRR**
PKDS Key Record Write	**CSFPKRW**
PKSC Interface	**CSFPKSC**
Prohibit Export	**CSFPEX**
Prohibit Export Extended	**CSFPEXX**
Random Number Generate	**CSFRNG**
Random Number Generate Long	**CSFRNGL**
Remote Key Export	**CSFRKX**
Restrict Key Attribute	**CSFRKA**
Retained Key Delete	**CSFRKD**
Retained Key List	**CSFRKL**
Secure Key Import	**CSFSKI**
Secure Key Import2	**CSFSKI2**
Secure Messaging for Keys	**CSFSKY**
Secure Messaging for PINs	**CSFSPN**
SET Block Compose	**CSFSBC**
SET Block Decompose	**CSFSBD**
Symmetric Key Export	**CSFSYX**
Symmetric Key Generate	**CSFSYG**
Symmetric Key Import	**CSFSYI**
Symmetric Key Import2	**CSFSYI2**
Symmetric MAC Generate	**CSFSMG**
Symmetric MAC Generate (with ALET)	**CSFSMG1**
Symmetric MAC Verify	**CSFSMV**
Symmetric MAC Verify (with ALET)	**CSFSMV1**
TR-31 Export	**CSFT31X**
TR-31 Import	**CSFT31I**

Table 14. Services and Their ICSF Names (continued)

Service	***ICSF Name***
Transaction Validation	**CSFTRV**
Transform CDMF Key	**CSFTCK**
Trusted Block Create	**CSFTBC**
User Derived Key	**CSFUDK**
VISA CVV Service Generate	**CSFCSG**
VISA CVV Service Verify	**CSFCSV**

Notes:

1. The alias for the ANSI X9.17 key management services is CSNAxxx.
2. The aliases for the PKA services is CSNDxxx or or CSNFxxx.
3. The aliases for the symmetric key services are CSNBxxx or CSNExxx.

Table 15. Compatibility Services and Their ICSF Names

Compatibility Service	***ICSF Name***
Encipher under Master Key	**CSFEMK**
Generate a key	**CSFGKC**
Import a key	**CSFRTC**
Cipher/Decipher	**CSFEDC**

Input

The installation exit for each service gets the address of the exit parameter block (EXPB) in register 0. ICSF obtains and initializes an EXP for every service call. Figure 8 on page 129 illustrates the contents of register 0, and Table 16 on page 129 illustrates the EXPB for the service exits.

Register 1 contains the address of an address list. Each address in the list points to a parameter. “Parameters” on page 133 describes the service parameter list. The parameters the exit receives are the same parameters that are passed on the call to the service. For more information about the parameters for each service, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

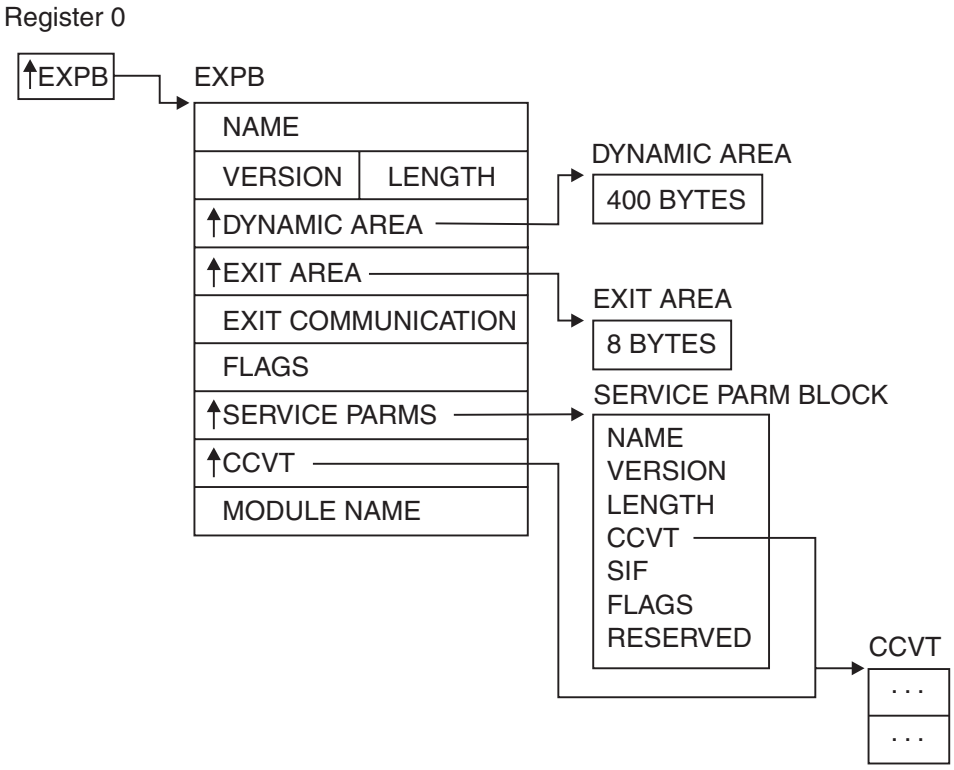

Figure 8. EXPB Control Block in the Service Exits

Exit parameter block

Table 16 describes the contents of the exit control block.

Table 16. EXPB Control Block Format for Services

Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. The field contains the character string `EXPB`.
4	2	Version. The version of the control block. The field contains the character string `01`.
6	2	Length. The length of the control block. The value is 40 in decimal.
8	4	Dynamic area. The address of a 400-byte area that the exit can use as a dynamic area.
12	4	Exit area address. The address of an 8-byte area for the preprocessing and postprocessing invocations of the exit to use for communication. ICSF does not check or change this field.

Table 16. EXPB Control Block Format for Services (continued)

Offset (Dec)	Number of Bytes	Description
16	4	Exit communication area. A character string that can be used for communication between preprocessing and postprocessing invocations of a service exit.
20	4	Flags. A flag byte. Each bit setting (on/off) indicates a particular condition. ICSF sets bit 0 and an exit cannot change that bit. Your exit can set any of the other bits. **Bit** — **Meaning When Set On/Off** **0** — Postprocessing invocation./Preprocessing invocation. **1** — **Reserved.** **2** — Use the return and reason code that the exit places in register 0 and register 15 as the service's return code/reason code. Do not use the exit's return code as the service return code in registers 0 and 15. The exit can pass any valid return code in register 15 and any valid reason code in register 0. If this bit is set on, ICSF uses these codes as the service's return and reason codes. See “Return Codes” on page 133 for more information about using exit return codes. **3** — Do not call the postprocessing invocation of the service exit./Call the postprocessing invocation of the service exit. **4** — Bypass the service./Run the service. **5** — Use the return and reason code that the exit places in the service's parameter list./Do not store codes the exit places in the service's parameter list. The exit can pass any valid return and reason code in the first two parameters of the service's parameter list. “Parameters” on page 133 describes the service parameter list. **6** — CSFSKRC bypass input label parsing./CSFSKRC parse the input label. **7–31** — **Reserved.**
24	4	Secondary parameter block. The address of the secondary parameter block. The exit can use the SPB to determine the environmental information of the service. For a description of the SPB, see “Secondary parameter block” on page 131.

Table 16. EXPB Control Block Format for Services (continued)

Offset (Dec)	Number of Bytes	Description
28	4	CCVT. Address of the Cryptographic Control Vector Table (CCVT). For a description of the CCVT, see “The Cryptographic Communication Vector Table (CCVT)” on page 266.
32	8	Module name. The installation exit's load module name. The field contains the value of the load module name you specified on the EXIT keyword in the installation options data set. The field is 8 bytes of characters, and the value is left-justified and padded with blanks.

Secondary parameter block

Offset +24 of EXPB contains the address of the secondary parameter block (SPB). The exit can use the SPB to determine the environmental conditions of the service. Table 17 describes the contents of SPB.

Table 17. SPB Control Block Format

Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. The field contains the character string `SPB`.
4	2	Version. The version of the control block. The field contains the character string `04`.
6	2	Length. The length of the control block.
8	4	CCVT. The address of the Cryptographic Communication Vector Table (CCVT). For a description of the CCVT, see“The Cryptographic Communication Vector Table (CCVT)” on page 266.
12	4	Signal Information Word. Bytes 1–2 Reserved. Bytes 3–4 of the field contain the installation-assigned code number for an installation-defined service.

Table 17. SPB Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
16	4	Flags and Indicators. Each byte of this field is either an indicator byte or contains flag bits. The contents of each byte in the field are: Byte 1—PSW key. This byte contains the original caller's program status word key. The first four bits are the key and the remaining four bits are zeros. Byte 2—Caller's state. Each bit in byte 2 indicates a condition of the caller's state. **Bit** — **Meaning When Set On** **0** — ICSF was entered via SVC entry from a PCF compatibility macro. **1** — Original caller in AMODE(31). **2** — Original caller in AR mode. **3** — Original caller in SRB mode. **4** — Original caller in supervisor state or system key. **5** — Original caller in AMODE(64). **6–7** — **Reserved**. Byte 3—Flag byte 1. The first flag byte. Each bit that is set on indicates a particular condition. **Note:** These bits are informational. Do not change bits 0 and 1. **Bit** — **Meaning When Set On** **0** — The service is using a “storage access” crypto instruction. **1** — Key record found in in-store KDS during delete operation. **2** — The recovery routine should not retry. **3 and 4** — **Reserved** **5** — Hardware initialization in process. **6** — NQAP in progress. **7** — **Reserved** Byte 4—Flag byte 2 **Bit** — **Meaning When Set On** **0** — The service parameter list has a position for a return code. **1** — The service parameter list has a position for a reason code. **2** — In-store CKDS record format is variable length. **3** — The caller has no exit data. **4** — Change master key processing holds global AP latch **5-7** — **Reserved**
20	4	Protected storage pointer.
24	4	Auxiliary SPB Pointer
28	4	EDC buffer pointer.
32	4	EDC buffer length.

Table 17. SPB Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
36	4	Address of XPB.
40	8	ID for latch manager.
48	4	Address for ERPB.
52	8	Original caller's register 1.
60	4	Address of CPRB request storage.
64	4	Length of CPRB request storage.
68	4	Address of CPRB reply storage.
72	4	Length of CPRB reply storage.
76	4	CCPS address.
80	4	Serialization block address.
84	4	Recovery token.
88	8	Recovery footprint for hash tables.
96	4	Pointer to ICSF/C resource owning object.
100	4	Pointer to metal C stack.
104	2	Entry point index of metal C caller.
106	2	Flags and indicators Byte 1 - Reserved for dump processing which will be overwritten when being copied. **Bit** **Meaning When Set On** **0** Dump CKDS in-store **1** Dump PKDS in-store **2** Dump TKDS in-store and session objects **3-7** **Reserved** Byte 2 - **Reserved**
108	4	ASCB of SPB owner.
112	4	Register 14 from CSFMIREC.
116	4	Address of MSTB.
120	24	**Reserved.**

Parameters

Each service has a unique parameter list. Parameters 1–4 are always the return code, reason code, exit data length, and exit data. The other parameters differ with each service. The installation exit gets passed the address of the service parameter list in Register 1. For a description of each service's parameter list, refer to *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Return Codes

To use a return code and reason code that are set in the postprocessing exit, you must set bit 2 in Offset +20 of EXPB. Setting bit 2 on causes ICSF to return the return code from the exit in register 15 and the reason code in register 0. Even though the application program receives the codes from the exit in the registers, the program still receives the codes from the service in the parameter list. The return code is the first parameter, and the reason code is the second parameter in the list.

Some control languages can access registers more easily than others. For this reason, ICSF allows you to return the return code and the reason code in both the registers and the parameter list. To do this, set bit 5 as well as bit 2 in Offset +20 of EXPB. The application then receives the return code and the reason code from the exit in both the registers and the parameter list.

If you do not set either of or both of the flag bits, the service ignores any return or reason code from the exit. The application program receives the codes from the service in both the registers and the parameter list.

The exit can pass back any valid return code for each service. For a listing of each service's return codes, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Cryptographic key data set entry retrieval installation exit

The cryptographic key data set entry retrieval installation exit (CSFCKDS) is called when a service requests an entry from the in-storage cryptographic key data set (CKDS) by label. ICSF calls this exit after it finds the record in the CKDS and before it returns the record to the service.

Note: This exit is given control only for a fixed-length record CKDS. The exit does not work with the variable-length record format of the CKDS.

Purpose and use of the exit

The exit point lists the entry that matches a certain label and type. You can use the exit to check fields in a record and decide whether to use the record. The exit sets a return code that specifies whether to use the record or not. Use the *exit_data* parameter in the service to specify what the exit should use as a search value.

For example, you can use the CKDS entry retrieval exit to perform a specific search of the installation data field. An installation can specify whatever it chooses to in the installation data field. The exit can select a record that matches a certain key label and key type. You can check the record and accept or reject it based on the installation data field.

Note: The cryptographic key data set entry retrieval installation exit will not be given control if SYSPLEXCKDS(YES,FAIL(xxx)) is specified in the ICSF installation options data set.

Environment of the exit

The exit receives control with these characteristics:

- Supervisor state
- Key 0
- APF-authorized
- TCB or SRB mode
- AR mode
- AMODE(31)
- RMODE(ANY)
- Cross memory mode

The exit can change the characteristics during its processing. However, the exit must return to its caller with the same characteristics as on entry.

The exit runs in the cross memory mode in the ICSF address space. The CKDS records are ALET-qualified. ICSF supplies the address and the ALET of a CKDS record as parameters to the CKDS retrieval exit.

For information about cross memory mode and AR mode, see *z/OS MVS Programming: Extended Addressability Guide*.

Installing the exit

Install the CKDS entry retrieval exit by installing the load module that contains the exit into an APF-authorized library. ICSF uses this normal z/OS search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and the load module name on the EXIT keyword in the installation options data set. “Parameters in the installation options data set” on page 38 describes the installation options data set in further detail. The EXIT keyword has this syntax:

EXIT (*ICSF name*, *load module name*, **FAIL (***options***))**

The **ICSF name** portion of the keyword refers to the ICSF name for the exit. The ICSF name for the CKDS entry retrieval exit is CSFCKDS. The **load module name** is the name of the load module that contains the exit. The name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded or if it ends abnormally. The valid FAIL options are:

NONE Do not take any action.

EXIT Do not call this exit again. The exit will not receive control during subsequent attempts at CKDS retrieval.

SERVICE Fail the service. All subsequent attempts at CKDS entry retrieval fail.

ICSF End ICSF.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit. If the exit ends abnormally, the attempt at CKDS entry retrieval fails, regardless of the FAIL option you specified. FAIL options only apply to subsequent attempts at CKDS entry retrieval.

Input

The CKDS entry retrieval exit receives the address of an address list passed in register 1. Each address in the list points to a parameter. The address list exists in the ICSF address space, and register 1 is not ALET-qualified.

Table 18 describes the parameters for the CKDS entry retrieval exit.

Table 18. The CKDS Entry Retrieval Exit Parameters

Parameter	Description
1	The address of the current CKDS record. See Table 26 on page 194 for a description of the CKDS record format.

Table 18. The CKDS Entry Retrieval Exit Parameters (continued)

Parameter	Description
2	The address of the ALET of the current CKDS record. This record is a fullword address.
3	The address of the record that matches a certain label and type. This value is a fullword integer. The parameter is in the ICSF address space and the exit can access the parameter using an ALET of 0.
4	The address of the record chosen. This value is a fullword integer. The parameter is in the ICSF address space and the exit can access the parameter using an ALET of 0.
5	The address of the exit data length. This value is a fullword integer. The parameter is in the caller's address space, which is the secondary address space, and the exit can access the parameter using an ALET of 1.
6	The address of the exit data. For a description of exit data, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*. The parameter is in the caller's address space, which is the secondary address space, and the exit can access the parameter using an ALET of 1.
7	The address of the secondary parameter block. See “Secondary parameter block” on page 131 for a description of the secondary parameter block. The parameter is in the ICSF address space and the exit can access the parameter using an ALET of 0.

Return codes

You can pass a return code back in register 15.

The valid decimal return codes are:

Return Code	Description
0	Use the record.
4	Do not use the record.

If you specify not to use any of the records that match the search value, ICSF returns control to the application. It returns with return code 12 and reason code 10024, which indicate that the exit rejected all the keys in the search.

PCF conversion program installation exit

Use the PCF conversion program to convert a CKDS from the Programmed Cryptographic Facility (PCF) format to the ICSF format. The conversion program converts each record in the PCF CKDS to the CKDS format that ICSF uses, and then writes the new record to an ICSF CKDS. The conversion program extends the label field to 64 bytes.

An ICSF CKDS record contains an installation data field that you can use to further identify the record. This field can contain any information about a record that your installation would like to use. You can use the conversion program exit to change the information in this field. You can also use the conversion program exit to have the conversion program not place a converted CKDS entry in the ICSF CKDS.

Chapter 8, “Migration from PCF to z/OS ICSF,” on page 173 contains more information about the PCF conversion program.

Note: The ICSF/MVS Version 1 Release 1 conversion program cannot run this exit.

Purpose and use of the exit

The PCF conversion program installation exit (CSFCONVX) is called at three points during processing of the conversion program:

- **During conversion program initialization.** This is known as the conversion preprocessing invocation. At this point, you can use the exit to change the ICSF CKDS header record installation data field.
- **During conversion program individual record processing.** This is known as the record processing invocation. At this point, the conversion program is converting the PCF entry but has not yet placed the entry into the ICSF CKDS. You can use the exit to change the installation data field in the entry for the ICSF CKDS. You can also specify that the conversion program not place the entry into the ICSF CKDS.
- **Just prior to conversion program termination.** This is known as the conversion postprocessing invocation. At this point, like the preprocessing exit point, you can use the exit to change the ICSF CKDS header record installation data field.

Environment of the exit

The exit receives control with these characteristics:

- Problem program state.
- APF-authorized
- TCB mode
- Address Space Control mode=primary
- AMODE(31)
- RMODE(ANY)

The exit can change the characteristics during its processing. However, the exit must return to its caller with the same characteristics as on entry.

The exit runs in task mode in the caller's own address space.

Installing the exit

Install the load module that contains the exit into an APF-authorized library. ICSF uses this normal OS/390 search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Joblib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and load module name on the EXIT keyword in the installation options data set. For more information about the installation options data set, see “Parameters in the installation options data set” on page 38. The EXIT keyword has this syntax:

EXIT (*ICSF name*, *load module name*, **FAIL (***options***))**

The **ICSF name** portion of the keyword refers to the ICSF name for the exit. The ICSF name for the conversion program exit is CSFCONVX. The **load module name** is the name of the load module that contains the exit. This name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword

specifies the action ICSF takes if the exit cannot be loaded. The valid FAIL options are **NONE**, **EXIT**, **SERVICE**, and **CSF**. For the conversion program exit, you can use these options only:

NONE Initialization continues even if exit cannot be loaded.
ICSF Initialization ends if exit cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit.

If the exit ends abnormally, the conversion program does also.

Input

ICSF supplies the address of the conversion program exit parameter block (CVXP) in register 2 each time it calls the PCF conversion program exit. The exit does not receive a parameter list. “Entry and return specifications” on page 113 gives a complete list of the registers on entry to the conversion program exit.

Table 19 describes the contents of the exit control block.

Table 19. CVXP Control Block Format

Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. The field contains the character string `CVXP`.
4	2	Version. The version of the control block. The field contains the character string `01`.
6	2	Length. The length of the control block. The value is 28 in decimal.
8	4	Return Code. The value the exit returns. Valid decimal values for this field are: **Return Code** **Description** **0** Normal. **4** Do not process the entry. **8** End conversion program.
12	4	Address of the ICSF CKDS installation data area.
16	4	The value in decimal of the length of the ICSF CKDS installation data area.
20	1	Action. Bit 0 is set on if the action was to change an entry on the ICSF CKDS. Bit 0 is set off if the action was to add an entry to the ICSF CKDS. The rest of the bits in this byte are reserved.

Table 19. CVXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
21	1	Call Point. Indicates the invocation point of the exit. The exit cannot change this field and the conversion program does not use this field on return from the exit. You can determine the invocation point by the bit that is set on. **Bit** — **Meaning When Set On** **0** — Conversion preprocessing invocation. **1** — Conversion postprocessing invocation. **2** — Record processing invocation. **3-7** — **Reserved**.
22	6	Reserved.

Return codes

You can pass a return code back to the conversion program in the CVXP control block (offset +8). The exit can use return codes to reject records for conversion processing or end the conversion program.

Return Code	Description
0	Normal.
4	Do not process the entry.
8	End conversion program.

Single-record, Read-write installation exit

ICSF provides an exit that is called when a record is read from or written to a CKDS or PKDS. ICSF calls the single-record, read-write (CSFSRRW) exit under these conditions:

- The PCF conversion program converts a record into ICSF CKDS format. The conversion program calls the exit right before it writes a converted record to the ICSF CKDS.
- ICSF reenciphers a disk copy of a CKDS under a new master key. ICSF calls the exit two times during this processing; after ICSF reads a record to reencipher it and before ICSF writes the reenciphered record.
- ICSF refreshes the in-storage copy of a CKDS. ICSF calls this exit after reading a record from the disk copy to place into storage.
- You enter a key into a disk copy of the CKDS by using the key entry hardware. ICSF calls the exit two times during this processing: after reading a partial key from the CKDS, and before writing a record into a CKDS.
- An application creates, reads, writes, or deletes a record from the PKDS.

Using the exit, you can do such things as prevent the record from being processed, or add user information to the record.

Note: This exit is given control only for a fixed-length record CKDS. The exit does not work with the variable-length record format of the CKDS.

Purpose and use of the exit

The exit receives a parameter block that describes the CKDS or PKDS record and the action occurring to the record. By setting a return code in the parameter block, the exit may affect the processing of the record. Depending on the return code, one of these actions occurs:

- ICSF continues to read the record.
- ICSF does not read or write the record.
- ICSF does not read or write the entire CKDS or PKDS.

The parameter block contains the address of the CKDS or PKDS record. The exit can add information into the installation data field of the record. For integrity reasons, ICSF receives only changes to this particular field. If the exit sets a return code to continue processing, ICSF processes the record with this information.

The KGUP exit, the PCF conversion program exit, and the single-record, read-write exit can add information to the installation data field of the CKDS or PKDS header record to identify the data set. If the header record installation data field contains information identifying the CKDS or PKDS, the single-record, read-write exit can check the field to ensure that it is processing the correct data set. If the exit finds that it is processing the wrong CKDS or PKDS, the exit can set a return code to stop the processing of the entire data set.

You can use the exit to prevent processing of a record. You can check certain fields in the record and specify that the record not be processed. For example, during postprocessing conversion, you can prevent the processing of any record of a certain key type. However, the exit should never prevent processing of a record containing a system key because ICSF uses these keys in its processing. You differentiate a system key record from other key records by its key label. A system key record label contains all binary zeros. All other key labels contain an alphabetic first character with the remaining characters as either alphabetic or numeric.

Environment of the exit

The exit receives control with these characteristics:

- Problem program state
- APF-authorized
- TCB mode
- Address Space Control mode=primary
- AMODE(31)
- RMODE(ANY)

The exit can change the characteristics during its processing. However, the exit must return to ICSF with the same characteristics as on entry.

When the single-record, read-write exit is called, the exit parameter block is in the caller's address space. The exit is loaded in the caller's address space. The caller is either the PCF conversion program, the utility program (CSFEUTIL), or an ICSF panel.

Installing the exit

Install the load module that contains the exit into an APF-authorized library. ICSF uses this search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Joblib (if one exists)

- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and load module name on the EXIT keyword of the installation options data set. For more information about the installation options data set, see “Parameters in the installation options data set” on page 38. The EXIT keyword has this syntax:

EXIT (*ICSF name*, *load module name*, **FAIL (***options*))

The **ICSF name** portion of the keyword refers to the ICSF name for the exit. The ICSF name for the single-record, read-write exit is CSFSRRW. The **load module name** is the name of the load module that contains the exit. The name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded or ends abnormally. The valid FAIL options are:

NONE Do not take any action.
EXIT Do not call this exit again.
SERVICE Fail the service that called the exit.
ICSF Fail the service that called the exit.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit. If you specify FAIL(ICSF) and the exit cannot be loaded, ICSF initialization does not continue. If you specify FAIL(ICSF) and the exit ends abnormally, ICSF issues an advisory message that ICSF should be ended.

Input

The single-record, read-write exit receives the address of the address list passed in register 1. The first address in the address list is for the read-write exit parameter block (RWXP). The exit does not receive a parameter list. “Entry and return specifications” on page 113 gives a complete list of the registers on entry to the single-record, read-write exit.

The RWXP parameter block contains the address of the CKDS or PKDS record that is being processed and information about the situation in which the exit is called. The exit sets a return code in a field in the block to specify whether the processing should continue. Table 20 describes the RWXP control block.

Table 20. RWXP Control Block Format

Offset (Dec)	Number of Bytes	Description
0	4	Name. The name of the control block. The field contains the character string `RWXP`.
4	2	Version. The version of the control block. The field contains the character string `02`.
6	2	Length. The length of the control block. The value of this field is 28 in decimal.

Table 20. RWXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
8	4	Return Code. The value the exit returns. Valid decimal values for this field are: **Return Code** — **Description** **0** — Process current CKDS record **4** — Do not process current CKDS record **8** — End processing
12	4	Address of the CKDS record.
16	4	The value in decimal of the length of the CKDS record.
20	7	Action. The field is a 7-byte character string describing the action performed on the CKDS record. The field can contain these values: • `READ` • `WRITE` • `DELETE` Note that the value of the field is left-justified and padded with blanks.
27	1	Exit Invocation Reason The reason that the exit was invoked. The field relates to only the CKDS and can contain one of these values: **2** — Refresh of the in-storage CKDS with a disk copy of a CKDS. The value of the Action field is READ. **3** — Reencipher of the in-storage CKDS from a disk copy of a CKDS. The value of the Action field is READ or WRITE. **5** — Conversion record postprocessing. The value of the Action field is WRITE. **8** — Key entry hardware input. The value of the Action field is READ or WRITE.
28	4	Data set type (either CKDS or PKDS).

Return codes

You can pass a return code back to the single-record, read-write process in the RWXP control block (offset +8) or in register 15. The exit can use the return code to reject records or to end the single record read-write process. These values are valid decimal return codes:

Return Code	Description
0	Process the current CKDS record.
4	Do not process the current CKDS record.
8	End processing.

Exit points for security installation exits

IBM-supplied security exit routines were removed in ICSF/MVS Version 2 Release 1. The exit points themselves are still available.

Security installation exits

ICSF provides these exit points to control access to the keys in the in-storage CKDS and to the services.

- Security Initialization Exit
- Security Termination Exit
- Security Service Exit
- Security Key Exit

Purpose and use of the exits

There are two groups of security exits. The security initialization exit (CSFESECI) and security termination exit (CSFESECT) are called during ICSF mainline processing to maintain a security communication area that is used by the other security exits.

Next is a description of each point where ICSF calls security exit routines.

Security initialization exit

ICSF calls this exit during initialization just before calling the ICSF mainline exit CSFEXIT. You can use this exit to anchor resource lists, work areas, and other data to the security communication area. The security service exit (CSFESECS) and security key exit (CSFESECK) can be used to control access to resources on ICSF and for logging in SMF the results of any authorization checks that are made. The security initialization exit defined in the options data set is only invoked if CSFESECS, CSFESECK, or both are also defined.

Security termination exit

ICSF calls this exit as the last function when ICSF ends, before deleting all the installation exits. You can use this exit to free whatever is anchored to the security communication area.

Security service exit

ICSF calls this exit when an application uses an IBM-supplied service, before calling any other installation exit that is associated with that service. You can use this exit to control access to a service. Refer to Table 14 on page 125 for a list of services.

Security key exit

ICSF calls this exit when an application uses a key in the in-storage CKDS, before any other installation exit associated with that use of the key is called. You can use this exit to control access to the keys in the CKDS.

Environment of the exits

The security initialization and termination exits receive control with these characteristics:

- Supervisor state
- Key 0
- APF-authorized
- TCB mode
- Address Space Control mode=access register mode

- AMODE(31)
- RMODE(ANY)

The exits can change the characteristics during their processing. However, the exits must return to ICSF with the same characteristics as on entry.

The security service and key exits receive control with these characteristics:
- Supervisor state
- Key 0
- APF-authorized
- TCB mode
- Cross memory mode
- AR mode
- AMODE(31)
- RMODE(ANY)

The exits can change the characteristics during their processing. However, the exits must return to ICSF with the same characteristics as on entry.

Note: The security exits are not called in SRB mode.

Installing the exits

You install the security exits by installing the load module that contains the exit into an APF authorized library. ICSF uses this normal search order to locate the exit:
- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Use the EXIT keyword in the installation options data set to define the ICSF name and load module name. For information about the installation options data set, see Parameters in the installation options data set. The EXIT keyword has this syntax:

EXIT (*ICSF name*, *load module name*, **FAIL (***options*))

The **ICSF name** portion of the keyword refers to the ICSF identifier for each exit, CSFESECI, CSFESECT, CSFESECS, and CSFESECK. The **load module name** is the name of the load module that contains the exit. The name can be any valid name your installation chooses. The action that the **FAIL** portion of the EXIT keyword specifies depends on the type of security exit.

For the security initialization and termination exits, the FAIL portion specifies the action ICSF takes if the exit cannot be loaded. The valid FAIL options mean:

NONE	Continue initialization even if exits cannot be loaded.
SERVICE	Continue initialization even if exits cannot be loaded.
EXIT	Continue initialization even if exits cannot be loaded.
ICSF	End ICSF if exits cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit.

If the security initialization exit ends abnormally, ICSF ends. If the security termination exit ends abnormally, ICSF continues to end.

For the security service and key exits, the FAIL portion specifies the action ICSF takes if the exit cannot be loaded or ends abnormally. When the service or key exit is loaded, the valid FAIL options mean:

NONE Continue initialization even if exits cannot be loaded.
SERVICE Continue initialization even if exits cannot be loaded.
EXIT Continue initialization even if exits cannot be loaded.
ICSF End ICSF if exits cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit.

When the security service exit ends abnormally, the valid FAIL options mean:

NONE Process subsequent calls to the service as if no abnormal ending occurred. Call the exit for each call of a service.

SERVICE Fail on subsequent calls to the particular service.

EXIT Do not call the exit again. Bypass the exit on subsequent calls to any IBM service.

ICSF End ICSF.

If the security service exit ends abnormally, ICSF ends the service call before performing the service.

When the security key exit ends abnormally, the valid FAIL options mean:

NONE Process subsequent attempts to access the in-storage CKDS as if no abnormal ending occurred. Call the exit for each access attempt.

SERVICE Fail on subsequent attempts to access the CKDS.

EXIT Do not call the exit again. Bypass the exit on subsequent accesses of the CKDS.

ICSF End ICSF.

If the security key exit ends abnormally, ICSF ends the attempt to access the CKDS before performing the access.

Input

The security initialization and termination exits receive the address of an 8-byte security communication area in register 1. When ICSF starts, the security initialization exit can use this area as an anchor for resource lists, work areas, or any other data that your service or keys security exits need to check authorizations. When ICSF ends, the security termination exit can free any system resources that are anchored to this area and used by the service or keys security exits. For example, the exit can free storage that is allocated from the common storage area (CSA).

When a call to a service occurs, the security service exit receives the address of an address list passed in register 1. Table 21 describes the parameters the exit receives:

Table 21. Parameters Received by the Security Service Exit

Parameter	Number of Bytes	Description
1	8	The security communication area.

Table 21. Parameters Received by the Security Service Exit (continued)

Parameter	Number of Bytes	Description
2	8	The character string CSFSERV.
3	8	The name of the service being called.

When an attempt to access a CKDS entry occurs, the security key exit receives the address of an address list passed in register 1. Table 22 describes the parameters this exit receives:

Table 22. Parameters Received by the Security Key Exit

Parameter	Number of Bytes	Description
1	8	The security communication area.
2	8	The character string CSFKEYS.
3	64	The label of the key entry being accessed.

Register 0 contains the address of the exit parameter block (EXPB). See Figure 8 on page 129 and Table 16 on page 129.

Return codes

All the security exits can pass back a return code in register 15. The security initialization exit supports these decimal return codes:

Return Code	Description
0	Proceed with initialization.
4	End ICSF.

Any return codes other than those listed cause ICSF to end abnormally.

The security termination exit supports these decimal return codes:

Return Code	Description
0 or 4	Proceed with termination.

Any return codes other than those listed cause ICSF to end abnormally.

The security service exit supports these decimal return codes:

Return Code	Description
0 or 4	Proceed with the service call.

Any return codes other than those that are listed cause the service call to fail.

The security key exit supports these decimal return codes:

Return Code	Description
0 or 4	Proceed with the access of the CKDS entry.

Any return codes other than those that are listed cause the access of the key to fail.

Key generator utility program installation exit

The key generator utility program (KGUP) generates and maintains keys in the cryptographic key data set (CKDS). You can use KGUP to generate or supply a key to update the CKDS. KGUP generates keys to use in key exchange with other systems. ICSF provides an exit for customizing KGUP processing. For information about using KGUP to managing cryptographic keys, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

Purpose and use of the exit

You can use the KGUP installation exit (CSFKGUP) to modify records in the CKDS, write copies of records to alternate data sets, or put additional information in the SMF record. There are many other uses for the KGUP exit depending on your installation's needs. Examine the calling points for an exit and the active control block fields at each calling point to determine other applications for the exit.

KGUP calling points

After an ICSF administrator submits a KGUP job for processing, KGUP calls exits at four points in processing:

1. **During KGUP initialization.** This is known as the KGUP preprocessing exit. After the KGUP job begins but before KGUP starts processing a control statement, KGUP calls this exit.

   You can use this exit to place additional information in the installation data field of the CKDS header record. You may want to do this if you need to process different cryptographic key data sets differently. You can place information in the installation data field of the record, and then subsequent calls of the exit can use this information as the basis for performing processes.

2. **Before KGUP processes a key that is identified by a control statement.** This is known as the record preprocessing exit. Before KGUP accesses the CKDS to retrieve the key that is requested in the control statement, KGUP calls the exit again.

   Note: This call occurs before KGUP accesses the CKDS. If an exit routine alters a key entry at this call, KGUP accesses the CKDS with the altered entry.

   You can use this exit to provide additional security for entering clear key values. When a user enters a clear key in a control statement, use the exit to change the value. In this way, the user never knows the actual clear value in the CKDS. For example, a user enters zeros for clear key values. Your exit generates some random number and replaces the user's clear key value. KGUP then processes the exit's random number as the value to write to the CKDS.

3. **Before KGUP updates the CKDS with a key entry.** This is known as the record postprocessing exit. After KGUP processes a key and before KGUP updates the CKDS, KGUP calls the exit a third time.

   At this call, the installation exit can change any information in the Key Output Data Set. Changing the Key Output Data Set also enters the changed keys into the Control Statement Output Data Set, if the keys are exportable. You can use this exit to create audit trails.

4. **During KGUP termination.** This is known as the KGUP postprocessing exit. Calls to this exit occur after KGUP completes processing but before KGUP returns control to ICSF.

Note: If an error occurs in exit processing, KGUP does not call the remaining exit invocations. If an error occurs in KGUP processing that does not result in an abnormal ending, KGUP does not call the remaining exit invocations.

Processing in the exit

At each call, the exit receives the address of the KGUP exit parameter block (KGXP) in register 1. The exit can access any of the data in KGXP. The exit can alter some of the fields in KGXP, while others are simply references. Also, the KGUP exit can alter some fields at some calls but not at other calls.

A field in KGXP gives the calling point of the exit. The exit uses this field to determine when to call the exit to perform appropriate processing. “Input” on page 149 gives a more detailed explanation of the KGXP control block, the values it contains, and when an exit can use or change the values.

Environment of the exit

The KGUP calls the exit only in the address space where KGUP is running. The exit receives control with these characteristics:

- Supervisor state
- APF-authorized
- TCB mode
- Address Space Control mode=primary
- AMODE(31)
- RMODE(ANY)

The exit can change the characteristics during its processing. However, the exit must return to its caller with the same characteristics as on entry.

Installing the exit

Install the load module that contains the exit into an APF authorized library. ICSF uses this search order to locate the exit:

- Job pack area
- Steplib (if one exists)
- Joblib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Define the ICSF name and load module name on the EXIT keyword in the installation options data set. For more information about the installation options data set, see “Parameters in the installation options data set” on page 38. The EXIT keyword has this syntax:

EXIT (*ICSF name*, *load module name*, **FAIL (***options***))**

The **ICSF name** portion of the keyword refers to the ICSF name for the KGUP exit. The ICSF name for the KGUP exit is CSFKGUP. The **load module name** is the name of the load module that contains the exit. The name can be any valid name that your installation chooses. The **FAIL** portion of the EXIT keyword specifies the action ICSF takes if the exit cannot be loaded. The valid FAIL options are **NONE, EXIT, SERVICE,** and **CSF.** The FAIL options available to the KGUP exit are:

NONE Initialization continues even if exit cannot be loaded.
ICSF Initialization ends if exit cannot be loaded.

You must specify a FAIL option. If you do not, ICSF returns an error message, ends abnormally, and generates an SVC dump when attempting to load the exit. If the exit ends abnormally, KGUP also ends abnormally.

Input

At each of the invocation points, the exit receives the address of the KGUP exit parameter block (KGXP) in register 1. The exit does not receive a parameter list. “Entry and return specifications” on page 113 gives a complete list of the registers on entry to the KGUP exit.

The KGUP exit can alter some of the fields in KGXP. Some fields only provide information to the exit and cannot be changed, and some fields do not apply to particular calls to the exit.

Table 23 describes the KGXP control block.

Table 23. KGXP Control Block Format

Offset (Dec)	Number of Bytes	Description
0	4	Block Identifier. The name of the control block. The field must contain the character string `KGXP`. The exit must not change the value and KGUP does not use the field upon return from the exit.
4	2	Block Version Number. The version of the control block. The field must contain the character string `03`. The exit cannot change this field and KGUP does not use this field on return from the exit.
6	2	Block Length. The length of the control block. The decimal value of the field is 408. The exit cannot change the field and KGUP does not use this field on return from the exit.
8	4	Return Code. The return code the exit supplies upon completion. Upon entry, KGUP initializes this field to zeros. The valid decimal return codes for each of the invocation points are: Record Pre- or postprocessing. **0** Normal, continue processing. **4** Reject control statement, but do not end KGUP. **8** End KGUP immediately. KGUP pre- or postprocessing. **0** Normal, continue processing. **> 0** End KGUP immediately.

Table 23. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
12	1	Call Point. Indicates the invocation point of the exit. The exit cannot change this field and KGUP does not use this field on return from the exit. You can determine the invocation point by the bit that is set on. **Bit** — **Meaning When Set On** **0** — KGUP preprocessing invocation. **1** — KGUP postprocessing invocation. **2** — Record preprocessing invocation. **3** — Record postprocessing invocation. **4-7** — **Reserved**.
13	1	Options. Indicates the keywords specified on the KGUP control statement. The exit cannot change this field and KGUP does not use the field upon return from the exit. The field is used only during the record preprocessing and postprocessing invocations. You can determine the keywords on the control statement by the bits that are set on. **Bit** — **Meaning When Set On** **0** — LABEL with multiple values specified. **1** — RANGE specified. **2** — KEY specified. **3** — CLEAR specified. **4** — SINGLE specified. **5** — NOCV specified. **6** — OUTTYPE specified. **7** — **Reserved.**

Table 23. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
14	1	Verb Type. Indicates the verb used on the KGUP control statement. The exit cannot change this field and KGUP does not use this field on return from the exit. The field is used only for the record preprocessing and record postprocessing invocations. You can determine the verb on the control statement by the bit that is set on. **Bit** — **Meaning When Set On** **0** — ADD **1** — UPDATE **2** — DELETE **3** — RENAME **4** — SET **5** — OPKYLOAD **6–7** — **Reserved.**
15	1	KGUP Flags. Indicates the processing conditions encountered by KGUP at the record postprocessing invocation. The exit cannot change this field and KGUP does not use the field upon return from the exit. The field is not used for the KGUP pre- or postprocessing invocations or the record preprocessing invocation. The processing conditions can be determined by examining whether bit 0 is set on. **Bit** — **Meaning When Set On** **0** — Non-odd parity key was imported. **1** — Algorithm is AES **2** — Algorithm is DES **3–7** — **Reserved.**

Table 23. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
16	72	Action Key. Contains the key index accessed by the KGUP control statement. The key index consists of the key label and type fields of a CKDS record entry (“Debugging Aids” on page 101 describes the CKDS record format in greater detail). The key index is the first 72 bytes of a CKDS record, and the information in the key index is used to differentiate one key from another. The exit can modify the field at the record preprocessing invocation. The field is not used for the KGUP pre- or postprocessing invocation or the record postprocessing invocation. If the exit modifies the field, KGUP uses the modified field to access the CKDS upon return from the exit. Before the record preprocessing invocation, KGUP places this in this field: • The key label or key old label from the LABEL or key label from the RANGE keyword of the control statement • The key type from the TYPE keyword of the control statement The exit cannot modify the key label, key old label, or key type.
88	72	Rename Key. Contains the key index used to rename a key when RENAME is the verb on the control statement. The key index consists of the key label and type fields of a CKDS record entry. The exit can modify the field at the record preprocessing invocation. The field is not used for the KGUP pre- or postprocessing or record postprocessing invocations. If the exit modifies the field, KGUP uses the modified field to access the CKDS upon return from the exit. Before the record preprocessing invocation, KGUP places this information in this field: • The key new label from the LABEL keyword of the control statement. • The key type from the TYPE keyword of the control statement. The exit cannot modify the key new label or the key type.

Table 23. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
160	72	Transkey key-label1. The key index of the TRANSKEY key-label1 on the KGUP control statement. The key index is the key label and type of the CKDS record entry. The exit can modify the field at the record preprocessing invocation. The field is not used for the KGUP pre- or postprocessing and record postprocessing invocations. If the exit modifies the field, KGUP uses the modified field to access the CKDS upon return from the exit. Before the record preprocessing invocation, KGUP places this information in this field: • The key-label1 from the TRANSKEY keyword of the control statement. • The key type. The type is IMPORTER, if keys are supplied; the type is EXPORTER, if keys are not supplied. The exit cannot modify the key-label1 or the key type.
232	72	Transkey key-label2. The key index of the TRANSKEY key-label2 on the KGUP control statement. The key index is the key label and type of the CKDS record entry. The exit can modify the field at the record preprocessing invocation. The field is not used for the KGUP pre- or postprocessing and record postprocessing invocations. If the exit modifies the field, KGUP uses the modified field to access the CKDS upon return from the exit. Before the record preprocessing invocation, KGUP places this information in this field: • The key-label2 from the TRANSKEY keyword of the control statement. • The key type. The key type is IMPORTER, if keys are supplied; the type is EXPORTER, if keys are not supplied. The exit cannot modify the key-label2 or the key type.
304	8	The OUTTYPE value, if specified. If no OUTTYPE is specified, this field set to binary zeros.
312	4	Key length in bytes. The value supplied by the LENGTH keyword or the byte length of the key value if the KEY option was selected. This value is for ease of processing the key values. The exit may not modify this value.

Table 23. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
316	16	Key key-value 1. The value of the key supplied on the KGUP control statement. The 16 bytes are hexadecimal characters representing the 8-byte hexadecimal key value. The field contains a value only if the KEY option was specified and a key value was supplied on the control statement. You can determine whether the KEY option was used by examining bit 2 at offset +13 in KGXP. If TRANSKEY was specified on the control statement, KGUP decrypts key-value1 under the transport key specified with the TRANSKEY keyword. If CLEAR was specified on the control statement, KGUP does not decrypt key-value1. The exit can modify the field at the record preprocessing invocation. This field is not used for the KGUP pre- or postprocessing invocations or the record postprocessing invocation. The field does not contain a value when generating keys. The exit is permitted to put values in this field only if a key was supplied on the control statement. The exit-supplied value must be edited for hexadecimal values and it then replaces the values entered on the input control statement.
332	16	Key key-value 2. The value of the second key supplied on the KGUP control statement. The 16 bytes are hexadecimal characters representing the 8-byte hexadecimal key value. The field contains a value only if the KEY option was specified and a key value was supplied on the control statement. You can determine whether the KEY option was used by examining bit 2 at offset +13 in KGXP. If TRANSKEY was specified on the control statement, KGUP decrypts the key-value 2 under the transport key specified with the TRANSKEY keyword. If SINGLE was specified on the control statement, the key-value 2 will be equal to the key-value. If CLEAR was specified on the control statement, KGUP does not decrypt the key-value 2. The exit can modify the field at the record preprocessing invocation. This field is not used at the KGUP pre- or postprocessing invocation or the record postprocessing invocation. The field does not contain a value when generating keys. The exit can put values in this field only if a key was supplied on the control statement. The exit-supplied value must be edited for hexadecimal values; it then replaces the values entered on the input control statement.

Table 23. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
348	16	Key key-value 3. The value of the third key supplied on the KGUP control statement. The 16 bytes are hexadecimal characters representing the 8-byte hexadecimal key value. The field contains a value only if the KEY option was specified and a key value was supplied on the control statement. You can determine whether the KEY option was used by examining bit 2 at offset +13 in KGXP. If TRANSKEY was specified on the control statement, KGUP decrypts the key-value 3 under the transport key specified with the TRANSKEY keyword. If CLEAR was specified on the control statement, KGUP does not decrypt the key-value 3. The exit can modify the field at the record preprocessing invocation. This field is not used at the KGUP pre- or postprocessing invocation or the record postprocessing invocation. The field does not contain a value when generating keys. The exit can put values in this field only if a key was supplied on the control statement. The exit-supplied value must be edited for hexadecimal values; it then replaces the values entered on the input control statement.
364	16	Key key-value 4. The value of the fourth key supplied on the KGUP control statement. The 16 bytes are hexadecimal characters representing the 8-byte hexadecimal key value. The field contains a value only if the KEY option was specified and a key value was supplied on the control statement. You can determine whether the KEY option was used by examining bit 2 at offset +13 in KGXP. The exit can modify the field at the record preprocessing invocation. This field is not used at the KGUP pre- or post-processing invocation or the record post-processing invocation. The field does not contain a value when generating keys. The exit can put values in this field only if a key was supplied on the control statement. The exit-supplied value must be edited for hexadecimal values; it then replaces the values entered on the input control statement.

Table 23. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
380	4	CSFKEYS record for transkey, key-label1. The address of the CSFKEYS data set record that is output for transkey key-label1 on the KGUP control statement. The field ONLY contains a value when generating keys. This field is filled in when CLEAR keys are generated. The exit can modify the field at the record postprocessing invocation. KGUP sets the address to zero for the KGUP pre- or postprocessing and record preprocessing invocations. KGUP does not check the field upon return from the exit. Normal CSFKEYS processing applies. KGUP uses key values on control statement creation. For the format of the CSFKEYS record, refer to *z/OS Cryptographic Services ICSF Administrator's Guide*.
384	4	CSFCKDS header record. The address of the CSFCKDS data set header record. The exit can check the field at the KGUP pre- or postprocessing invocations. However, the exit can modify the field only at the KGUP postprocessing invocation. KGUP sets the value of the field to zero for the record pre- or postprocessing invocations. The exit can modify the installation data field of the CKDS header record (see “Debugging Aids” on page 101 for a description of the CKDS header record. Offset +196 of the CKDS header record is the installation data field). The installation data field supplied by the exit is placed in the CKDS header record after the KGUP postprocessing invocation returns control to KGUP.
388	4	CSFCKDS record. The address of the CSFCKDS data set record processed by the KGUP control statement. KGUP sets the address to zero if the TRANSKEY keyword has two labels of transport keys. The exit can check the field only at the record postprocessing invocation. KGUP sets the address to zero for the record preprocessing and KGUP pre- or postprocessing invocations. The exit can modify the record area if the TRANSKEY keyword does not have two labels.

Table 23. KGXP Control Block Format (continued)

Offset (Dec)	Number of Bytes	Description
392	4	RENAME CSFCKDS record. The address of the CSFCKDS data set record processed when the RENAME verb is used in a control statement. You can determine whether the RENAME verb was used by examining bit 3 at offset +14 in KGXP. The exit can modify the field at the record postprocessing invocation. KGUP sets the address to zero for the record preprocessing and KGUP pre- or postprocessing invocations. The exit can modify the record area. KGUP does not check this field upon return from the invocation. Normal CSFCKDS processing applies.
396	4	Installation data. The address of the data specified on the INSTDATA keyword of the KGUP control statement. The address of the area is zero if a SET control statement has not been processed. “The SET statement” describes how to use the field in greater detail.
400	4	Installation exit area. The address of an area set by the installation that is preserved across all invocations of the exit. The first byte of the area contains the length of the area (including the length byte). After KGUP completes, the first 64 bytes of the area are written to the SMF data set. The exit has exclusive control of modifying this area. The area is only used as input to SMF processing upon completion of KGUP.

The SET statement

Use the SET control statements to specify data to send to a KGUP installation exit. For a more detailed description of the SET statement, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

The installation data field in KGXP (offset +396) contains the address of the data SET statement specifies. Data that is specified on a SET statement can be especially useful if you alter key entries. You may want to keep track of the entries you change by putting the original data and the changed data in the installation data area.

Return codes

You can pass a return code back to KGUP in the KGXP control block (offset +8). The exit can use the return code to cause KGUP to reject control statements or to end KGUP. Return code values, in decimal, for record pre- or postprocessing exit calls are:

Return Code	Description
0	Normal, continue processing.
4	Reject control statement, but do not end KGUP.
8	End KGUP.

All other return codes are not valid and cause KGUP to end.

Return code values, in decimal, for the KGUP pre- or postprocessing invocations are:

Return Code	Description
0	Normal, continue processing.
>0	End KGUP.

Chapter 6. Installation-Defined Callable Services

This topic contains Programming Interface information.

ICSF provides callable services that perform cryptographic functions. For example, the ICSF encipher callable service enciphers data. You call and pass parameters to a callable service from an application program. See *z/OS Cryptographic Services ICSF Application Programmer's Guide* for a description of the ICSF callable services.

Besides the callable services that ICSF provides, you can write your own callable services; these are known as *installation-defined callable services*.

Attention: Only an experienced system programmer should attempt to write an installation-defined callable service. The writing and installation of such a service require a thorough knowledge of system programming in an z/OS environment. If, without having this knowledge, you attempt to write or to install installation-defined callable services, you run the risk of seriously degrading the performance of your system and causing complete system failure.

To write an installation-defined callable service, you must first write the callable service and link-edit it into a load module. Then define the service in the installation options data set. Use the SERVICE installation option keyword to specify a number to identify the service and the load module that contains the service.

You must also write a service stub. To run an installation-defined callable service, you call a service stub from your application program. The service stub connects the application program with the installation-defined callable service. In the service stub, you specify the service number that identifies the callable service.

During ICSF startup, ICSF loads the load module that contains the service into the ICSF address space with the ICSF callable services. ICSF binds the service with the service number that you specified in the installation options data set.

This topic describes how to perform these tasks:
- Write a callable service.
- Define a callable service.
- Write a service stub.

Writing a callable service

An installation-defined callable service receives parameters from the application program when the program calls the service stub that is associated with the service. An installation-defined service can also access information in the secondary parameter block (SPB). The address of the SPB is passed in register 0. See “Secondary parameter block” on page 131 for a description of the SPB.

The service receives control with these characteristics.
- Supervisor state
- Key 0
- APF authorized
- TCB or SRB mode
- Cross memory mode
- AR mode
- AMODE(31) or AMODE(64)

- RMODE(ANY)

The service can change the characteristics during their processing. However, the service must return to its caller with the same characteristics as on entry.

You must write the services in assembler, because you are in Access Register and cross memory mode, and the addresses of some of the parameters you may access are ALET-qualified. In particular, parameters passed into a callable service are in the user's address space, which you can access with an ALET of 1. See *z/OS MVS Programming: Extended Addressability Guide* for information about cross memory and AR mode.

Contents of Registers

The contents of the registers on entry to the callable service are:

Register 0 Address of the secondary parameter block (SPB)
Register 1 Address of the parameter list
Register 2–13 Unpredictable
Register 14 Return address
Register 15 Service entry point address

The contents of the registers on exit from the callable service are:

Register 0 Reason code
Register 1–14 Same as on entry
Register 15 Return code

Figure 9 shows an example of entry and exit code for a generic service.

```
MYSERV    CSECT
MYSERV    AMODE  31
MYSERV    RMODE  ANY
          USING  *,15
          B      PROLOG                Branch around header text
          DC     C'some text'
          DC     C'compile date/time'
PROLOG    EQU    *
          DROP   15
          BSM    R14,0
          BAKR   14,0                  Save callers info on stack
          LAE    12,0                  Clear access register 12
          LR     12,15                 Load reg 15 into 12
PROGSTRT  EQU    *
          USING  MYSERV,12             Set up base register
*                                       addressability
          .
          .
          .
          Get dynamic area for program
          .. STORAGE OBTAIN or CELLPOOL or own scheme ...
          .
          .
          Free dynamic area for program
          .
          .
          .
RETURN    L      0,REASON_CODE         Put reason code in reg 0
          L      15,RETURN_CODE        Put return code in reg 15
          PR
```

Figure 9. Example of a Service Entry and Exit

The example uses the instructions BAKR and PR to replace standard linkage. With these instructions, you no longer need to pass the save area in a register.

If the callable service ends abnormally, ICSF takes a system dump. The ICSF service functional recovery routine (FRR) PROTECTS an installation-defined service. You can, however, write your own recovery routine.

Security access control checking

For the ICSF-defined services, ICSF performs security access control checking to determine if the caller is authorized to access the service and the results of the authorization check can be logged in SMF. This checking is not performed by ICSF for installation-defined services or UDXs. Any security access control checking must be performed by the installation-defined service or UDX itself.

Checking the parameters

For the ICSF-defined services, ICSF checks the integrity of user-passed parameters. An error in a parameter that causes a system abend does not cause a system dump. For an installation-defined callable service, you must perform your own integrity checking of parameters. An error in a user parameter that results in a system abend causes a system dump. You can suppress the system dump by setting a bit on in the SPB. To suppress the dump, set the bit on before you check the integrity of the parameters. This bit (the SPBTERM bit) is the third bit of the flag byte at offset 16 in the SPB.

Link-Editing the callable service

After you write the callable service, you need to link-edit it into a load module, and install the load module into an APF authorized library. ICSF uses this normal search order to locate the service:

- Job pack area
- Steplib (if one exists)
- Link pack area (LPA)
- Link list (SYS1.LINKLIB concatenation)

Defining a callable service

Use the SERVICE keyword in the installation options data set to specify information about the callable service. ICSF uses this information at ICSF startup to enable the service. See “Steps to create the Installation Options Data Set” on page 26 for more information about ICSF installation options.

The SERVICE keyword has this syntax:

SERVICE(*service-number***,***load-module-name***,FAIL(***fail-option***))**

The service-number is a number that identifies the service to ICSF. The valid service numbers are 1 through 32767, inclusive. The load-module-name is the name of the module that contains the service your installation wrote. During ICSF startup, ICSF loads the module and binds it to the service number you specified.

Using the fail-option, you specify the action ICSF takes if the loading of the service ends abnormally. ICSF loads all installation-defined services at ICSF startup.

Specify one of these values for the fail-option:

YES ICSF abends if your service cannot be loaded.

NO ICSF continues to start if your service cannot be loaded.

If the callable service ends abnormally while it is processing, ICSF does not end.

This SERVICE installation option statement identifies a specific installation-defined service to ICSF:

```
SERVICE(50,KSUST,FAIL(NO))
```

When ICSF starts, it binds the service number 50 to the load module KSUST, which contains the callable service you wrote. Because the fail option is NO, if your service cannot be loaded, ICSF continues to start anyway.

Writing a service stub

Besides writing the callable service itself, you must write a service stub, which is the connection between the application program and the installation-defined service. In an application program, you call the service stub, which accesses the installation-defined service. The service stub can be any name you choose to call it.

The service stub must:

- Check that ICSF is active.
- Place the service number for the installation-defined callable service into register 0.
- Call the IBM-supplied processing routine, CSFAPRPC.

CSFAPRPC is used to access the callable services on ICSF. In the service stub, you must call CSFAPRPC. ICSF stores the address of the CSFAPRPC entry point in the CCVTPRPC field of the ICSF cryptographic communication vector table (CCVT). If running in a CICS address space, then, after you call CSFVCCPP, the system calls the callable service that corresponds to the service number in register 0. “The Cryptographic Communication Vector Table (CCVT)” on page 266 describes the format of the CCVT.

The contents of the registers on entry to the service stub are:
Register 0 Unpredictable
Register 1 Address of the parameter list
Register 2–13 Unpredictable
Register 14 Return address
Register 15 Service stub entry point address

The contents of the registers on exit from the service stub are:
Register 0 Reason code
Register 1–14 Same as on entry
Register 15 Return code

To run an installation-defined callable service, an application program calls the service stub. You must link-edit the service stub with the application program that calls the service stub. Any application program that calls a service stub must be link-edited with the service stub.

To call an installation-defined service from an application program, use this statement:

```
CALL  <service-stub-name> <service-parameters>
```

The service-stub-name is the name of the service stub for the installation-defined callable service. The service-parameters are the parameters you want to pass to the installation-defined service. You supply the parameters according to the syntax of the programming language that you use to write the application program.

Example of a Service Stub

Figure 10 shows an example of a service stub for an installation-defined callable service.

```
**** START OF SPECIFICATIONS *****************************************
*                                                                     *
*    MODULE NAME = CSFGEN                                             *
*    DESCRIPTIVE NAME = SERVICE STUB                                  *
*                                                                     *
*    FUNCTION =                                                       *
*    THIS IS A SAMPLE SERVICE STUB. IT IS MEANT TO BE LINKEDITED      *
*    WITH THE APPLICATION AND ENTERED VIA A CALL CSFGEN. THIS STUB    *
*    CAUSES THE EXECUTION OF THE SERVICE WITH SERVICE NUMBER = 50     *
*    (DECIMAL).                                                       *
*    MODULE TYPE = ASSEMBLER                                          *
*       PROCESSOR = ASSEMBLER                                         *
*       MODULE SIZE = ONE BASE REGISTER                               *
*                                                                     *
**** END OF SPECIFICATIONS *******************************************
 CSFGEN  START 0
 GENSNUM EQU   50
 CSFGEN  CSECT
 CSFGEN  AMODE 31
 CSFGEN  RMODE ANY
 MAINENT DS    0H
         USING *,R15
         LAE   R15,0(R15,0)
         L     R15,=A(CICSTEST)
         BAKR  0,R15              PR from CICSTEST will restore GPRs
         LTR   R15,R15
         BC    2,NOCICS
 *
YESCICS DS    0H
         SAC   0
         STM   R14,R12,12(R13)
         LR    R12,R15
         DROP  R15
         USING MAINENT,R12
         LR    R3,R0
         B     NORMAL
       *
         NOCICS  DS    0H
         USING MAINENT,R12
         BSM   R14,0
         BAKR  R14,0
         LAE   R12,0
         LR    R12,R15
         SLR   R13,R13
 ********************************************************************
 * At this point, R0 must contain the service number.
 *                If we are to call the TRUE, R13 is non-zero
 *                R1 points to the caller's parameter list.
 ********************************************************************
 NORMAL  DS    0H
         LA    R0,GENSNUM           R0 gets service number
         SLR   R10_ZERO,R10_ZERO
         LR    RC,R10_ZERO
         L     R2,CVTPTR
         USING CVT,R2
         L     R2,CVTABEND
```

Figure 10. Example of a Service Stub (Part 1 of 5)

```
         CLR   R2,R10_ZERO
         BC    8,NOICSF
         USING SCVTSECT,R2
         L     R2,SCVTCCVT
         CLR   R2,R10_ZERO
         BC    8,NOICSF
         USING CCVT,R2
         TM    CCVTSFG1,B'00110000' IS ICSF ACTIVE
         BC    1,YESICSF
NOICSF   LA    RC,12               Set return code to 12 decimal
         L     R7,RETURN_CODE_PTR(,R1)
         ST    RC,RETURN_CODE(,R7)
         SLR   R0,R0
         L     R7,REASON_CODE_PTR(,R1)
         ST    R0,REASON_CODE(,R7)
         B     FINISHED
YESICSF  DS    0H
*********************************************************************
* Note that, if we're in CICS, the prolog code pointed R3 at the AFCB
* and R13 at the caller's savearea--they're still pointing.  Also, R0
* contains the service number, with the high order bit ON if the TRUE
* has been tried and found wanting. In this last case, CSFVCCPP will
* check the high order bit and not attempt to call the TRUE.
* If R13 is zero, we're using the linkage stack. That means we can
* call CSFAPRPC.
* If R13 is not zero, we're using non-stack linkage. That means the
* caller's savearea will be used. CSFVCCPP uses this kind of linkage.
* But note that CSFVCCPP won't return here. Instead, it will return
* directly to the caller--that is, to the owner of the only save
* area around.
 *********************************************************************
          CLR   R13,R10_ZERO
          BC    8,EXECPRPC
          L     R15,CCVTPRPD
          BALR  R14,R15
LR    RC,R15
          B     FINISHED
 EXECPRPC L     R15,CCVTPRPC
          BALR  R14,R15
          LR    RC,R15
 FINISHED DS    0H
 *
 ****************************************************************
 * This routine uses the linkage stack to save the caller's regs
 * if this is not a CICS environment.  In CICS, it uses the save
 * area pointed to by register 13.  So the epilog code takes one
 * of two forms.  If this is CICS (i.e. if R13 is non-zero),
 * return is via LM and BR 14.  If this is not CICS, return is
 * via PR.
 *
 * On return, the PR of ESA linkage does not restore registers
 * 0, 1, 14 and 15.  In the LM of normal BR 14 linkage, however,
 * everything but 13 gets restored.  Since this routine has no
 * autodata, there's no way to pass back return and reason codes
 * unless we leave 0 and 15 intact.  The solution is to deviate
 * slightly from normal BR 14 linkage and restore only registers
 * 1 through 12 and 14.
 ****************************************************************
           LTR   R13,R13
           BC    8,ENDNOCICS
```

Figure 10. Example of a Service Stub (Part 2 of 5)

```
ENDCICS    LR    R15,RC
           L     R14,SAVE14(,R13)
           LM    R1,R12,24(R13)
           BR    R14
 *
EDNOCICS   DS    0H
           LR    R15,RC
           LA    R7,12
           CR    R15,R7
           BNE   ENDSVC
           LA    R7,16
           CR    R0,R7
           BNE   ENDSVC
           L     R7,RETURN_CODE_PTR(,R1)
           ST    R15,RETURN_CODE(,R7)
           L     R7,REASON_CODE_PTR(,R1)
           ST    R0,REASON_CODE(,R7)
ENDSVC     LR    R15,RC
           PR
 ********************************************************************
 ********************************************************************
 **  CICSTEST:  Decides whether this is a CICS environment
 ********************************************************************
 ********************************************************************
 CICSTEST DS   0H
         LAE   R12,0               Clear AR 12
         LR    R12,R15             Addressability via R12
         USING CICSTEST,R12
         L     R15,=A(CSFGEN)       R15 gets caller's base reg
         L     R2,CVTPTR            GET CVT POINTER
         USING CVT,R2
         L     R2,CVTABEND          AND SECONDARY CVT POINTER
         USING SCVTSECT,R2
         L     R2,SCVTCCVT          POINT TO CSF CCVT
         LTR   R2,R2                IS CRYPTO INSTALLED?
         BZ    RETRN                IF NOT, GO HOME
         USING CCVT,R2
         TM    CCVTSFG1,B'00110000' IS ICSF ACTIVE
         BNO   RETRN                IF NOT , GO HOME

 * Check for wait list routine
 *
         TM    CCVTCICS,B'10000000' Q. CCVTPRPA ON?
         BZ    RETRN                     no---No CICS capability
         TM    CCVTCICS,B'01000000' Q. CCVTCKWL ON?
         BZ    CKWLHERE                  no---use imbedded routine
 *                                          yes--use installed routine
         LA    R0,GENSNUM           R0 gets service number
         LR    R3,R1                R3 saves R1
         LR    R4,R14               R4 saves R14
         LR    R5,R15               R5 saves R15
         L     R15,CCVTCKWL         R15 gets routine address
         BALR  R14,R15              Go check for CICS
         LR    R0,R15               Save return code in R0
         LR    R15,R5               Restore R15
         LR    R14,R4               Restore R14
         LR    R1,R3                Restore R1
         LTR   R0,R0                Q. CICS?
         BZ    RETRN                  no---return
 *                                    yes--pass info along
         O     R15,M_CICS           Enable high bit of R15 to CICS
         B     RETRN                Return
```

Figure 10. Example of a Service Stub (Part 3 of 5)

```
* Cannot use installed routine.  Use imbedded routine
*
CKWLHERE DS    0H                  Imbedded check for TRUE routine
         SLR   R0,R0               Init R0 to 0
         CPYA  R8,R12              Zero AR 8
         SLR   R8,R8               Init R8 to 0
         USING PSA,R8
         L     R8,PSATOLD          R8->TCB
         USING TCB,R8
         LTR   R8,R8               Q.  Is there a TCB?
         BC    8,RETRN               no---return
*                                    yes--check state and key
         CPYA  R11,R12             Zero AR 11
         LA    R11,1               Get PSW state and key in R6
         ESTA  R6,R11
         LR    R7,R6               Copy of state & key in R7
         N     R7,M_KEY            Q.  problem key?
         BZ    RETRN                 no---return
*                                    yes--check state
         N     R6,M_STATE          Q.  problem state?
         BZ    RETRN                 no---return
*                                    yes--get the CICS eye-catcher
         LA    R6,2                Set ARs 6 and 8 to home
         SAR   R6,R6
         SAR   R8,R6
         L     R8,TCBEXT2          R8->TCB extension
         USING TCBXTNT2,R8
         ICM   R4,B'1111',TCBCAUF  R4 gets AFCX address
*                                  Q.  Address there?
         BZ    RETRN                     no---return
*                                        yes--check eye-catch
         CLC   0(4,R4),CICS_EYE    Q.  CICS?
         BNE   RETRN                 no---return
*                                    yes--pass info along
         LR    R0,R4              R0 gets the AFCX pointer
         O     R15,M_CICS          Enable high order bit of R15
RETRN    DS    0H
         DROP  R12                  Free R12
          PR                        Return from CICSTEST subroutine
*
          LTORG
          DS    0D
*
GENSDATA DS    0F
R10_ZERO EQU   10
RC       EQU   05
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
*
```

Figure 10. Example of a Service Stub (Part 4 of 5)

```
INPUT_PARMS     EQU 0,8,C'C'
RETURN_CODE_PTR EQU INPUT_PARMS,4,C'A'
REASON_CODE_PTR EQU INPUT_PARMS+4,4,C'A'
RETURN_CODE     EQU 0,4,C'F'
REASON_CODE     EQU 0,4,C'F'
*
SAVAREA  EQU   0,72,C'C'
SAVE14   EQU   SAVAREA+12,4,C'A'
SAVE01   EQU   SAVAREA+24,4,C'A'
SCVTSPTR EQU   CVTABEND,4,C'F'
TCBPTR   EQU   PSATOLD,4,C'F'
         DS    0D
*
         DS    0F                   Align
M_KEY    DC    X'00800000'         Problem key mask
M_STATE  DC    X'00010000'         Problem state mask
M_NOCICS DC    X'7FFFFFFF'         Not-CICS mask
M_CICS   DC    X'80000000'         Yes-CICS mask
         DS    0D
CICS_EYE DC    CL4'AFCX'           CICS eye catcher
*
         IHAPSA
         TITLE 'DSECT CVT'
         CVT DSECT=YES
         TITLE 'DSECT SCVT'
         IHASCVT DSECT=YES
         TITLE 'DSECT TCB'
         IKJTCB
         TITLE 'DSECT CCVT'
         CSFCCVT
*
         LA R7,12
         CR R15,R7
         BNE ENDGSVC
         LA R7,16
         CR R0,R7
         BNE ENDGSVC
         L R7,RETURN_CODE_PTR(,R1)
         ST R15,RETURN_CODE(,R7)
         L R7,REASON_CODE_PTR(,R1)
         ST R0,REASON_CODE(,R7)
         ENDGSVC DS 0H

         END
```

Figure 10. Example of a Service Stub (Part 5 of 5)

In Figure 10 on page 163, the service stub, CSFGEN, checks that ICSF is active, places the service number 50 into register 0, and calls CSFAPRPC.

The service number 50 (in the case of this example) must be bound to the installation-defined service by using the SERVICE keyword in the installation options data set. The service number is bound to the service when ICSF interprets the SERVICE installation option statement and loads the service at ICSF startup. To run the callable service that is associated with service number 50, call the service stub CSFGEN from an application program.

For flexibility, to create a service stub for a different installation-defined callable service, you can copy an existing service stub and just change the service number that you load into register 0.

Chapter 7. Converting a CKDS from fixed length to variable length record format

ICSF provides a CKDS conversion program, CSFCNV2, that converts a fixed length record format CKDS to a variable length record format. There will be no changes to the key token in the CKDS record. Only the format of the record will be changed.

Note: You can also use the CSFCNV2 utility to rewrap encrypted DES values in the CKDS. For more information on this capability of the CSFCNV2 utility, refer to *z/OS Cryptographic Services ICSF Administrator's Guide*.

There is no conversion from variable length to fixed length records.

You run the conversion utility program by submitting a batch job. On the EXEC statement, specify PGM=CSFCNV2.

This example is a JCL that runs the conversion program:

```
//CKDSCNV2 EXEC PGM=CSFCNV2,PARM='FORMAT,OLD.CKDS,NEW.CKDS'
```

Where:

OLD.CKDS
: The fixed length record format CKDS to be converted. This is the source CKDS for the conversion.

NEW.CKDS
: An empty disk copy of a variable length record format CKDS. This is the CKDS into which the conversion utility writes the converted records. The data set must be defined and empty before you run the conversion program.

Refer to the SYS1.SAMPLIB CSFCKD2 member sample described in “Steps to create the CKDS” on page 17 for example JCL that defines a VSAM CKDS for variable length records.

The CSFV0560 message in the joblog will indicate the results of processing.

Return Code	Meaning
0	Process successful.
4	Minor error occurred.
8	RACF authorization check failed.
12	Process unsuccessful.
60 or 92	CKDS processing has failed. A return code 60 indicates the error was detected in the new KDS. A return code 92 indicates the error was detected with the old KDS.

When the program is invoked from another program, the invoking program receives the reason code in General Register 0 along with the return code in General Register 15. The following list describes the meaning of the reason codes. If a

particular reason code is not listed, refer to the listing of ICSF and TSS return and reason codes in the *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Return code 0 has this reason code:

Reason Code	Meaning
36132	CKDS reencipher/Change MK processed only tokens encrypted under the DES master key.

Return code 4 has these reason codes:

Reason Code	Meaning
0	Parameters are incorrect.
4004	Rewrapping is not allowed for one or more keys.
36112	CKDS conversion completed successfully but some tokens could not be rewrapped because the control vector prohibited rewrapping from the enhanced wrapping method.
36164	Input CKDS is already in the variable-length record format. No conversion is necessary.

Return code 8 has this reason code:

Reason Code	Meaning
16000	Invoker has insufficient RACF access authority to perform function.

Return code 12 has these reason codes:

Reason Code	Meaning
0	ICSF has not been started
11060	The required cryptographic coprocessor was not active or the master key has not been set
36000	Unable to change master key. Check hardware status.
36008	Crypto master key register(s) in improper state.
36020	Input CKDS is empty or not initialized (authentication pattern in the control record is invalid).
36036	The new master key register for Coprocessor 1 (C1) is not full, but C0 is ready and the current master key is valid.
36040	The new master key register for C0 is not full, but C1 is ready and the current master key is valid.
36044	The master key authentication pattern for the CKDS does not match the authentication pattern of the coprocessors, which are not equal.
36048	The master key authentication pattern for the CKDS does not match the authentication pattern of either of the coprocessors, which are not equal.
36052	A valid new master key is present in C0, but its authentication pattern does not match that of C1 or the CKDS, which are equal.

36056 A valid new master key is present in C1, but its authentication pattern does not match that of C0 or the CKDS, which are equal.

36060 The new master key register(s) is/are not full.

36064 Both new master key registers are full but not equal.

36068 The input KDS is not enciphered under the current master key.

36076 The new master key register for C0 is not full, but the CPUs are online.

36080 The new master key register for C1 is not full, but the CPUs are online.

36084 The master key register cannot be changed since ICSF is running in compatibility mode.

36104 Option not available. There were no Cryptographic Coprocessors available to perform the service that was attempted.

36108 PKA callable services are enabled, and the PKDS is the active PKDS as specified in the options data set.

36120 The CKDS is unusable. The CKDS does not support record level authentication.

36124 The CKDS is unusable. The CKDS only supports encrypted AES keys and encrypted DES support is required.

36128 The CKDS is unusable. The CKDS does not support encrypted DES keys which is required.

36160 The attempt to reencipher the CKDS failed because there is an enhanced token in the CKDS.

36168 A CKDS has an invalid LRECL value for the requested function. For wrapping, the input and output CKDS LRECLs must be the same.

36172 The level of hardware required to perform the operation is not available.

Return code 60 or 92 has these reason codes:

Reason Code	Meaning
3078	The CKDS was created with an unsupported LRECL.
5896	The CKDS does not exist.
6008	A service routine has failed. The service routines that may be called are: **CSFMGN** MAC generation **CSFMVR** MAC verification **CSFMKVR** Master key verification
6012	The single-record, read-write installation exit (CSFSRRW) returned a return code greater than 4.
6016	An I/O error occurred reading or writing the CKDS.
6020	The CSFSRRW installation exit abended and the installation options EXIT keyword specifies that the invoking service should end.

6024 The CSFSRRW installation exit abended and the installation options EXIT keyword specifies that ICSF should end.

6028 The CKDS access routine could not establish the ESTAE environment.

6040 The CSFSRRW installation exit could not be loaded and is required.

6044 Information necessary to set up CSFSRRW installation exit processing could not be obtained.

6048 The system keys cannot be found while attempting to write a complete CKDS data set.

6052 For a write CKDS record request, the current master key verification pattern (MKVP) does not match the CKDS header record MKVP.

6056 The output CKDS is not empty.

Note: It is possible that you will receive MVS reason codes rather than ICSF reason codes, for example, if the reason code indicates a dynamic allocation failure. For an explanation of Dynamic Allocation reason codes, see *z/OS MVS Programming: Authorized Assembler Services Guide*

Chapter 8. Migration from PCF to z/OS ICSF

If your installation uses the cryptographic product, Programmed Cryptographic Facility (PCF), ICSF helps you migrate PCF applications to ICSF. You can run PCF applications on ICSF to gain the enhanced performance and availability of ICSF and to test ICSF. Eventually, you should convert these applications to use ICSF services, rather than the PCF macros.

During migration, you can run PCF applications on ICSF because ICSF continues to support the PCF macros (GENKEY, RETKEY, EMK, and CIPHER). If GENKEY or RETKEY macro exits exist, you should reevaluate their applicability to ICSF. If an exit performs a necessary function, you need to rewrite the exit for ICSF. Exits exist for the compatibility services on ICSF.

If a PCF application uses a key in the PCF cryptographic key data set, you must convert the key to an ICSF cryptographic key data set before you run the PCF application on ICSF. ICSF provides a program to make this conversion.

Running PCF and z/OS ICSF on the same system

You can run PCF and ICSF simultaneously on the same z/OS system or separately in three different modes. You can run ICSF in compatibility, coexistence, or noncompatibility mode.

In compatibility mode, you can run either PCF or ICSF, but you cannot run them simultaneously on the same z/OS system. You can continue to run PCF applications on PCF or you can run PCF applications on ICSF. ICSF supports the PCF macros that the PCF applications call. However, you cannot run the PCF key generator utility program (KGUP) on ICSF. You do not have to reassemble PCF applications to run the applications on ICSF.

In coexistence mode, you can run PCF and ICSF simultaneously on the same z/OS system. You can continue to run a PCF application on PCF or you can reassemble the PCF application to run on ICSF. In this mode, ICSF supports the PCF macros when a reassembled PCF application calls these macros.

In noncompatibility mode, you can run PCF and ICSF simultaneously and independently on the same z/OS system. You can run PCF applications on PCF and ICSF applications on ICSF. You cannot run PCF applications on ICSF, because ICSF does not support the PCF macros in this mode.

You can run PCF simultaneously and independently in coexistence and noncompatibility mode. Therefore, in these modes, you can run PCF KGUP on PCF while running ICSF. The PCF KGUP updates keys on a PCF CKDS.

The ICSF installation option COMPAT(YES, COEXIST or NO) allows you to specify which mode you want ICSF to run in. You specify COMPAT(YES) for compatibility mode, COMPAT(COEXIST) for coexistence mode, and COMPAT(NO) for noncompatibility mode. See “Steps to create the Installation Options Data Set” on page 26 for information about creating the installation options data set and “Parameters in the installation options data set” on page 38 for details about these options.

Running in Compatibility Mode

In compatibility mode, you can run a PCF application on ICSF without reassembling the application. A PCF application running on ICSF can still use PCF macros, because ICSF supports these macros. The PCF application gains the enhanced performance, reliability, and availability of ICSF.

You cannot run PCF and ICSF simultaneously on the same z/OS system in compatibility mode. If you start PCF, you must stop PCF before you can start ICSF. If you start ICSF, you must stop ICSF before you can start PCF.

A PCF application may have used keys on the PCF cryptographic key data set (CKDS). When you run the application on ICSF, these keys must be in the ICSF CKDS. The format of a key entry on the PCF CKDS differs from the format of a key entry on the ICSF CKDS. Therefore, you need to run a conversion program to convert the PCF CKDS entries and place the entries in the ICSF CKDS. See "Converting a PCF CKDS to ICSF format" on page 177 for a description of how to convert a PCF CKDS.

For encryption, ICSF supports the Data Encryption Standard (DES), the Commercial Data Masking Facility (CDMF), or both. Wherever possible, the key token is marked to signal which algorithm to use.

PCF macros receive identical error return codes if they run on ICSF or PCF, with one exception. If a key is installed on the ICSF CKDS with the correct label but with the wrong key type, an attempt to use that key by RETKEY or GENKEY results in a return code of 8 from PCF. This indicates that the key was not of the correct type. ICSF issues return code 12, indicating that it could not find the key. Ensure that PCF LOCAL or CROSS 1 keys are installed in the ICSF CKDS as EXPORTER keys. Also, ensure that REMOTE and CROSS 2 keys are installed in the ICSF CKDS as IMPORTER keys.

In compatibility mode, the safest method for changing the master key is to re-IPL the system. To change the master key in compatibility mode, see "Changing the master key in compatibility or coexistence mode" on page 175.

Note: To use AMS REPRO encryption, you need to run ICSF in compatibility mode.

Running in Coexistence Mode

In coexistence mode, you can run ICSF and PCF simultaneously on the same z/OS system and run a PCF application on PCF or on ICSF. A PCF application running on ICSF gains the enhanced performance, reliability, and availability of ICSF.

A PCF application running on ICSF can still use PCF macros, because ICSF supports these macros. ICSF ships changed PCF macros in SAMPLIB that run only on ICSF. Because these changed PCF macros already exist unchanged on PCF, the changed PCF macros shipped with ICSF are named differently.

On ICSF, in SAMPLIB:
- The changed PCF EMK macro is named CSFEMK.
- The changed PCF CIPHER macro is named CSFCIPH.
- The changed PCF RETKEY macro is named CSFRKY.
- The changed PCF GENKEY macro is named CSFGKY.

You can rename these macros to the PCF names when you want to run a PCF application on ICSF.

To run a PCF application on ICSF, you must:

- Rename the changed PCF macro shipped in ICSF SAMPLIB to the appropriate PCF name.
- Place the macro in the appropriate macro library.
- Reassemble the PCF application against the changed PCF macro.

Then the application can run only on ICSF. To run a PCF application on PCF, just run the application without reassembling the application.

During migration, you can start ICSF and start PCF so that both products are running simultaneously. If you want to run a PCF application using the PCF macros on PCF, do not reassemble the application. If you want to run a PCF application using the changed PCF macros on ICSF, reassemble the application against the changed macros. Coexistence mode enables you to run the products simultaneously and choose whether to run a PCF application on PCF or ICSF.

A PCF application can use keys on the PCF CKDS. When you run the application on ICSF, those keys must be in the ICSF CKDS. The format of a key entry on the PCF CKDS differs from the format of a key entry on the ICSF CKDS. Therefore, you need to run a conversion program to convert the PCF CKDS entries and place the entries in the ICSF CKDS. See “Converting a PCF CKDS to ICSF format” on page 177 for a description of how to convert a PCF CKDS.

In coexistence mode, the safest method for changing the master key is to re-IPL the system. See “Changing the master key in compatibility or coexistence mode” for a description of the process used to change the master key in coexistence mode.

Changing the master key in compatibility or coexistence mode

In compatibility and coexistence modes, the safest way to activate a master key after changing it is to re-IPL the system. This process is different from the usual process for entering and activating a master key. For information about changing the master key, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

A re-IPL ensures that a program does not access a cryptographic service with a key that is encrypted under a different master key. If a program is using an operational key, the program either re-creates the key or imports the key again.

In compatibility or coexistence mode, the ICSF administrator can use the ICSF panels to enter the key value into the new master key register. However, the master key cannot be *activated* using the panels in compatibility or coexistence mode. The value entered remains in the new master key register until you re-IPL the system. (In noncompatibility mode, the ICSF administrator can use the ICSF panels to enter the key value into the new master key register and to activate the master key.)

If the new master key is different than the current master key, the ICSF administrator must reencipher the CKDS under this new master key. To do this, choose the change option on the master key management panel. This reenciphers a CKDS under the master key in the new master key register. Reencipher all the disk copies of the CKDSs, and leave the ICSF panels without changing the master key.

Then re-IPL the system and restart ICSF. In the installation options data set, the CKDSN installation option must specify a disk copy of the CKDS that is reenciphered under the new master key. When ICSF starts again, it detects that the current master key is not the one that enciphered the CKDS that is specified in the

installation options data set. ICSF detects that the CKDS is enciphered under the new master key and makes that master key active.

If your installation requires 24-hour availability and it is not possible to re-IPL the system, an alternative method is to stop all cryptographic applications, especially those using PCF macros. This helps eliminate inadvertent use of operational keys that are encrypted under the old master key. After you restart CSF, applications using an operational key can either re-create or reimport the key.

Running in noncompatibility mode

In noncompatibility mode PCF and ICSF can run simultaneously and independently. You can run both ICSF and PCF at the same time. Just start one and then the other. Both ICSF and PCF run completely separate from each other. Each has its own applications and each uses its own services and CKDS.

You cannot run a PCF application on ICSF, even if you reassemble it. If you run an application on ICSF that calls a PCF macro, the application ends abnormally, because ICSF does not support the PCF macros in noncompatibility mode.

Because each product runs separately, neither product loses any function in exchange for compatibility. When ICSF is in compatibility or coexistence mode, you can no longer change the master key dynamically. In noncompatibility mode, this function is still possible. Therefore, except for when your installation is migrating to ICSF, you probably want to run ICSF in noncompatibility mode.

Note: When you initialize ICSF for the first time, noncompatibility mode must be active.

Specifying compatibility modes during migration

The process and duration to migrate from PCF to ICSF depend on your installation. You can use different modes in different stages of migration. To change modes, change the COMPAT option in the installation options data set and restart ICSF. When you complete migration to ICSF, you can run in noncompatibility mode to use the full function of ICSF.

When you first install an ICSF system, you can continue to run PCF for production and just test ICSF. Because you are running the products separately but simultaneously on the same z/OS system, you can run in noncompatibility or coexistence mode. To run in compatibility mode, you need more than one z/OS system. You can run the test applications on ICSF on one z/OS system while you run your production on PCF on another z/OS system.

When you begin testing ICSF, you can run existing applications in either compatibility mode or coexistence mode to test the PCF macros on ICSF. After you run the test applications, you may want to bring up production using PCF applications on ICSF. When you bring over PCF applications to ICSF, you can run in coexistence mode. In this mode, you can run an application on PCF and then reassemble the application to run the application on ICSF.

While, or after, you bring PCF applications into production on ICSF, you can run test applications that call ICSF services. You can then convert the applications that call PCF macros to applications that call the ICSF services. The ICSF services provide enhanced key separation, performance, and function. After you convert all your PCF applications to ICSF applications, you can activate noncompatibility mode and have the full function of ICSF.

Converting a PCF CKDS to ICSF format

During migration, you may need to convert a PCF CKDS into ICSF CKDS format if:

- PCF applications running on ICSF use keys stored in a PCF CKDS.
- Your installation uses the PCF key generator utility program to create keys and uses ICSF for other cryptographic operations. To use the keys in ICSF applications, you must convert the PCF CKDS.

ICSF provides a PCF conversion program, CSFCONV, that converts a PCF CKDS into an ICSF CKDS. The conversion program runs with certain defaults. The program converts all the entries in a PCF CKDS and converts the PCF key types into certain corresponding ICSF key types. You can use the conversion program override file to instruct the conversion program not to convert certain entries. You can also tell the conversion program to convert a PCF key type into a different ICSF key type than the default.

These topics describe how:

- The conversion program runs with certain defaults
- To use the override file to make it run differently
- To run the conversion program

How the PCF conversion program runs

You can run the PCF conversion program only after you initialize the master key and CKDS for ICSF. To run the conversion program on CCF systems, the CKDS you specify at ICSF startup must be initialized to contain NOCV-enablement keys. For non-CCF systems, NOCV-enablement keys are not required. For information about master key and CKDS initialization and NOCV-enablement keys, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

When the conversion program processes a PCF CKDS, the program duplicates the single length key values to create double length keys.

The conversion program merges the PCF CKDS with an input ICSF CKDS. The input ICSF CKDS is an existing disk copy of an ICSF CKDS. The input ICSF CKDS must contain a header record and include system keys entries, but may or may not contain other key entries. ICSF uses NOCV enablement keys to create keys to communicate with systems that do not use control vectors. If the ICSF CKDS resulting from the conversion contains converted importer or exporter key-encrypting keys, the input ICSF CKDS must contain NOCV enablement keys. For information about initializing an ICSF CKDS, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

The PCF conversion program places the input ICSF CKDS entries and the converted PCF entries into an output CKDS. You must create an empty VSAM data set to be the output CKDS before running the conversion program. See “Steps to create the CKDS” on page 17 for information about creating the data set.

The PCF conversion program converts all the entries in a PCF CKDS. When you run the PCF conversion program, the program does these conversions of PCF key types into ICSF key types:

- Converts each PCF local key entry into an ICSF NOCV exporter key-encrypting key entry.
- Converts each PCF remote key entry into an ICSF NOCV importer key-encrypting key entry.

- Converts each PCF cross key entry into two ICSF key entries: an NOCV exporter key-encrypting key and an NOCV importer key-encrypting key.

You use the override file to not convert all the entries in a PCF CKDS or to convert a PCF key into a different key type than the default key type.

When the PCF conversion program converts a PCF entry, the program places any installation data from the installation data field of the PCF entry into the ICSF entry. You can use the override file to place different installation data into the ICSF entry.

Note: ICSF copies any installation data in the input CSF CKDS header record into the output ICSF CKDS header record.

As the conversion program reads the PCF CKDS, the input ICSF CKDS, and the override file, the program places key entries into a virtual image of the output ICSF CKDS. When the virtual image CKDS is complete, the conversion program reenciphers the key values of the PCF entries from under the PCF master key to under the ICSF master key. The conversion program places the reenciphered entries into the actual output CKDS.

As the conversion program creates the virtual image ICSF CKDS, the conversion program takes information from the PCF entry and possibly the override file. For each PCF entry, the conversion program checks if its key label exists in the override file. If the label does exist in the override file, the conversion program takes the action that is specified in the override file. The program either converts or bypasses the entry. If the key label does not exist in the override file, ICSF converts the entry.

The conversion program compares the converted PCF entries by label and type with the ICSF entries that already exist in the input ICSF CKDS. If there is a match, the conversion program replaces the key value from the converted entry of the PCF source into the virtual image CKDS. If there is not a match, the conversion program converts each PCF entry after checking the override file. If the label matches and the type does not, the conversion program checks to see if the type requires a unique label. If a unique label is not required, the conversion program converts the PCF entry after checking the override file. If a unique label is required, the conversion program does not convert the PCF entry and issues an error message. If the record type is DATA, DATAXLAT, MAC, MACVER, or NULL the CKDS record requires a unique label. The CKDS record also requires a unique label if the record has ever been updated by the dynamic CKDS update callable services. The conversion program also places all the input ICSF CKDS entries into the virtual image CKDS.

Calling installation exits during conversion

You can call two installation exits during conversion program processing: the conversion program exit (CSFCONVX) and the single-record, read-write exit (CSFSRRW). The conversion program calls the exit at three different times: before, during, and after conversion program processing. See Chapter 5, “Installation Exits,” on page 111 for a description of the conversion program and single-record, read-write exit control blocks.

The conversion program calls the CSFCONVX exit after you submit the conversion program job, but before the program actually begins processing. At this point, you can use the exit to change the output ICSF CKDS header record installation data field.

The conversion program also calls the CSFCONVX exit during processing as the conversion program completes the virtual image ICSF CKDS, but before the conversion program reenciphers the key values. The conversion program calls the exit as it writes each record to the virtual image ICSF CKDS. At this point, you can use the exit to specify that the conversion program not place an entry into the output ICSF CKDS.

The conversion program also calls the CSFCONVX exit after the conversion program completes processing. At this point, you can use the exit to change the output ICSF CKDS header record installation data field.

As the conversion program reads the records from the virtual image ICSF CKDS to the actual output ICSF CKDS it calls the single-record, read-write exit. The conversion program calls the single-record, read-write exit as it writes each record to the output ICSF CKDS. You can use this exit to specify that the conversion program not place an entry into the output ICSF CKDS.

The conversion program writes every entry from the PCF CKDS and input ICSF CKDS into the output ICSF CKDS unless an override record or installation exit indicates that the conversion program should bypass the entry from the PCF CKDS.

Using the conversion program override file

The conversion program converts all entries in a PCF CKDS into ICSF entries. The conversion program also converts each type of PCF key into a specific ICSF key type. If you want the conversion program to bypass certain key entries or convert a specific key or key type differently than it does by default, use the override file.

By specifying override records, you can have the conversion program:
- Bypass conversion of key entries
- Include information in key entries
- Convert key types differently than it does by default

These actions can relate to entries explicitly identified with a key label or entries that are identified globally.

You specify information in certain fields in an override record and leave other fields blank, depending on the action you want the conversion program to take. You can specify a global record affecting more than one PCF CKDS entry or a record that affects only one PCF CKDS entry.

All the override data set records should be in ascending sequence by key label and old key type. If you use global entries, they must be the initial entries in the override record. Table 24 on page 180 shows the syntax of a record in the override file.

Note: All the fields should contain character values and be left-justified.

If you specify a key label in an override record, the conversion program processes the key entry identified by that key label. If you do not specify a key label in an override record, you are using a global override record. The conversion program processes all the key labels that pertain to the information specified by the override file.

You can use a global override record to affect all the entries in a CKDS and then use override records to explicitly affect entries you did not want to have that global override record affect.

Table 24. Format of Records in the Override File

Column	Length	Description
1	8	***Key Label*** The key label of the PCF entry you want to convert The field can have these values: • Blanks • A key label existing in the PCF CKDS that you want to convert
9	1	This field must be blank.
10	8	***Old Key Type*** The key type of the key entry you want to convert in the PCF CKDS. The field can have these values: • Blanks • LOCAL • REMOTE
18	1	This field must be blank.
19	8	***New Key Type*** The key type that you want the converted key entry to be in the ICSF CKDS. The master key variant for the key type enciphers the key in the ICSF CKDS entry that the conversion program creates. The field can have these values: • Blanks • OPINENC • EXPORTER • IPINENC • IMPORTER
27	1	This field must be blank.
28	8	***Ignored*** In ICSF/MVS Version 1 Release 1, this field contained the key qualifier. The CKDS for ICSF/MVS Version 1 Release 2 or above does not support key qualifiers. If your installation has a PCF conversion program override file created with ICSF/MVS Version 1 Release 1, you can still use it with z/OS ICSF. Any key qualifier entries are ignored.
36	1	This field must be blank.
37	1	***Bypass Flag*** Used to indicate that an input CKDS entry is not to be included in the new ICSF CKDS. If you set this field to `Y`, the conversion program does not convert the entry. The field can have these values: • Blank (same as N) • N • Y
38	1	This field must be blank.

Table 24. Format of Records in the Override File (continued)

Column	Length	Description
39	52	***Installation Data*** Any additional information your installation records about a key. The information appears in the installation data field of the new ICSF CKDS. The field can contain any value.

Bypassing Conversion of Entries

Using an override record, you can bypass a PCF entry so it is not converted and placed in the ICSF CKDS. You can use a global override record to bypass all the entries in the data set and then use explicit override records to convert certain entries. You can also convert most of a PCF CKDS and just bypass certain entries using explicit override records.

These are some examples of override records for bypassing conversion.

Example 1: This example shows an override record specifying that the conversion program not convert any PCF CKDS entry with a certain key label.

```
EXTOATM3                        Y
```

The conversion program bypasses any PCF CKDS entry with the label EXTOATM3.

Example 2: This example shows an override record specifying that the conversion program not convert any PCF CKDS entry with a certain key label and key type.

```
CRLABEL4 REMOTE                 Y
```

The conversion program bypasses any PCF CKDS entry with the label CRLABEL4 and key type REMOTE.

Example 3: This example shows a global override record specifying that the conversion program bypass all the entries in a PCF CKDS.

```
                                Y
```

The conversion program does not convert any of the entries in the PCF CKDS.

After you specify this global override record, you can use explicit override records to convert certain entries in the PCF CKDS. For example, you can use an override record like this one to explicitly convert PCF entries with a certain label.

```
ATM03                           N
```

In this example, the conversion program converts any PCF CKDS entry with the label ATM03.

Example 4: This example shows a global override record specifying that the conversion program bypass all the entries with a certain PCF key type in a PCF CKDS.

```
REMOTE                          Y
```

The conversion program does not convert any of the entries with a key type of REMOTE in the PCF CKDS. After you specify this global override record, you can use explicit override records to convert specific entries with a key type of REMOTE in the PCF CKDS.

Including Information in a Key Entry

Programming Interface information

An ICSF key entry contains an installation data field that an installation can use to further identify a key. The installation data field contains any information that an installation wants to supply about a key.

PCF records contain an installation data field. The conversion program places the information in the field into the installation data field of the converted entry in the output ICSF CKDS. You can use an override record to specify installation data information for the converted entry in the output ICSF CKDS. The installation data information supplied in the override record overrides any information from the PCF installation data field. If you do not use an override record, the conversion program places any installation data from the PCF entry into the leftmost 8 bytes of the ICSF entry.

These are examples of override records for including key information.

Example 1: This example shows an override record providing the conversion program with installation data information to place in the ICSF CKDS for any converted PCF entry with a certain key label.

```
ATMKEY12                          CONVERTED FROM CUSP1.CKDS 10/01/98
```

When the conversion program converts an entry that is labeled ATMKEY12, it places `CONVERTED FROM CUSP1.CKDS 10/01/98` in the installation data field for the converted entry.

Example 2: This example shows an override record providing the conversion program with installation data information to place in the ICSF CKDS for any converted PCF entry with a certain key label and key type.

```
LOCAL890 LOCAL                    CONVERTED FROM PCF12.CKDS
```

When the conversion program converts an entry that is labeled LOCAL890 with a key type of LOCAL, it places `CONVERTED FROM PCF12.CKDS` in the installation data field for the converted entry.

Example 3: This example shows a global override record that provides the conversion program with installation data information to place in the ICSF CKDS for all converted entries.

```
                                  CONVERTED FROM PCF10.CKDS
```

When the conversion program converts the PCF CKDS, it places `CONVERTED FROM PCF10.CKDS` in the installation data field. The information is placed into every converted key entry. After you specify this global override record, you can use explicit override records to provide different information for specific entries in the PCF CKDS.

End of Programming Interface information

Converting Key Types

By default, the conversion program converts PCF keys into certain ICSF key types. You can use the override file to override the defaults. For example:

- Instead of automatically converting a PCF local key into a NOCV exporter key-encrypting key, you can convert the local key into an output PIN-encrypting key.

- Instead of automatically converting a PCF remote key into a NOCV importer key-encrypting key, you can convert the remote key into an input PIN-encrypting key.
- Instead of automatically converting a PCF cross key into a NOCV exporter key-encrypting key and a NOCV importer key-encrypting key, you can convert the cross key into an output PIN-encrypting key and an input PIN-encrypting key.

You can use a global override record to convert all keys of a certain type into a type other than the conversion program default key type. Then using an explicit override record, you can specify that the conversion program convert a specific record into a the default key type. For example, you can use a global override record to convert all remote keys into input PIN-encrypting keys, and then use an override record to convert specific remote entries into importer key-encrypting keys.

These are some examples of override records for key type conversion.

Example 1: This example shows an override record specifying that the conversion program convert a local key to an output PIN-encrypting key for any PCF CKDS entry with a certain key label. The override record also provides the conversion program with installation data.

```
CRLABEL1    LOCAL  OPINENC            OPINENC FOR ATM123
```

When the conversion program converts any PCF entry labeled CRLABEL1 with a key type of local, it converts the key from a local key type to an output PIN-encrypting key type. The program also places `OPINENC FOR ATM123` in the installation data field.

If you did not specify this override record, the conversion program would automatically convert the entry from a local key type to an exporter key-encrypting key type.

Example 2: This example shows an override record specifying that the conversion program convert a remote key to an input PIN-encrypting key for any PCF CKDS entry with a certain key label. The override record also provides the conversion program with installation data.

```
CRLABEL2   REMOTE  IPINENC            IPINENC FOR ATM123
```

When the conversion program converts any PCF CKDS entry labeled CRLABEL2 with a key type of remote, it converts the key from a remote key type to an input PIN-encrypting key type. The program also places `IPINENC FOR ATM123` in the installation data field.

If you did not specify this override record, the conversion program would automatically convert the entry from a remote key type to an importer key-encrypting key type.

Example 3: This example shows an override record specifying that the conversion program convert a local key to an exporter key-encrypting key for any PCF CKDS entry with a certain key label. The override record also provides the conversion program with installation data.

```
LOLABEL1 LOCAL    EXPORTER           EXPORTER CONVERTED FROM CUSP12.CKDS
```

The conversion program automatically converts a local key to an exporter key-encrypting key. You can use this override record if you previously submitted an override record that had the conversion program convert all the local key types to

output PIN-encrypting keys. You can use this override record to explicitly convert the key entry that is labeled LOLABEL1 from a local key type to an exporter key-encrypting key type.

With the example override record, when the conversion program converts any PCF entry labelled LOLABEL1 with a key type of local, it converts the key from a local key type to an exporter key-encrypting key type. The program also places `EXPORTER CONVERTED FROM CUSP12.CKDS` in the installation data field.

Example 4: This example shows an override record specifying that the conversion program convert a remote key to an importer key-encrypting key for any PCF CKDS entry with a certain key label. The override record also provides the conversion program with installation data.

```
RECKDS12 REMOTE   IMPORTER            IMPORTER CONVERTED FROM CUSP12.CKDS
```

The conversion program automatically converts remote keys to importer key-encrypting keys. You can use this override record if you supplied an override record to convert all the remote key types to input key-encrypting keys. Use this override record to explicitly convert key entries labeled RECKDS12 from remote key types to importer key-encrypting key types.

With the example override record, when the conversion program converts any PCF entry labeled RECKDS12 with a key type of remote, it converts the key from a remote key type to an importer key-encrypting key type. The program also places `IMPORTER CONVERTED FROM CUSP12.CKDS` in the installation data field.

Example 5: This example shows a global override record specifying that the conversion program convert a local key to an output PIN-encrypting key for any PCF CKDS entry with a key type of local. The override record also provides the conversion program with installation data.

```
LOCAL  OPINENC           OPINENC FROM CUSP.PIN12.CKDS
```

When the conversion program converts any PCF entry with a key type of local, the program converts the key from a local key type to an output PIN-encrypting key type. The program also places `OPINENC FROM CUSP.PIN12.CKDS` in the installation data field. After you specify this global override record, you can use explicit override records to place different installation data in the ICSF CKDS entries.

Example 6: This example shows a global override record specifying that the conversion program convert a remote key to an input PIN-encrypting key for any PCF CKDS entry with a key type of remote. The override record also provides the conversion program with installation data.

```
REMOTE  IPINENC           IPINENC FROM CUSP.PIN12.CKDS
```

When the conversion program converts any CUSP/PCF entry with a key type of remote, it converts the key from a remote key type to an input PIN-encrypting key type. The program also places `IPINENC FROM CUSP.PIN12.CKDS` in the installation data field for the entry in the ICSF CKDS. After you specify this global override record, you can use explicit override records to place different installation data information in the ICSF CKDS entries.

Running the Conversion Program

You can run the conversion program only after you initialize the master key and CKDS for ICSF. The CKDS you specify at ICSF startup must be initialized to

contain NOCV-enablement keys. For information about defining keys on ICSF, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

If the PCF master key and the ICSF master key are not the same, you must define the PCF master key in the input ICSF CKDS. Define the PCF master key as an importer key-encrypting key in the input ICSF CKDS. You define the key by entering the key through the key entry hardware, or by importing the key using the ICSF key generator utility program. For information about direct key entry through the key entry hardware and the key generator utility program, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

Note: Be careful defining the PCF master key in the input ICSF CKDS, because there is no programmed way to determine its validity.

You run the conversion program by submitting a batch job. On the EXEC statement, specify PGM=CSFCONV. If the PCF master key and ICSF master key are not the same in the PARM= field on the EXEC statement, specify the label of the PCF master key entry in the input ICSF CKDS. If you do not specify the parameter, the conversion program assumes that the PCF master key and ICSF master key are the same.

This example is a JCL that runs the conversion program:

```
//CKDSCONV EXEC PGM=CSFCONV,PARM='CUSPMKEY'
//CSFVSRC  DD   DSN=PROD.CUSP.CKDS,DISP=SHR
//CSFVINP  DD   DSN=TEST.CSF.CKDS,DISP=SHR
//CSFVOVR  DD   DSN=OVERRIDE.DATA,DISP=OLD
//CSFVNEW  DD   DSN=MERGED.CSF.CKDS,DISP=OLD
//CSFVRPT  DD   SYSOUT=A
//
```

In the example, CUSPMKEY is the label of the entry in the input ICSF CKDS for the PCF master key in importer key-encrypting key form. All the data sets necessary to run the conversion program are specified using DD statements.

The conversion program uses these data sets:

CSFVSRC
: The PCF CKDS containing entries that you want to convert into ICSF format and place in the output ICSF CKDS. This is the source CKDS for the conversion. For a description of the PCF CKDS record format, see *OS/VS1 and OS/VS2 MVS Programmed Cryptographic Facility*.

CSFVINP
: A disk copy of the input ICSF CKDS. The input CKDS should already contain the header record and the ICSF system keys and can contain other ICSF key entries. If the CKDS does not contain NOCV-enablement keys, the output ICSF CKDS will not contain NOCV-enablement keys. For more information about NOCV-enablement keys, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

  Note: The input ICSF CKDS does not have to be the CKDS you specify when you start ICSF.

CSFVOVR
: The override file with information specifying how you want the conversion program to process PCF key entries. If no override data is required, this data set is optional. However, you must code a dummy DD statement in the JCL.

This JCL is an example of a dummy DD statement for an override file:

```
//CSFVOVR DD DUMMY,DCB=(RECFM=FB,LRECL=90,BLKSIZE=3600)
```

See “Using the conversion program override file” on page 179 for a description of when and how to use the override file.

CSFVNEW
An empty disk copy of an ICSF CKDS. This is the ICSF CKDS into which the conversion program places key entries. The conversion program places key entries from the input ICSF CKDS and the PCF CKDS into the output ICSF CKDS. The data set must be defined and empty before you run the conversion program.

CSFVRPT
The activity report that the conversion program creates. The report describes any override records and gives a summary of CKDS entries that were affected by the conversion program.

Attention: If a conversion program run ends prematurely, the results of the job are unpredictable. You should not read a CKDS involved in the conversion into storage for use. For a description of the conversion program return codes, see the explanation of message CSFV0026 in *z/OS Cryptographic Services ICSF Messages*.

When you run the conversion program, the program produces information about the conversion in an activity report. The activity report lists each override entry, the action each override entry applies to the input PCF CKDS, and any error messages. The activity report also lists the data sets that were used in the conversion and a summary of processing. The summary of processing contains totals that apply to CKDS entries in the conversion program job.

Example of a Conversion Initial Activity Report

Figure 11 on page 187 is an example of an activity report with five explicit override records and no global override records.

```
CRYPTOGRAPHIC CONVERSION ACTIVITY REPORT                DATE: 2001/06/01 (YYYY/MM/DD) TIME: 10:13:09 PAGE: 1
 OVERRIDE--> CRLABEL3 LOCAL    OPINENC             Used in transfers to Main Office.
 >>>CSFV0192 TYPE FOR KEY ENTRY CRLABEL3 LOCAL CONVERTED TO OPINENC.
 >>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY CRLABEL3 OPINENC SET TO Used in transfers to Main Office

 OVERRIDE--> CRLABEL3 REMOTE   IPINENC             Used in receiving from the Main Office
 >>>CSFV0192 TYPE FOR KEY ENTRY CRLABEL3 REMOTE CONVERTED TO IPINENC.
 >>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY CRLABEL3 IPINENC SET TO Used in receiving from the Main Office.

 OVERRIDE--> KGLABEL1 LOCAL    OPINENC             Used for sending encrypted PINs
 >>>CSFV0292 NO KEY ENTRY FOUND FOR KGLABEL1 LOCAL.


 OVERRIDE--> LOLABEL2                             Valid for January 2001
 >>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY LOLABEL2 EXPORTER SET TO Valid for January 2001.

 OVERRIDE--> ZZZZ1    LOCAL                      Y Eliminate Key from output CKDS
 >>>CSFV0382 ADD/CHANGE SPECIFICATIONS IGNORED ON OVERRIDE ENTRY. BYPASS_FLAG VALUE IS "Y".
 >>>CSFV0292 NO KEY ENTRY FOUND FOR ZZZZ1 LOCAL.


 >>>CSFV0012 CONVERSION PROCESSING COMPLETED. RETURN CODE = 4.
CRYPTOGRAPHIC CONVERSION ACTIVITY REPORT                DATE: 2001/06/01 (YYYY/MM/DD) TIME: 10:13:09 PAGE: 2

    CKDS DDNAME     Data Set Name
    ------------    --------------
    CSFVSRC         PROD.CUSP.CKDS
    CSFVINP         TEST.CSF.CKDS
    CSFVNEW         MERGED.CSF.CKDS

    PROCESSING SUMMARY

            Source CKDS Entries               Converted Entries                    ICSF Entries
            --------------------------------  ---------------------------------    ---------------------------------
               LOCAL                     4    * Candidates                  16     + Changed Input Entries          2
               REMOTE                    4      Bypassed by Overrides       ( 0)     Unchanged Input Entries       13
               CROSS                     4                                          --------------------------------
               ------------------------------     --------------------------------   TOTAL ICSF Input Entries      15
             * TOTAL Source Entries      12       TOTAL Converted Entries     16    + Entries Added from Source     14
                                                                                      Entries Bypassed by Exit     ( 0)
                                                                                      --------------------------------
                                                                                      TOTAL Output ICSF Entries     29


  * One Source CKDS CROSS entry converts to two Candidates.
  + Total Converted Entries = Changed Input Entries + Entries Added from Source.
```

Figure 11. Example of a Conversion Initial Activity Report

In the report, the first override record specifies that when the conversion program converts a PCF entry labeled CRLABEL3 with a key type of local, the program should convert the entry into an output PIN-encrypting key. The conversion program also places the information `Used in transfers to Main Office` in the installation data field of the output ICSF CKDS entry.

The second override record specifies that when the conversion program converts a PCF entry labeled CRLABEL3 with a key type of remote, the program should convert the key into an input PIN-encrypting key. The conversion program places the information `Used in receiving from the Main Office` in the installation data field of the output ICSF CKDS entry.

The label specified by the third override record does not exist in the PCF CKDS. Therefore, the conversion program ignores this override record.

The fourth override record specifies that when the conversion program converts a PCF entry labelled LOLABEL2, the program should place the information `Valid for January 2001` in the installation data field of the output ICSF CKDS record.

The label specified by the fifth override record does not exist on the PCF CKDS that the conversion program is converting. Therefore, the conversion program ignores this override record.

The message that the conversion processing has been completed is followed by a return code. Return codes are listed under message CSFV0026 in *z/OS Cryptographic Services ICSF Messages*.

After describing the five override records, the conversion report lists the data sets the conversion program used in the conversion. PROD.CUSP.CKDS is the PCF CKDS that the program converted. TEST.CSF.CKDS is the input ICSF CKDS containing the ICSF entries input during the conversion. MERGED.CSF.CKDS is the output ICSF CKDS where the conversion program placed the converted entries.

Then the activity report lists totals pertaining to the conversion. The PCF CKDS has a total of 12 entries: four with a key type of local, four with a key type of remote, and four with a key type of cross. Because the conversion of each cross key entry results in two ICSF entries, the total ICSF entries that are candidates for conversion from the PCF is 16. None of these candidates was bypassed because of an override record, so 16 PCF entries were converted.

There were 15 entries in the input ICSF CKDS, and two of these entries were updated because they had identical key labels in the PCF CKDS. Fourteen new output ICSF CKDS entries were added from the PCF CKDS. The total number of entries in the output ICSF CKDS is 29. This includes the 15 entries in the input ICSF CKDS and the 14 entries added from the PCF CKDSN. No entries were bypassed because of the conversion program exit.

Example of a Conversion Update Activity Report

Figure 12 on page 189 is an example of an activity report with a global override record that has the conversion program bypass all the entries in the PCF CKDS. Then two override records are used to convert specific entries.

```
CRYPTOGRAPHIC CONVERSION ACTIVITY REPORT                DATE: 2001/06/01 (YYYY/MM/DD) TIME: 10:13:09 PAGE: 1
OVERRIDE-->                                     Y
 >>>CSFV0172 ALL  ENTRIES BYPASSED.

 OVERRIDE--> CRLABEL3 LOCAL    OPINENC             Used in transfers to Main Office
 >>>CSFV0222 KEY ENTRY CRLABEL3 LOCAL NOT BYPASSED.
 >>>CSFV0192 TYPE FOR KEY ENTRY CRLABEL3 LOCAL CONVERTED TO OPINENC.
 >>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY CRLABEL3 OPINENC SET TO Used in transfers to Main Office.

OVERRIDE--> LOLABEL2                            Valid for January 2001
 >>>CSFV0222 KEY ENTRY LOLABEL2 LOCAL NOT BYPASSED.
 >>>CSFV0232 INSTALLATION DATA FOR KEY ENTRY LOLABEL2 EXPORTER SET TO Valid for January 2001.


 >>>CSFV0012 CONVERSION PROCESSING COMPLETED.  RETURN CODE = 0.

CRYPTOGRAPHIC CONVERSION ACTIVITY REPORT                DATE: 2001/06/01 (YYYY/MM/DD) TIME: 10:13:09 PAGE: 2

    CKDS DDNAME     Data Set Name
    ------------    --------------
    CSFVSRC         PROD.PCF.CKDS
    CSFVINP         INTEST.CSF.CKDS
    CSFVNEW         NEWTEST.CSF.CKDS

    PROCESSING SUMMARY

             Source CKDS Entries                Converted Entries                    ICSF Entries
             -------------------------------    -------------------------------      -------------------------------
                LOCAL                      4     * Candidates                  16      + Changed Input Entries       1
                REMOTE                     4       Bypassed by Overrides     (14)        Unchanged Input Entries    27
                CROSS                      4                                            -------------------------------
                ---------------------------        -------------------------------      TOTAL ICSF Input Entries   28
              * TOTAL Source Entries      12        TOTAL Converted Entries     2      + Entries Added from Source     1
                                                                                        Entries Bypassed by Exit    ( 0)
                                                                                        -------------------------------
                                                                                        TOTAL Output ICSF Entries  29


  * One Source CKDS CROSS entry converts to two Candidates.
  + Total Converted Entries = Changed Input Entries + Entries Added from Source.
```

Figure 12. Example of a Conversion Update Activity Report

The first override record specifies that the conversion program bypass all the entries in the PCF CKDS. The second override record specifies that the conversion program convert a PCF entry labeled CRLABEL3 with a key type of local into an output PIN-encrypting key. This second override record also instructs the conversion program to place the phrase `Used in transfers to Main Office` in the installation data field of the output ICSF CKDS entry. The third override record specifies that the conversion program convert a PCF entry labeled LOLABEL2 and place `Valid for January 2001` in the installation data field of the output ICSF CKDS entry.

After describing the three override records, the conversion report lists the data sets the conversion program used in the conversion. PROD.PCF.CKDS is the PCF CKDS that the program converted. INTEST.CSF.CKDS is the input ICSF CKDS that contains the ICSF entries input containing the ICSF entries input during the conversion. NEWTEST.CSF.CKDS is the output ICSF CKDS where the conversion program placed the converted entries.

Then the activity report lists totals pertaining to the conversion. The PCF CKDS has a total of 12 entries: four with a key type of local, four with a key type of remote, and four with a key type of cross. Because the conversion of each cross key entry results in two ICSF entries, the total ICSF records that are candidates for conversion from PCF is 16. Fourteen of those 16 entries were bypassed because of the global override record.

There were 28 entries in the input ICSF CKDS, and one of these entries was updated because it had an identical key label in the PCF CKDS. The total number

of entries in the output ICSF CKDS is 29. This includes the 28 entries in the input ICSF CKDS plus the one added from the PCF CKDS. No entries were bypassed because of the conversion program exit.

Chapter 9. Compatibility and Coexistence of 4753-HSP and ICSF

The Transaction Security System products provide a range of cryptographic facilities. These facilities can be implemented throughout an organization using a compatible set of services at both workstation and host locations. One component of the Transaction Security System is the channel-attached IBM 4753 Network Security Processor (NSP) and its supporting software. The 4753 NSP can be installed at the IBM System/370, IBM System/OS/390 MVS or OS/390 host locations. The IBM Network Security Processor Support Program (referred to as 4753-HSP) provides host software support for the 4753 NSP. The Network Security Processor Control Program runs in the IBM 4753 NSP and processes encryption requests that are received from the host. If your installation is currently using 4753-HSP, you can either add ICSF and the Cryptographic Coprocessor Feature to your OS/390 host (where it can coexist with 4753-HSP) or migrate to ICSF.

Because both 4753-HSP and ICSF support the CCA, applications developed to run with 4753-HSP may run with ICSF without recompiling if they contain common verbs.

This topic gives a brief overview of 4753-HSP and ICSF coexistence considerations. See “Migrating from 4753-HSP” on page 72 for migration considerations.

Running 4753-HSP and ICSF on the same z/OS system

Although the 4753-HSP and ICSF can coexist in the same z/OS environment on a logical partition of a S/390 or z/OS complex, some restrictions apply.

Both systems can run simultaneously in noncompatibility mode. However, because both systems support the CCA API, they use the same verbs to call cryptographic services. For this reason, you must link your applications with the appropriate library routines to ensure that they are routed to the correct system. Use 4753-HSP stubs to link applications that are intended for 4753-HSP. Use ICSF stubs in SYS1.SCSFMOD0 to link applications that are intended for ICSF.

Both ICSF and 4753-HSP are capable of running CUSP/PCF compatibility mode, but only one system can provide this service at a time. Use of compatibility mode is effectively serialized by ownership of the CVTCCVT field.

The two systems use the external key token for key exchange. Internal key tokens are not interchangeable between 4753-HSP and ICSF, and each system uses different control vectors internally for data keys.

Generally, 4753-HSP provides additional function for a given service call. Be sure to use the common subset of services when an application operates with both of the systems. (Concurrent use of 4753-HSP and ICSF is beyond the scope of this information.)

z/OS ICSF supports a PKA implementation that differs from the Transaction Security System PKA implementation. The Transaction Security System supports both PKA92 and PKA96 versions. Applications that are written to one PKA version will not run on the other PKA version. Because ICSF does not support PKA92, 4753-HSP techniques that use RSA keys that have been implemented in PKA92 for

DES key distribution are incompatible with ICSF applications. For PKA96, APIs are the same for services that ICSF and the 4753-HSP have in common. With the addition of RSA key support in the PKA Generate function, it becomes easier for the PKA96 to move to ICSF. The main difference is that the 4753-HSP does not support the Digital Signature Standard (DSS). RSA digital signatures that use ISO9796 formatting can be exchanged between the two products.

Appendix A. Diagnosis Reference Information

This appendix contains Diagnosis, Modification, or Tuning Information.

This appendix contains descriptions of the cryptographic key data set (CKDS), the public key data set (PKDS), PKA key tokens, the Cryptographic Communication Vector Table (CCVT), and Cryptographic Communication Vector Table Extension (CCVE) data areas.

For more information about key tokens, refer to *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Cryptographic Key Data Set (CKDS) Formats

There are two formats of the CKDS: a fixed length record (supported by all releases of ICSF) and a new, variable length record (supported by HCR7780 and later releases). The variable length record format is only required if HMAC keys are to be stored in the CKDS. The variable length record format can be used to store all existing symmetric keys and the new HMAC keys.

Fixed-Length Cryptographic Key Data Set (CKDS) Record Format

The CKDS includes a header record, data set record, and an internal key token record. Tables in these topics show the format of each of these records.

Format of the Fixed-Length CKDS Header Record

Table 25 presents the format of the CKDS header record.

Table 25. Cryptographic Key Data Set Header Record Format

Offset (Dec)	Number of Bytes	Field Name	Description
0	72	*Constant*	The field is set to binary zeros and is not used for the header record.
72	8	*Creation date*	The date the CKDS was initialized in the format *yyyymmdd*.
80	8	*Creation time*	The initial time the CKDS was created in the format *hhmmssth*.
88	8	*Last update date*	The most recent date the CKDS was updated, in the format *yyyymmdd*.
96	8	*Last update time*	The most recent time the CKDS was updated, in the format *hhmmssth*.
104	2	*Sequence number*	Initially zero in binary. Incremented each time the data set is processed.

Table 25. Cryptographic Key Data Set Header Record Format (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
106	2	*CKDS header flag bytes*	Flag bytes. **Bit** — **Meaning When Set On** **0** — The DES master key verification pattern is valid. **1** — The DES master key authentication pattern is valid. **2** — The AES master key verification pattern is valid. **3–7** — Reserved. **8** — Record level authentication is disabled. **9–15** — Reserved. **Note:** After the bits are set on, the given values remain constant in ICSF.
108	8	*DES master key verification pattern*	The system DES master key verification pattern.
116	8	*DES master key authentication pattern*	The system DES master key authentication pattern.
124	8	*AES master key verification pattern.*	The AES master key verification pattern.
132	64	*Reserved*	The field is set to binary zeros.
196	52	*Installation data*	Installation data associated with the CKDS record, as supplied by an installation exit.
248	4	*Authentication code*	The code generated by the authentication process that ensures that the CKDS record has not been modified since the last update. The authentication code is placed in the CKDS header record when the CKDS is initialized. ICSF verifies the CKDS header record authentication code whenever a CKDS is reenciphered, refreshed, or converted from PCF to ICSF format.This field is not used when the record level authentication flag is set in the CKDS header flag bytes field of the CKDS header record.

Format of the Fixed-Length CKDS Record

Table 26 presents the format of each data set record.

Table 26. Cryptographic Key Data Set Record Format

Offset (Dec)	Number of Bytes	Field Name	Description
0	64	*Key label*	The key label specified by the KGUP control statement or Clear Key Input panel when the record was created. When using KGUP and the callable services, you can specify the label to identify the record. The key label is the first field of the key index.

Table 26. Cryptographic Key Data Set Record Format (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
64	8	*Key type*	The type of key the record contains. The master key variant for the key type enciphers the key. A KGUP control statement or Clear Key Input panel specifies the key type when the record is created. The key type is the second field of the key index.
72	8	*Creation date*	The initial date the CKDS record was created in the format *yyyymmdd*.
80	8	*Creation time*	The initial time the CKDS record was created in the format *hhmmssth*.
88	8	*Last update date*	The most recent date the CKDS record was updated in the format *yyyymmdd*.
96	8	*Last update time*	The most recent time the CKDS record was updated in the format *hhmmssth*.
104	64	*Key token*	The internal key token. A key token contains the key value. Refer to "DES Internal Key Token" on page 218 for the format of the internal key token.
168	2	*CKDS flag bytes*	Flag bytes. **Bit** — **Meaning When Set On** **0** — The key within the key token field (offset 104) is a partial key. You can enter key parts through the key entry hardware. A partial key is a key whose final key part has not been entered yet. **1** — Cryptographic key token (CKT) delete. **2** — CKDS label must be unique. **3–7** — Reserved. **Note:** When bit 0 is off, the key within the key token field (offset 104) is an entire key.
170	26	*Reserved*	Reserved.
196	52	*Installation data*	Installation data associated with the CKDS record as supplied by an installation exit.
248	4	*Authentication code*	The code generated by the authentication process that ensures the CKDS record has not been modified since the last update. The authentication code is placed in the CKDS record when the record is created. When you refresh, reencipher, or convert a CKDS, ICSF verifies each CKDS record as ICSF performs the action. This field is not used when the record level authentication flag is set in the CKDS header flag bytes field of the CKDS header record.

Variable-Length Cryptographic Key Data Set (CKDS) Record Format

The CKDS record includes the CKDS header and the key record. These tables show the format of each of these records.

Format of the Variable-Length Header Record

The following table presents the format of the variable-length CKDS header record

Table 27. Cryptographic Key Data Set Header Record Format

Offset (Dec)	Number of Bytes	Field Name	Description
0	72	*Constant*	VSAM key of the CKDS header.
72	8	*Creation date*	The date the CKDS was initialized in the format *yyyymmdd*.
80	8	*Creation time*	The initial time the CKDS was created in the format *hhmmssth*.
88	8	*Last update date*	The most recent date the CKDS was updated, in the format *yyyymmdd*.
96	8	*Last update time*	The most recent time the CKDS was updated, in the format *hhmmssth*.
104	2	*Sequence number*	Initially zero in binary. Incremented each time the data set is processed.
106	2	*header flag bytes*	Flag bytes. **Bit** — **Meaning When Set On** **0** — The DES master key verification pattern is valid. **1** — The DES master key authentication pattern is valid. **2** — The AES master key verification pattern is valid. **3–8** — Reserved. **9** — The record format is variable — always 1 **10-15** — Reserved. **Note:** After the bits are set on, the given values remain constant in ICSF.
108	8	*DES master key verification pattern*	The system DES master key verification pattern.
116	8	*DES master key authentication pattern*	The system DES master key authentication pattern.
124	8	*AES master key verification pattern.*	The system AES master key verification pattern.
132	4	Record length	Length of the record in bytes.
136	60	*Reserved*	
196	52	*Installation data*	
248	4	*Authentication code*	CKDS header authentication code.

Format of the Variable-Length CKDS Record

The following table presents the format of each variable-length data set record.

Table 28. Variable-Length Cryptographic Key Data Set Record Format

Offset (Dec)	Number of Bytes	Field Name	Description
0	64	*Key label*	The label or name of this CKDS record. The key label is the first field of the key index.
64	8	*Key type*	The type of key the record contains. The key type is the second field of the key index.
72	8	*Creation date*	The initial date the CKDS record was created in the format *yyyymmdd*.
80	8	*Creation time*	The initial time the CKDS record was created in the format *hhmmssth*.
88	8	*Last update date*	The most recent date the CKDS record was updated in the format *yyyymmdd*.
96	8	*Last update time*	The most recent time the CKDS record was updated in the format *hhmmssth*.
104	4	*Record length*	Length of the entire record including the key token.
108	60		Reserved.
168	2	*CKDS flag bytes*	Flag bytes. **Bit** — **Meaning When Set On** **0** — The key within the key token field is a partial key. **1** — Reserved. **2** — CKDS label must be unique. **3** — The record format is variable — always 1 **4–7** — Reserved. **Note:** When bit 0 is off, the key within the key token field (offset 104) is an entire key.
170	26		Reserved.
196	52	*Installation data*	
248	20	*Authentication code*	The record authentication code.
268	variable	*Key token*	The key token.

Public Key Data Set (PKDS) Format

The PKDS record includes the PKDS header and the PKA key token. These tables show the format of each of these records.

Format of the PKDS Header Record

Table 29. Public Key Data Set Header Record Format

Offset (Dec)	Number of Bytes	Field Name	Description
0	64	*PKHVKEY*	VSAM key of the PKDS header.
64	8		Reserved.

Table 29. Public Key Data Set Header Record Format (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
72	8	*PKHCRDTE*	The date the PKDS was created in the format *yyyymmdd*.
80	8	*PKHCRTIM*	The initial time the PKDS was created in the format *hhmmssth*.
88	8	*PKHUPDTE*	The most recent date the PKDS header was updated, in the format *yyyymmdd*.
96	8	*PKHUPTIM*	The most recent time the PKDS header was updated, in the format *hhmmssth*.
104	4	*PKHRLEN*	Length of the PKDS header entry.
108	16	*PKHKMKHP*	The hash pattern of the KMMK.
124	16	*PKHSMKHP*	The hash pattern of the SMK.
140	8	*PKHEMKVP*	The verification pattern of the ECC MK.
148	12		Reserved.
160	20	*PKHAUTH*	PKDS header authentication code.

Format of the PKDS Record

Table 30. Public Key Data Set Record Format

Offset (Dec)	Number of Bytes	Field Name	Description
0	64	*PKDLABEL*	Label or name of this PKDS entry.
64	8		Reserved.
72	8	*PKDCRDTE*	The date this PKDS record was created in the format *yyyymmdd*.
80	8	*PKDCRTIM*	The initial time this PKDS record was created in the format *hhmmssth*.
88	8	*PKDUPDTE*	The most recent date this PKDS record was updated, in the format *yyyymmdd*.
96	8	*PKDUPTIM*	The most recent time this PKDS record was updated, in the format *hhmmssth*.
104	4	*PKDRLEN*	Length of the entire PKDS record entry.
108	52	*PKDUDATA*	User data.
160	20	*PKDAUTH*	The entry authentication code.
180	1868	*PKDTOKEN*	The public or private key token.

Token data set (TKDS) format

A z/OS PKCS #11 token represents a virtual cryptographic device, and can contain multiple objects. The token data set (TKDS) contains definitions of z/OS PKCS #11 tokens and token objects.

The token data set includes a header record and records for each of the individual z/OS PKCS #11 tokens and token objects. Each object associated with a particular z/OS PKCS #11 token has the token's name in its handle. The records are variable length records, and contain a length field specifying the total length of the record.

Format of the header record of the token data set

There is one header record for the token data set.

Table 31. Format of the header record of the token data set

Offset (decimal)	Length of field (bytes)	Description
0	72	VSAM key of the TKDS header **Bytes 0-39:** Binary zeros **Bytes 40-43:** EBCDIC “THDR” **Bytes 44-71:** Binary zeros
72	8	Reserved for IBM's use
80	8	The date that the TKDS was created, in the format *yyyymmdd*
88	8	The time that the TKDS was created, in the format *hhmmmssth*
96	8	The most recent date that the TKDS header was updated, in the format *yyyymmdd*
104	8	The most recent time that the TKDS header was updated, in the format *hhmmmssth*
112	4	Length of the TKDS header record
116	40	Reserved for IBM's use

Format of the token and object records

Each z/OS PKCS #11 token record and token object record begins with the same 188 bytes of data. The remainder of the record is specific to the token or object.

Common section of the token and object records

Every record in the token data set, with the exception of the header record, begins with these 188 bytes of data.

Table 32. Format of the common section of the token and object records

Offset (decimal)	Length of field (bytes)	Description
0	72	Handle of token or object **Bytes 0-31:** Token name **Bytes 32-39:** Sequence number **Byte 40:** Character “T” for token object **Bytes 41-43** Blank characters **Bytes 44-71:** Binary zeros
72	8	Reserved for IBM's use
80	8	The date that this record was created, in the format *yyyymmdd*
88	8	The time that this record was created, in the format *hhmmssth*
96	8	The most recent date that this record was updated, in the format *yyyymmdd*
104	8	The most recent time that this record was updated, in the format *hhmmssth*
112	4	Length of the entire TKDS record entry
116	20	Reserved for IBM's use

Table 32. Format of the common section of the token and object records (continued)

Offset (decimal)	Length of field (bytes)	Description
136	52	User data
188	variable	The TKDS token or object (see mappings)

Format of the token-specific section of the token record

Each z/OS PKCS #11 token record begins with the 188 bytes. The remainder of the record contains the contents of the token. The mapping of the record shows the data beginning at offset 0, which is its offset into the token-specific portion of the record; however, that portion of the record is at an offset of 188 into the entire record.

Table 33. Format of the unique section of the token record

Offset (decimal) 188 +	Length of field (bytes)	Description
0	4	Eye catcher for token: “TOKN”
4	2	Version number of structure: EBCDIC '00'
6	2	Length of structure in bytes
8	4	Reserved for IBM's use. Must be zeros.
12	8	Last assigned sequence number
20	32	Manufacturer identification
52	16	Model
68	16	Serial number
84	8	Date of the most recent update to this token, expressed as Coordinated Universal Time (UTC) in the format *yyyymmdd*. This includes any update to token information or to a token object.
92	8	Time of the most recent update to this token, expressed as Coordinated Universal Time (UTC) in the format *hhmmssth*. This includes any update to token information or to a token object.
100	44	Reserved for IBM's use
144		End of token

Format of the object-specific sections of the token object records

The following classes of objects can be associated with a z/OS PKCS #11 token:

- Certificate
- Public key
- Private key
- Secret key
- Data objects
- Domain parameters

The token object record for each begins with the common section described “Common section of the token and object records” on page 199, followed by a section specific to the class of object. Each of the object-specific sections begins with a 12-byte header record, followed by a variable-length section. Each 12-byte header contains a 4-byte flag field that has the same mapping for all classes of objects.

Table 34. Format of the token object flags. This 4-byte flag field occurs in the object header section of each token object record.

Offset (decimal)	Field name	Description
Flag byte 1		
Bit 0	OBJ_IS_TOKOBJ	When on, the object is a token object. When off, the object is a session object.
Bit 1	OBJ_IS_PRVOBJ	When on, the object is a private object. When off, the object is a public object.
Bit 2	OBJ_IS_MODOBJ	When on, the object is modifiable.
Bit 3	KEY_DERIVE	When on, the key supports key derivation.
Bit 4	KEY_LOCAL	When on, the key was generated locally.
Bit 5	KEY_ENCRYPT	When on, the key supports encryption.
Bit 6	KEY_DECRYPT	When on, the key supports decryption.
Bit 7	KEY_VERIFYA	When on, the key supports verification where the signature is an appendix to the data.
Flag byte 2		
Bit 0	KEY_VERIFYR	When on, the key supports verification where the data is recovered from the signature
Bit 1	KEY_SIGA	When on, the key supports signatures where the signature is an appendix to the data.
Bit 2	KEY_SIGR	When on, the key supports signatures where the data is recovered from the signature.
Bit 3	KEY_WRAP	When on, the key supports wrapping.
Bit 4	KEY_UNWRAP	When on, the key supports unwrapping.
Bit 5	KEY_EXTRACT	When on, the key is extractable.
Bit 6	KEY_IS_SENSITIVE	When on, the key is sensitive.
Bit 7	KEY_IS_ALWAYS_SENSITIVE	When on, the SENSITIVE attribute (KEY_IS_SENSITIVE) is always true.
Flag byte 3		
Bit 0	KEY_NEVER_EXTRACT	When on, the EXTRACTABLE attribute (KEY_EXTRACT) is never true. When off, the EXTRACTABLE attribute (KEY_EXTRACT) can be true.
Bit 1	OBJ_IS_TRUSTED	When on, the certificate can be trusted for the application for which it was created.
Bit 2	CERT_IS_DEFAULT	When on, this is the default certificate.
Bit 3	FIPS140	When on, key is only to be used in a FIPS-compliant manner.

Table 34. Format of the token object flags (continued). This 4-byte flag field occurs in the object header section of each token object record.

Offset (decimal)	Field name	Description
Bits 4-7		Reserved for IBM's use
Flag byte 4		
Bits 0-7		Reserved for IBM's use

Table 35. Format of the token certificate object

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for certificate object: “CERT”
4	2	Version: EBCDIC '00'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 34 on page 201)
Object type-specific section		
12	4	TYPE attribute: X'00000000': CKC_X_509
16	4	Certificate category **0** Undefined **1** Token user **2** Certificate authority **3** Other entity
20	8	Reserved for IBM's use
28	32	Reserved for IBM's use
60	2	Length of SUBJECT attribute in bytes (*aa*)
62	2	Length of ID attribute in bytes (*bb*)
64	2	Length of ISSUER attribute in bytes (*cc*)
66	2	Length of SERIAL_NUMBER attribute in bytes (*dd*)
68	2	Length of VALUE attribute in bytes (*ee*)
70	2	Length of LABEL attribute in bytes (*ff*)
72	2	Length of APPLICATION attribute in bytes (*gg*)
74	22	Reserved for IBM's use
96	4	Offset of SUBJECT attribute in bytes
100	4	Offset of ID attribute in bytes
104	4	Offset of ISSUER attribute in bytes
108	4	Offset of SERIAL_NUMBER attribute in bytes
112	4	Offset of VALUE attribute in bytes
116	4	Offset of LABEL attribute in bytes
120	4	Offset of APPLICATION attribute in bytes
124	44	Reserved for IBM's use
168	*aa + bb + cc + dd + ee + ff + gg*	Certificate attributes (variable length)

Table 35. Format of the token certificate object (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
168 + *aa* + *bb* + *cc* + *dd* + *ee* + *ff* + *gg*		End of certificate object

Table 36. Format of the token public key object (Version 0)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for public key object: “PUBK”
4	2	Version: EBCDIC '00'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 34 on page 201)
Object type-specific section		
12	4	TYPE attribute: CKK_RSA
16	8	Start date for the key, in the format *yyyymmdd*
24	8	End date for the key, in the format *yyyymmdd*
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
72	4	Length in bits of modulus n
76	256	Modulus n
332	256	Reserved
588	256	Public exponent e
844	256	Reserved
1100	2	Length of SUBJECT attribute in bytes (*aa*)
1102	2	Length of ID attribute in bytes (*bb*)
1104	2	Length of LABEL attribute in bytes (*cc*)
1106	2	Length of APPLICATION attribute in bytes (*dd*)
1108	20	Reserved
1128	4	Offset of SUBJECT attribute in bytes
1132	4	Offset of ID attribute in bytes
1136	4	Offset of LABEL attribute in bytes
1140	4	Offset of APPLICATION attribute in bytes
1144	40	Reserved
1184	*aa+bb+cc+dd*	Public key attributes (variable length)
1184+*aa+bb+cc+dd*		End of public key object

Table 37. Format of the token public key object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for public key object: “PUBK”
4	2	Version: EBCDIC '01'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 34 on page 201)
Object type-specific section		
12	4	TYPE attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key, in the format *yyyymmdd*
24	8	End date for the key, in the format *yyyymmdd*
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus *n*
76	512	Modulus *n*
588	512	Public exponent *e*
Algorithm-specific section (DSA)		
72	4	Length in bits of prime *p*
76	128	Reserved
204	128	Prime *p*
332	128	Reserved
460	128	Base *g*
588	128	Reserved
716	128	Value *y*
844	20	Reserved
864	20	Subprime *q*
884	216	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime *p*
76	256	Prime *p*
332	256	Base *g*
588	256	Value *y*
844	256	Reserved
Algorithm-specific section (EC)		

Table 37. Format of the token public key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	128	Reserved
204	136	EC point Q (DER encoded)
340	760	Reserved
Variable length attribute section		
1100	2	Length of SUBJECT attribute in bytes (*aa*)
1102	2	Length of ID attribute in bytes (*bb*)
1104	2	Length of LABEL attribute in bytes (*cc*)
1106	2	Length of APPLICATION attribute in bytes (*dd*)
1108	20	Reserved
1128	4	Offset of SUBJECT attribute in bytes
1132	4	Offset of ID attribute in bytes
1136	4	Offset of LABEL attribute in bytes
1140	4	Offset of APPLICATION attribute in bytes
1144	40	Reserved
1184	*aa+bb+cc+dd*	Public key attributes (variable length)
1184+*aa+bb+cc+dd*		End of public key object

Table 38. Format of the token public key object (Version 2)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		

Table 38. Format of the token public key object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
0	4	Eye catcher for public key object: “PUBK”
4	2	Version: EBCDIC '02'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 34 on page 201)
Object type-specific section		
12	4	TYPE attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key, in the format *yyyymmdd*
24	8	End date for the key, in the format *yyyymmdd*
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus *n*
76	512	Modulus *n*
588	512	Public exponent *e*
Algorithm-specific section (DSA)		
72	4	Length in bits of prime *p*
76	256	Prime *p*
332	256	Base *g*
588	256	Value *y*
844	8	Reserved
852	32	Subprime *q*
884	216	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime *p*
76	256	Prime *p*
332	256	Base *g*
588	256	Value *y*
844	256	Reserved
Algorithm-specific section (EC)		

Table 38. Format of the token public key object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	128	Reserved
204	136	EC point Q (DER encoded)
340	760	Reserved
Variable length attribute section		
1100	2	Length of SUBJECT attribute in bytes (*aa*)
1102	2	Length of ID attribute in bytes (*bb*)
1104	2	Length of LABEL attribute in bytes (*cc*)
1106	2	Length of APPLICATION attribute in bytes (*dd*)
1108	20	Reserved
1128	4	Offset of SUBJECT attribute in bytes
1132	4	Offset of ID attribute in bytes
1136	4	Offset of LABEL attribute in bytes
1140	4	Offset of APPLICATION attribute in bytes
1144	40	Reserved
1184	*aa+bb+cc+dd*	Public key attributes (variable length)
1184+*aa+bb+cc+dd*		End of public key object

Table 39. Format of the token private key object (Version 0)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		

Table 39. Format of the token private key object (Version 0) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
0	4	Eye catcher for private key object: “PRIV”
4	2	Version: EBCDIC '00'
6	2	Length of object (in bytes)
8	4	Flags (see Table 34 on page 201)
Object type-specific section		
12	4	Type attribute: CKK_RSA
16	8	Start date for the key (in the format *yyyymmdd*)
24	8	End date for the key (in the format *yyyymmdd*)
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
72	4	Length in bits of modulus n
76	256	Modulus: modulus n
332	256	Reserved
588	256	Public exponent e
844	256	Reserved
1100	32	Reserved
1132	256	Private exponent d
1388	256	Reserved
1644	136	Prime p
1780	128	Reserved
1908	128	Prime q
2036	128	Reserved
2172	136	Private exponent d modulo p-1
2300	128	Reserved
2428	128	Private exponent d modulo q-1
2556	128	Reserved
2684	136	CRT coefficient q-1 mod p
2820	128	Reserved
2948	2	Length of SUBJECT attribute in bytes (*xx*)
2950	2	Length of ID attribute in bytes (*yy*)
2952	2	Length of LABEL attribute in bytes (*zz*)
2954	2	Length of APPLICATION attribute in bytes (*ww*)
2956	20	Reserved
2976	4	Offset of SUBJECT attribute in bytes
2980	4	Offset of ID attribute in bytes
2984	4	Offset of LABEL attribute in bytes
2988	4	Offset of APPLICATION attribute in bytes
2992	40	Reserved

Table 39. Format of the token private key object (Version 0) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
3032	*xx+yy+zz+ww*	Private key attributes (variable length)
3032+*xx+yy+zz+ww*		End of private key object

Table 40. Format of the token private key object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for private key object: “PRIV”
4	2	Version: EBCDIC '01'
6	2	Length of object (in bytes)
8	4	Flags (see Table 34 on page 201)
Object type-specific section		
12	4	Type attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key (in the format *yyyymmdd*)
24	8	End date for the key (in the format *yyyymmdd*)
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus *n*
76	512	Modulus: modulus *n*
588	512	Public exponent *e*
1100	32	Reserved
1132	512	Private exponent *d*
1644	264	Prime *p*
1908	256	Prime *q*
2164	264	Private exponent *d* modulo *p*-1
2428	256	Private exponent *d* modulo *q*-1
2684	264	CRT coefficient *q*-1 mod *p*
Algorithm-specific section (DSA)		
72	4	Length in bits of prime *p*
76	128	Reserved
204	128	Prime *p*
332	128	Reserved
460	128	Base *g*
588	236	Reserved
824	20	Value *x*
844	20	Reserved
864	20	Subprime *q*

Table 40. Format of the token private key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
884	2064	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime *p*
76	256	Prime *p*
332	256	Base *g*
588	236	Reserved
824	20	Value *x*
844	2104	Reserved
Algorithm-specific section (EC)		
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	64	Reserved
140	66	Value *d*
206	2742	Reserved
Variable length attribute section		
2948	2	Length of SUBJECT attribute in bytes (*xx*)
2950	2	Length of ID attribute in bytes (*yy*)
2952	2	Length of LABEL attribute in bytes (*zz*)
2954	2	Length of APPLICATION attribute in bytes (*ww*)
2956	20	Reserved
2976	4	Offset of SUBJECT attribute in bytes
2980	4	Offset of ID attribute in bytes
2984	4	Offset of LABEL attribute in bytes

Table 40. Format of the token private key object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
2988	4	Offset of APPLICATION attribute in bytes
2992	40	Reserved
3032	*xx+yy+zz+ww*	Private key attributes (variable length)
3032+*xx+yy+zz+ww*		End of private key object

Table 41. Format of the token private key object (Version 2)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for private key object: “PRIV”
4	2	Version: EBCDIC '02'
6	2	Length of object (in bytes)
8	4	Flags (see Table 34 on page 201)
Object type-specific section		
12	4	Type attribute: CKK_RSA, CKK_DSA, CKK_EC, or CKK_DH
16	8	Start date for the key (in the format *yyyymmdd*)
24	8	End date for the key (in the format *yyyymmdd*)
32	4	Key generate mechanism: CK_UNAVAILABLE_INFORMATION
36	36	Reserved
Algorithm-specific section (RSA)		
72	4	Length in bits of modulus *n*
76	512	Modulus: modulus *n*
588	512	Public exponent *e*
1100	32	Reserved
1132	512	Private exponent *d*
1644	264	Prime *p*
1908	256	Prime *q*
2164	264	Private exponent *d* modulo *p*-1
2428	256	Private exponent *d* modulo *q*-1
2684	264	CRT coefficient *q*-1 mod *p*
Algorithm-specific section (DSA)		
72	4	Length in bits of prime *p*
76	256	Prime *p*
332	256	Base *g*
588	224	Reserved
812	32	Value *x*
844	8	Reserved
852	32	Subprime *q*

Table 41. Format of the token private key object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
884	2064	Reserved
Algorithm-specific section (DH)		
72	4	Length in bits of prime *p*
76	256	Prime *p*
332	256	Base *g*
588	256	Value *x*
844	4	Length in bits of value *x*
848	2100	Reserved
Algorithm-specific section (EC)		
72	4	EC params curve constant – x'00000001' secp192r1 - { 1 2 840 10045 3 1 1 } x'00000002' secp224r1 - { 1 3 132 0 33 } x'00000003' secp256r1 - { 1 2 840 10045 3 1 7 } x'00000004' secp384r1 - { 1 3 132 0 34 } x'00000005' secp521r1 - { 1 3 132 0 35 } x'00000006' brainpoolP160r1 - { 1 3 36 3 3 2 8 1 1 1 } x'00000007' brainpoolP192r1 - { 1 3 36 3 3 2 8 1 1 3 } x'00000008' brainpoolP224r1 - { 1 3 36 3 3 2 8 1 1 5 } x'00000009' brainpoolP256r1 - { 1 3 36 3 3 2 8 1 1 7 } x'0000000A' brainpoolP320r1 - { 1 3 36 3 3 2 8 1 1 9 } x'0000000B' brainpoolP384r1 - { 1 3 36 3 3 2 8 1 1 11 } x'0000000C' brainpoolP512r1 - { 1 3 36 3 3 2 8 1 1 13 }
76	64	Reserved
140	66	Value *d*
206	2742	Reserved
Variable length attribute section		
2948	2	Length of SUBJECT attribute in bytes (*xx*)
2950	2	Length of ID attribute in bytes (*yy*)
2952	2	Length of LABEL attribute in bytes (*zz*)
2954	2	Length of APPLICATION attribute in bytes (*ww*)
2956	20	Reserved
2976	4	Offset of SUBJECT attribute in bytes
2980	4	Offset of ID attribute in bytes
2984	4	Offset of LABEL attribute in bytes

Table 41. Format of the token private key object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
2988	4	Offset of APPLICATION attribute in bytes
2992	40	Reserved
3032	*xx+yy+zz+ww*	Private key attributes (variable length)
3032+*xx+yy+zz+ww*		End of private key object

Table 42. Format of the token secret key object (Version 0)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for secret key object: “SECK”
4	2	Version: EBCDIC '00'
6	2	Length of the object in bytes
8	4	Flags (see Table 34 on page 201)
Object type-specific section		
12	4	Type of key: CKK_DES, CKK_DES2, CKK_DES3, CKK_AES
16	8	Start date for the key (in the format *yyyymmdd*)
24	8	End date for the key (in the format *yyyymmdd*)
32	4	Key generate mechanism CK_UNAVAILABLE_INFORMATION
36	2	Length of the key in bytes
38	32	Reserved
70	64	VALUE: value of the key
134	538	Reserved
672	4	Usage counter field
676	2	Reserved
678	2	Length of LABEL attribute in bytes (*xx*)
680	2	Length of APPLICATION attribute in bytes (*yy*)
682	2	Length of the ID attribute in bytes (*zz*)
684	20	Reserved
704	4	Offset of LABEL attribute in bytes
708	4	Offset of APPLICATION attribute in bytes
712	4	Offset of the ID attribute in bytes
716	40	Reserved
756	*xx+yy+zz*	Secret key attributes (variable length)
756+*xx+yy+zz*		End of secret key object

Table 43. Format of the token secret key object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for secret key object: “SECK”
4	2	Version: EBCDIC '01'
6	2	Length of the object in bytes
8	4	Flags (see Table 34 on page 201)
Object type-specific section		
12	4	Type of key: CKK_DES, CKK_DES2, CKK_DES3, CKK_BLOWFISH, CKK_RC4, CKK_GENERIC_SECRET, and CKK_AES.
16	8	Start date for the key (in the format *yyyymmdd*)
24	8	End date for the key (in the format *yyyymmdd*)
32	4	Key generate mechanism CK_UNAVAILABLE_INFORMATION
36	2	Length of the key in bytes
38	32	Reserved
70	256	VALUE: value of the key
326	346	Reserved
672	4	Usage counter field
676	2	Reserved
678	2	Length of LABEL attribute in bytes (*xx*)
680	2	Length of APPLICATION attribute in bytes (*yy*)
682	2	Length of the ID attribute in bytes (*zz*)
684	20	Reserved
704	4	Offset of LABEL attribute in bytes
708	4	Offset of APPLICATION attribute in bytes
712	4	Offset of the ID attribute in bytes
716	40	Reserved
756	*xx+yy+zz*	Secret key attributes (variable length)
756+*xx+yy+zz*		End of secret key object

Table 44. Format of the token domain parameters object (Version 1)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for token domain object: “DOMP”
4	2	Version: EBCDIC '01'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 34 on page 201)
Object type-specific section		

Table 44. Format of the token domain parameters object (Version 1) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
12	4	TYPE attribute: CKK_DSA or CKK_DH
16	28	Reserved
Algorithm-specific section (DSA)		
44	4	Length in bits of prime *p*
48	128	Reserved
176	128	Prime *p*
304	128	Reserved
432	128	Base *g*
560	20	Reserved
580	20	Subprime *q*
600	636	Reserved
Algorithm-specific section (DH)		
44	4	Length in bits of prime *p*
48	4	Reserved
52	256	Prime *p*
308	256	Reserved
564	256	Base *g*
820	416	Reserved
Variable length attribute section		
1236	2	Length of LABEL attribute in bytes (*aa*)
1238	2	Length of APPLICATION attribute in bytes (*bb*)
1240	20	Reserved
1260	4	Offset of LABEL attribute in bytes
1264	4	Offset of APPLICATION attribute in bytes
1268	40	Reserved
1308	*aa+bb*	Domain parameters attributes (variable length)
1308*+aa+bb*		End of domain parameters object

Table 45. Format of the token domain parameters object (Version 2)

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for token domain object: “DOMP”
4	2	Version: EBCDIC '02'
6	2	Length of the object (in bytes)
8	4	Flags (see Table 34 on page 201)
Object type-specific section		
12	4	TYPE attribute: CKK_DSA or CKK_DH
16	28	Reserved
Algorithm-specific section (DSA)		

Table 45. Format of the token domain parameters object (Version 2) (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
44	4	Length in bits of prime *p*
48	256	Prime *p*
304	256	Base *g*
560	8	Reserved
568	32	Subprime *q*
600	636	Reserved
Algorithm-specific section (DH)		
44	4	Length in bits of prime *p*
48	4	Reserved
52	256	Prime *p*
308	256	Reserved
564	256	Base *g*
820	416	Reserved
Variable length attribute section		
1236	2	Length of LABEL attribute in bytes (*aa*)
1238	2	Length of APPLICATION attribute in bytes (*bb*)
1240	20	Reserved
1260	4	Offset of LABEL attribute in bytes
1264	4	Offset of APPLICATION attribute in bytes
1268	40	Reserved
1308	*aa+bb*	Domain parameters attributes (variable length)
1308*+aa+bb*		End of domain parameters object

Table 46. Format of the token data object

Offset (decimal) 188 +	Length of field (bytes)	Description
Object header		
0	4	Eye catcher for data object: “DATA”
4	2	Version: EBCDIC '00'
6	2	Length of object, in bytes
8	4	Flags (see Table 34 on page 201)
Object type-specific section		
12	4	Reserved for IBM's use
16	28	Reserved for IBM's use
44	2	Length of VALUE attribute in bytes (*aa*)
46	2	Length of OBJECT_ID attribute in bytes (*bb*)
48	2	Length of LABEL attribute in bytes (*cc*)
50	2	Length of APPLICATION attribute in bytes (*dd*)
52	2	Length of ID attribute in bytes (*ee*)
54	22	Reserved for IBM's use

Table 46. Format of the token data object (continued)

Offset (decimal) 188 +	Length of field (bytes)	Description
76	4	Offset of VALUE attribute in bytes
80	4	Offset of OBJECT_ID attribute in bytes
84	4	Offset of LABEL attribute in bytes
88	4	Offset of APPLICATION attribute in bytes
92	4	Offset of ID attribute in bytes
96	44	Reserved for IBM's use
140	*aa + bb + cc + dd + ee*	Data attributes (variable length)
140 + *aa + bb + cc + dd + ee*		End of data object

AES Key Token Format

AES Internal Key Token

Table 47 shows the format for an AES internal key token.

Table 47. Internal Key Token Format

Bytes	Description
0	X'01' (flag indicating this is an internal key token)
1–3	Implementation-dependent bytes (X'000000' for ICSF)
4	Key token version number (X'04')
5	Reserved - must be set to X'00'
6	Flag byte **Bit** **Meaning When Set On** **0** Encrypted key and master key verification pattern (MKVP) are present. Off for a clear key token, on for an encrypted key token. **1** Control vector (CV) value in this token has been applied to the key. **2** No key is present or the AES MKVP is not present if the key is encrypted. **3- 7** Reserved. Must be set to 0.
7	1-byte LRC checksum of clear key value.
8–15	Master key verification pattern (MKVP) (For a clear AES key token this value will be hex zeros.)
16–47	128-bit, 192-bit, or 256-bit key value, left-justified and padded on the right with hex zeros.
48–55	8-byte control vector. (For a clear AES key token this value will be hex zeros.)
56–57	2-byte integer specifying the length in bits of the clear key value.
58–59	2-byte integer specifying the length in bytes of the encrypted key value. (For a clear AES key token this value will be hex zeros.)
60–63	Token validation value (TVV). See “Token Validation Value” on page 218 for more information.

Token Validation Value

ICSF uses the *token validation value (TVV)* to verify that a token is valid. The TVV prevents a key token that is not valid or that is overlaid from being accepted by ICSF. It provides a checksum to detect a corruption in the key token.

When an ICSF callable service generates a key token, it generates a TVV and stores the TVV in bytes 60-63 of the key token. When an application program passes a key token to a callable service, ICSF checks the TVV. To generate the TVV, ICSF performs a twos complement ADD operation (ignoring carries and overflow) on the key token, operating on four bytes at a time, starting with bytes 0-3 and ending with bytes 56-59.

DES Key Token Formats

DES Internal Key Token

Table 48 shows the format for a DES internal key token.

Table 48. Internal Key Token Format

Bytes	Description
0	X'01' (flag indicating this is an internal key token)
1–3	Implementation-dependent bytes (X'000000' for ICSF)
4	Key token version number (X'00' or X'01')
5	Reserved (X'00')
6	Flag byte **Bit** **Meaning When Set On** **0** Encrypted key and master key verification pattern (MKVP) are present. **1** Control vector (CV) value in this token has been applied to the key. **2** Key is used for no control vector (NOCV) processing. Valid for transport keys only. **3** Key is an ANSI key-encrypting key (AKEK). **4** AKEK is a double-length key (16 bytes). **Note:** When bit 3 is on and bit 4 is off, AKEK is a single-length key (8 bytes). **5** AKEK is partially notarized. **6** Key is an ANSI partial key. **7** Export prohibited.
7	**Bit** **Meaning When Set On** **0-2** Key value encryption method. • 000 - the key is encrypted using the original CCA method (ECB). • 001 - the key is encrypted using the X9.24 enhanced method (CBC). These bits are ignored if the token contains no key or a clear key. **3-7** Reserved.
8–15	Master key verification pattern (MKVP)
16–23	A single-length key, the left half of a double-length key, or Part A of a triple-length key. The value is encrypted under the master key when flag bit 0 is on, otherwise it is in the clear.

Table 48. Internal Key Token Format (continued)

Bytes	Description
24–31	X'0000000000000000' if a single-length key, or the right half of a double-length operational key, or Part B of a triple-length operational key. The right half of the double-length key or Part B of the triple-length key is encrypted under the master key when flag bit 0 is on, otherwise it is in the clear.
32–39	The control vector (CV) for a single-length key or the left half of the control vector for a double-length key.
40–47	X'0000000000000000' if a single-length key or the right half of the control vector for a double-length operational key.
48–55	X'0000000000000000' if a single-length key or double-length key, or Part C of a triple-length operational key. Part C of a triple-length key is encrypted under the master key when flag bit 0 is on, otherwise it is in the clear.
56-58	Reserved (X'000000')
59 bits 0 and 1	**B'10'** Indicates CDMF DATA or KEK. **B'00'** Indicates DES for DATA keys or the system default algorithm for a KEK. **B'01'** Indicates DES for a KEK.
59 bits 2 and 3	**B'00'** Indicates single-length key (version 0 only). **B'01'** Indicates double-length key (version 1 only). **B'10'** Indicates triple-length key (version 1 only).
59 bits 4 –7	B'0000'
60–63	Token validation value (TVV).

Note: A key token stored in the CKDS will not have an MKVP or TVV. Before such a key token is used, the MKVP is copied from the CKDS header record and the TVV is calculated and placed in the token. See “Token Validation Value” on page 218 for more information.

DES External Key Token

Table 49 shows the format for a DES external key token.

Table 49. Format of External Key Tokens

Bytes	Description
0	X'02' (flag indicating an external key token)
1	Reserved (X'00')
2–3	Implementation-dependent bytes (X'0000' for ICSF)
4	Key token version number (X'00' or X'01')
5	Reserved (X'00')
6	Flag byte **Bit** **Meaning When Set On** **0** Encrypted key is present. **1** Control vector (CV) value has been applied to the key. Other bits are reserved and are binary zeros.
7	**Bit** **Meaning When Set On** **0-2** Key value encryption method. • 000 - the key is encrypted using the original CCA method (ECB). • 001 - the key is encrypted using the X9.24 enhanced method (CBC). These bits are ignored if the token contains no key or a clear key. **3-7** Reserved.
8–15	Reserved (X'0000000000000000')
16–23	Single-length key or left half of a double-length key, or Part A of a triple-length key. The value is encrypted under a transport key-encrypting key when flag bit 0 is on, otherwise it is in the clear.
24–31	X'0000000000000000' if a single-length key or right half of a double-length key, or Part B of a triple-length key. The right half of a double-length key or Part B of a triple-length key is encrypted under a transport key-encrypting key when flag bit 0 is on, otherwise it is in the clear.
32–39	Control vector (CV) for single-length key or left half of CV for double-length key
40–47	X'0000000000000000' if single-length key or right half of CV for double-length key
48–55	X'0000000000000000' if a single-length key, double-length key, or Part C of a triple-length key. This key part is encrypted under a transport key-encrypting key when flag bit 0 is on, otherwise it is in the clear.
56–58	Reserved (X'000000')
59 bits 0 and 1	B'00'
59 bits 2 and 3	**B'00'** Indicates single-length key (version 0 only). **B'01'** Indicates double-length key (version 1 only). **B'10'** Indicates triple-length key (version 1 only).
59 bits 4–7	B'0000'
60-63	Token validation value (see “Token Validation Value” on page 218 for a description).

External RKX DES Key Token

Table 50 on page 221 defines an external DES key-token called an *RKX key-token*. An RKX key-token is a special token used exclusively by the Remote Key Export

(CSNDRKX) and DES key-storage callable services (for example, Key Record Write). No other callable services use or reference an RKX key-token or key-token record.

Note: Callable services other than CSNDRKX and the DES key-storage do not support RKX key tokens or RKX key token records.

As can be seen in the table, RKX key tokens are 64 bytes in length, have a token identifier flag (X'02'), a token version number (X'10'), and room for encrypted keys like normal CCA DES key tokens. Unlike normal CCA DES key-tokens, RKX key tokens do not have a control vector, flag bits, and a token-validation value. In addition, they have a confounder value, a MAC value, and room for a third encrypted key.

Table 50. External RKX DES key-token format, version X'10'

Offset	Length	Meaning
00	1	X'02' (a token identifier flag that indicates an external key-token)
01	3	Reserved, binary zero
04	1	The token version number (X'10')
05	2	Reserved, binary zero
07	1	Key length in bytes, including confounder
08	8	Confounder
16	8	Key left
24	8	Key middle (binary zero if not used)
32	8	Key right (binary zero if not used)
40	8	Rule ID The trusted block rule identifier used to create this key token. A subsequent call to Remote Key Export (CSNDRKX) can use this token with a trusted block rule that references the rule ID that must have been used to create this token. The trusted block rule can be compared with this rule ID for verification purposes. The Rule ID is an 8-byte string of ASCII characters, left justified and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.
48	8	Reserved, binary zero
56	8	MAC value ISO 16609 TDES CBC-mode MAC, computed over the 56 bytes starting at offset 0 and including the encrypted key value and the rule ID using the same MAC key that is used to protect the trusted block itself. This MAC value guarantees that the key and the rule ID cannot be modified without detection, providing integrity and binding the rule ID to the key itself. This MAC value must verify with the same trusted block used to create the key, thus binding the key structure to that specific trusted block.

Notes:

1. A fixed, randomly derived variant is exclusive-ORed with the MAC key before it is used to encipher the generated or exported key and confounder.

2. The MAC key is located within a trusted block (internal format) and can be recovered by decipherment under a variant of the PKA master key.
3. The trusted block is originally created in external form by the CSNDTBC callable service and then converted to internal form by the CSNDPKI callable service prior to the CSNDRKX call.

DES Null Key Token

Table 51 shows the format for a DES null key token.

Table 51. Format of Null Key Tokens

Bytes	Description
0	X'00' (flag indicating this is a null key token).
1–15	Reserved (set to binary zeros).
16–23	Single-length encrypted key, or left half of double-length encrypted key, or Part A of triple-length encrypted key.
24–31	X'0000000000000000' if a single-length encrypted key, the right half of double-length encrypted key, or Part B of triple-length encrypted key.
32–39	X'0000000000000000' if a single-length encrypted key or double-length encrypted key.
40–47	Reserved (set to binary zeros).
48–55	Part C of a triple-length encrypted key.
56–63	Reserved (set to binary zeros).

Variable-length Symmetric Key Token Formats

Variable-length Symmetric Key Token

The following table presents the presents the format for a variable-length symmetric key token. The length of the token depends on the key type and algorithm.

Table 52. Variable-length Symmetric Key Token

Offset (Dec)	Length of Field (Bytes)	Description
Header		
0	1	Token flag **X'00'** for null token **X'01'** for internal tokens **X'02'** for external tokens
1	1	Reserved (X'00')
2	2	Length of the token in bytes
4	1	Token version number X'05'
5	3	Reserved (X'000000')
Wrapping information		

Table 52. Variable-length Symmetric Key Token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
8	1	Key material state. **X'00'** no key present (internal or external) **X'01'** key is clear (internal) **X'02'** key is encrypted under a key-encrypting key (external) **X'03'** key is encrypted under the master key (internal)
9	1	Key verification pattern (KVP) type. **X'00'** No KVP **X'01'** AES master key verification pattern **X'02'** key-encrypting key verification pattern
10	16	Verification pattern of the key used to wrap the payload. Value is left justified.
26	1	Wrapping method - This value indicates the wrapping method used to protect the data in the encrypted section. **X'00'** key is in the clear **X'02'** AESKW **X'03'** PKOAEP2
27	1	Hash algorithm used in wrapping algorithm. • For wrapping method X'00' **X'00'** None. For clear key tokens. • For wrapping method X'02' **X'02'** SHA-256 • For wrapping method X'03' **X'01'** SHA-1 **X'02'** SHA-256 **X'04'** SHA-384 **X'08'** SHA-512
28	2	Reserved (X'0000')
AESKW Components: Associated data and clear key or encrypted AESKW payload		
Associated data section		
30	1	Associated data version (X'01')
31	1	Reserved (X'00')
32	2	Length of the associated data in bytes: *adl*
34	1	Length of the key name in bytes: *kl*
35	1	Length of the IBM extended associated data in bytes: *iead*
36	1	Length of the installation-definable associated data in bytes: *uad*
37	1	Reserved (X'00')
38	2	Length of the payload in bits: *pl*
40	1	Reserved (X'00')

Table 52. Variable-length Symmetric Key Token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
41	1	Type of algorithm for which the key can be used **X'02'** AES **X'03'** HMAC
42	2	Key type: For algorithm AES: **X'0001'** CIPHER **X'0003'** EXPORTER **X'0004'** IMPORTER For algorithm HMAC: **X'0002'** MAC
44	1	Key-usage field count (*kuf*) - (1 byte)
45	*kuf* * 2	Key-usage fields (*kuf* * 2 bytes) • For HMAC algorithm keys, refer to Table 53 on page 227. • For AES algorithm Key-Encrypting Keys (Exporter or Importer), refer to Table 54 on page 228. • For AES algorithm Cipher Keys, refer to Table 55 on page 231.
45 + *kuf* * 2	1	Key-management field count (*kmf*): 2 (no pedigree information) or 3 (has pedigree information)
46 + *kuf* * 2	2	Key-management field 1 High-order byte: **1xxx xxxx** Allow export using symmetric key **x1xx xxxx** Allow export using unauthenticated asymmetric key **xx1x xxxx** Allow export using authenticated asymmetric key **xxx1 xxxx** Allow export in RAW format. All other bits are reserved and must be zero. Low-order byte: --symmetric-- **1xxx xxxx** Prohibit export using DES key. **x1xx xxxx** Prohibit export using AES key. --asymmetric-- **xxxx 1xxx** Prohibit export using RSA key. All other bits are reserved and must be zero.

Table 52. Variable-length Symmetric Key Token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
48 + *kuf* * 2	2	Key-management field 2 High-order byte: **11xx xxxx** Key, if present, is incomplete. Key requires at least 2 more parts. **10xx xxxx** Key, if present, is incomplete. Key requires at least 1 more part. **01xx xxxx** Key, if present, is incomplete. Key can be completed or have more parts added. **00xx xxxx** Key, if present, is complete. No more parts can be added. All other bits are reserved and must be zero. Low-order byte (Security History): **xxx1 xxxx** Key was encrypted with an untrusted KEK **xxxx 1xxx** Key was in a format without type/usage attributes **xxxx x1xx** Key was encrypted with key weaker than itself **xxxx xx1x** Key was in a non-CCA format **xxxx xxx1** Key was encrypted in ECB mode. All other bits are reserved and must be zero.
50 + *kuf* * 2	2	Key-management field 3 - Pedigree (this field may or may not be present) Indicates how key was originally created and how it got into the current system. High-order byte: Pedigree Original. **X'00'** Unknown (Key Token Build2, Key Translate2) **X'01'** Other - method other than those defined here, probably used in UDX **X'02'** Randomly Generated (Key Generate2) **X'03'** Established by key agreement (ECC Diffie-Hellman) **X'04'** Created from cleartext key components (Key Part Import2) **X'05'** Entered as a cleartext key value (Key Part Import2, Secure Key Import2) **X'06'** Derived from another key **X'07'** Cleartext keys or key parts that were entered at TKE and secured from there to the target card (operational key load) All unused values are reserved and undefined.

Table 52. Variable-length Symmetric Key Token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
		Low-order byte: Pedigree Current. **X'00'** Unknown (Key Token Build2) **X'01'** Other - method other than those defined here, probably used in UDX **X'02'** Randomly Generated (Key Generate2) **X'03'** Established by key agreement (ECC Diffie-Hellman) **X'04'** Created from cleartext key components (Key Part Import2) **X'05'** Entered as a cleartext key value (Key Part Import2, Secure Key Import2) **X'06'** Derived from another key **X'07'** Imported from a CCA 05 variable length token with pedigree field (Symmetric Key Import2) **X'08'** Imported from a CCA 05 variable length token with no pedigree field (Symmetric Key Import2) **X'09'** Imported from a CCA token that had a CV **X'0A'** Imported from a CCA token that had no CV or a zero CV **X'0B'** Imported from a TR-31 key block that contained a CCA CV (ATTR-CV option) (TR-31 Import) **X'0C'** Imported from a TR-31 key block that did not contain a CCA CV (TR-31 Import) **X'0D'** Imported using PKCS 1.2 RSA encryption (Symmetric Key Import2) **X'0E'** Imported using PKCS OAEP encryption (Symmetric Key Import2) **X'0F'** Imported using PKA92 RSA encryption (Symmetric Key Import2) **X'10'** Imported using RSA ZERO-PAD encryption (Symmetric Key Import2) **X'11'** Converted from a CCA token that had a CV (Key Translate2) **X'12'** Converted from a CCA token that had no CV or a zero CV (Key Translate2) **X'13'** Cleartext keys or key parts that were entered at TKE and secured from there to the target card (operational key load) **X'14'** Exported from a CCA 05 variable length token with pedigree field (Symmetric Key Export) **X'15'** Exported from a CCA 05 variable length token with no pedigree field (Symmetric Key Export) **X'16'** Exported using PKCS OAEP encryption (Symmetric Key Export) All unused values are reserved and undefined.
46 + *kuf* * 2 + *kmf* * 2	*kl*	Key name
46 + *kuf* * 2 + *kmf* * 2 + *kl*	*iead*	IBM extended associated data
46 + *kuf* * 2 + *kmf* * 2 + *kl* + *iead*	*uad*	Installation-defined associated data

Table 52. Variable-length Symmetric Key Token (continued)

Offset (Dec)	Length of Field (Bytes)	Description
Clear key or encrypted payload		
30 + *adl*	*(pl+7)/8*	**Encrypted AESKW payload (internal keys)**: The encrypted AESKW payload is created from the unencrypted AESKW payload which is made up of the ICV/pad length/hash options and hash length/hash options/hash of the associated data/key material/padding. See unencrypted AESKW payload below. **Encrypted PKOAEP2 payload (external keys)**: The encrypted PKOAEP2 payload is created using the PKCS #1 v1.2 encoding method for a given hash algorithm. The message (M) inside the encoding contains: [2 bytes: bit length of key] || [clear HMAC key]. M is encoded using OAEP and then encrypted with an RSA public key according to the standard. **Clear key payload**: When the key is clear, only the key material will be in the payload padded to the nearest byte with binary zeros.
30 + *adl* + (pl+7)/8		End of AESKW components
Unencrypted AESKW payload (This data will never appear in the clear outside of the cryptographic coprocessor)		
0	6	Integrity check value. Six byte constant: X'A6A6A6A6A6A6'.
6	1	Length of the padding in bits: *pb*
7	1	Length of hash options and hash of the associated data in bytes (*hoh*)
8	4	Hash options
12	*hoh* - 4	Hash of the associated data
8 + *hoh*	(*pl*/8) – 8 - *hoh*	Key data and padding (key data is left justified).
pl/8		*pl* is the bit length of the payload

Table 53. HMAC Algorithm Key-usage fields

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (*kuf*): 2

Table 53. HMAC Algorithm Key-usage fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	Key-usage field 1 High-order byte: **1xxx xxxx** Key can be used for generate. **x1xx xxxx** Key can be used for verify. All unused bits are reserved and must be zero. Low-order byte: **xxxx 1xxx** The key can only be used in UDXs (used in KGN, KIM, KEX). **xxxx 0xxx** The key can be used in both UDXs and CCA. **xxxx xuuu** Reserved for UDXs, where uuu are UDX-defined bits. All unused bits are reserved and must be zero.
47	2	Key-usage field 2 High-order byte: **1xxx xxxx** SHA-1 hash method is allowed for the key. **x1xx xxxx** SHA-224 hash method is allowed for the key. **xx1x xxxx** SHA-256 hash method is allowed for the key. **xxx1 xxxx** SHA-384 hash method is allowed for the key. **xxxx 1xxx** SHA-512 hash method is allowed for the key. All unused bits are reserved and must be zero. Low-order byte: All bits are reserved and must be zero.

Table 54. AES Algorithm KEK Key-usage fields

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (*kuf*): 4

Table 54. AES Algorithm KEK Key-usage fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
45	2	Key-usage field 1 High-order byte for EXPORTER: **1xxx xxxx** Key can be used for EXPORT. **x1xx xxxx** Key can be used for TRANSLAT. **xx1x xxxx** Key can be used for GENERATE-OPEX. **xxx1 xxxx** Key can be used for GENERATE-IMEX. **xxxx 1xxx** Key can be used for GENERATE-EXEX. **xxxx x1xx** Key can be used for GENERATE-PUB. All unused bits are reserved and must be zero. High-order byte for IMPORTER: **1xxx xxxx** Key can be used for IMPORT. **x1xx xxxx** Key can be used for TRANSLAT. **xx1x xxxx** Key can be used for GENERATE-OPIM. **xxx1 xxxx** Key can be used for GENERATE-IMEX. **xxxx 1xxx** Key can be used for GENERATE-IMIM. **xxxx x1xx** Key can be used for GENERATE-PUB. All unused bits are reserved and must be zero. Low-order byte: **xxxx 1xxx** The key can only be used in UDXs (used in KGN, KIM, KEX). **xxxx 0xxx** The key can be used in both UDXs and CCA. **xxxx xuuu** Reserved for UDXs, where uuu are UDX-defined bits. All unused bits are reserved and must be zero.

Table 54. AES Algorithm KEK Key-usage fields (continued)

Offset (Dec)	Length of Field (Bytes)	Description
47	2	Key-usage field 2 High-order byte: **1xxx xxxx** Key can wrap a TR-31 key. All unused bits are reserved and must be zero. Low-order byte: **xxxx xxx1** This KEK can export a key in RAW format. All unused bits are reserved and must be zero
49	2	Key-usage field 3 High-order byte: **1xxx xxxx** Key can wrap DES keys **x1xx xxxx** Key can wrap AES keys **xx1x xxxx** Key can wrap HMAC keys **xxx1 xxxx** Key can wrap RSA keys **xxxx 1xxx** Key can wrap ECC keys All unused bits are reserved and must be zero. Low-order byte: All bits are reserved and must be zero.
51	2	Key-usage field 4 High-order byte: **1xxx xxxx** Key can wrap DATA class keys **x1xx xxxx** Key can wrap KEK class keys **xx1x xxxx** Key can wrap PIN class keys **xxx1 xxxx** Key can wrap DERIVATION class keys **xxxx 1xxx** Key can wrap CARD class keys All unused bits are reserved and must be zero. Low-order byte: All bits are reserved and must be zero.

Table 55. AES Algorithm Cipher Key Associated Data

Offset (Dec)	Length of Field (Bytes)	Description
44	1	Key-usage field count (*kuf*): 2
45	2	Key-usage field 1 High-order byte: **1xxx xxxx** Key can be used for encryption. **x1xx xxxx** Key can be used for decryption. All unused bits are reserved and must be zero. Low-order byte: **xxxx 1xxx** The key can only be used in UDXs (used in KGN, KIM, KEX). **xxxx 0xxx** The key can be used in both UDXs and CCA. **xxxx xuuu** Reserved for UDXs, where uuu are UDX-defined bits. All unused bits are reserved and must be zero.
47	2	Key-usage field 2 High-order byte: **X'00'** Key can be used for Cipher Block Chaining (CBC). **X'01'** Key can be used for Electronic Code Book (ECB). **X'02'** Key can be used for Cipher Feedback (CFB). **X'03'** Key can be used for Output Feedback (OFB). **X'04'** Key can be used for Galois/Counter Mode (GCM) **X'05'** Key can be used for XEX-based Tweaked CodeBook Mode with CipherText Stealing (XTS) All unused values are reserved and must not be used. Low-order byte: All bits are reserved and must be zero.

Variable-length Symmetric Null Key Token

The following table shows the format for a variable-length symmetric null key token.

Table 56. Variable-length Symmetric Null Token

Bytes	Description
0	X'00' Token identifier (indicates that this is a null key token).
1	Version, X'00'.
2-3	X'0008' Length of the key token structure.
4-7	Ignored (zero).

PKA Key Token Formats

As with DES key tokens, the first byte of a PKA key token indicates the type of token. If the first byte of the key identifier is X'1E' or X'1F', this indicates that it is a **PKA key token**.

A first byte of X'1E' indicates an external token with a cleartext public key and optionally a private key that is either in cleartext or enciphered by a transport key-encrypting key.

A first byte of X'1F' indicates an internal token with a cleartext public key and a private key that is enciphered by the master key and ready for internal use.

Although DES tokens are 64 bytes, PKA tokens are of variable length because they contain either RSA or DSS key values, which are variable in length. Consequently, length parameters precede all PKA token parameters. The maximum allowed size is 3500 bytes. PKA key tokens consist of a token header, any required sections, and optional sections, which depend on the token type.

A PKA key token can be a public or private key token, and a private key token can be internal or external. Therefore, there are three basic types of tokens, each of which can contain either RSA or DSS information:

- Public key tokens
- Private external key tokens
- Private internal key tokens

Public key tokens contain only the public key. Private key tokens contain the public and private key pair.

Internal PKA Tokens

Programming Interface information

PKA private internal key tokens contain both private and public key information. There is no need for an internal token with only the public key information because the public values are in the clear.

The first byte of X'1F' indicates an internal token with a cleartext public key and a private key that is enciphered with a PKA master key and ready for local (internal) use.

The format of a PKA private internal key token is similar to that of a private external token. The only differences are changes in the private key section and the addition of some internal information at the end of the token. This last section starts with the eyecatcher 'PKTN' rather than with a token or section marker.

End of Programming Interface information

PKA Null Key Token

Table 57 shows the format for a PKA null key token.

Table 57. Format of PKA Null Key Tokens

Bytes	Description
0	X'00' Token identifier (indicates that this is a null key token).
1	Version, X'00'
2–3	X'0008' Length of the key token structure.
4–7	Ignored (should be zero).

RSA Key Token Formats

RSA Public Key Token

An RSA public key token contains the following sections:

- A required token header, starting with the token identifier X'1E'
- A required RSA public key section, starting with the section identifier X'04'

Table 58 presents the format of an RSA public key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format).

Table 58. RSA Public Key Token

Offset (Dec)	Number of Bytes	Description
Token Header (required)		
000	001	Token identifier. X'1E' indicates an external token.
001	001	Version, X'00'.
002	002	Length of the key token structure.
004	004	Ignored. Should be zero.
RSA Public Key Section (required)		
000	001	X'04', section identifier, RSA public key.
001	001	X'00', version.
002	002	Section length, 12+xxx+yyy.
004	002	Reserved field.
006	002	RSA public key exponent field length in bytes, “xxx”.
008	002	Public key modulus length in bits.
010	002	RSA public key modulus field length in bytes, “yyy”.
012	xxx	Public key exponent (this is generally a 1-, 3-, or 64- to 512-byte quantity), e. e must be odd and 1<e<n. (Frequently, the value of e is 2^{16}+1)
12+xxx	yyy	Modulus, n.

RSA Private External Key Token

An RSA private external key token contains the following sections:

- A required PKA token header starting with the token identifier X'1E'
- A required RSA private key section starting with one of the following section identifiers:

  - X'02' which indicates a modulus-exponent form RSA private key section (not optimized) with modulus length of up to 1024 bits for use with the Cryptographic Coprocessor Feature or the PCI Cryptographic Coprocessor.
  - X'08' which indicates an optimized Chinese Remainder Theorem form private key section with modulus bit length of up to 4096 bits for use with the PCICC, PCIXCC, CEX2C, or CEX3C.
  - X'09' which indicates a modulus-exponent form RSA private key section (not optimized) with modulus length of up to 4096 bits for use with the CEX2C or CEX3C.
- A required RSA public key section, starting with the section identifier X'04'
- An optional private key name section, starting with the section identifier X'10'

Table 59 presents the basic record format of an RSA private external key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format). All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

Table 59. RSA Private External Key Token Basic Record Format

Offset (Dec)	Number of Bytes	Description
Token Header (required)		
000	001	Token identifier. X'1E' indicates an external token. The private key is either in cleartext or enciphered with a transport key-encrypting key.
001	001	Version, X'00'.
002	002	Length of the key token structure.
004	004	Ignored. Should be zero.
RSA Private Key Section (required) • For 1024-bit Modulus-Exponent form refer to “RSA Private Key Token, 1024-bit Modulus-Exponent External Form” on page 235 • For 4096-bit Modulus-Exponent form refer to “RSA Private Key Token, 4096-bit Modulus-Exponent External Form” on page 236 • For 4096-bit Chinese Remainder Theorem form refer to “RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Form” on page 237		
RSA Public Key Section (required)		
000	001	X'04', section identifier, RSA public key.
001	001	X'00', version.
002	002	Section length, 12+xxx.
004	002	Reserved field.
006	002	RSA public key exponent field length in bytes, “xxx”.
008	002	Public key modulus length in bits.
010	002	RSA public key modulus field length in bytes, which is zero for a private token. **Note:** In an RSA private key token, this field should be zero. The RSA private key section contains the modulus.
012	xxx	Public key exponent, e (this is generally a 1-, 3-, or 64- to 512-byte quantity). e must be odd and 1<e<n. (Frequently, the value of e is 2^{16}+1 (=65,537).
Private Key Name (optional)		
000	001	X'10', section identifier, private key name.

Table 59. RSA Private External Key Token Basic Record Format (continued)

Offset (Dec)	Number of Bytes	Description
001	001	X'00', version.
002	002	Section length, X'0044' (68 decimal).
004	064	Private key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.

RSA Private Key Token, 1024-bit Modulus-Exponent External Form: This RSA private key token and the external X'02' token is supported on the Cryptographic Coprocessor Feature and PCI Cryptographic Coprocessor.

Table 60. RSA Private Key Token, 1024-bit Modulus-Exponent External Format

Offset (Dec)	Number of Bytes	Description
000	001	X'02', section identifier, RSA private key, modulus-exponent format (RSA-PRIV)
001	001	X'00', version.
002	002	Length of the RSA private key section X'016C' (364 decimal).
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.
024	004	Reserved; set to binary zero.
028	001	Key format and security: **X'00'** Unencrypted RSA private key subsection identifier. **X'82'** Encrypted RSA private key subsection identifier.
029	001	Reserved, binary zero.
030	020	SHA-1 hash of the optional key-name section. If there is no key-name section, then 20 bytes of X'00'.
050	004	Key use flag bits. **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. **6** The key is translatable. All other bits reserved, set to binary zero.
054	006	Reserved; set to binary zero.
060	024	Reserved; set to binary zero.
084	Start of the optionally-encrypted secure subsection.	
084	024	Random number, confounder.
108	128	Private-key exponent, d. $d=e^{-1} \bmod((p-1)(q-1))$, and $1<d<n$ where e is the public exponent.
	End of the optionally-encrypted subsection; the confounder field and the private-key exponent field are enciphered for key confidentiality when the key format and security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the ede2 algorithm.	
236	128	Modulus, n. n=pq where p and q are prime and $1<n<2^{1024}$.

RSA Private Key Token, 4096-bit Modulus-Exponent External Form: This RSA private key token and the external X'09' token is supported on the Crypto Express2 Coprocessor and Crypto Express3 Coprocessor.

Table 61. RSA Private Key Token, 4096-bit Modulus-Exponent External Format

Offset (Dec)	Number of Bytes	Description
000	001	X'09', section identifier, RSA private key, modulus-exponent format (RSAMEVAR).
001	001	X'00', version.
002	002	Length of the RSA private key section 132+ddd+nnn+xxx.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.
024	002	Length of the encrypted private key section 8+ddd+xxx.
026	002	Reserved; set to binary zero.
028	001	Key format and security: **X'00'** Unencrypted RSA private key subsection identifier. **X'82'** Encrypted RSA private key subsection identifier.
029	001	Reserved, set to binary zero.
030	020	SHA-1 hash of the optional key-name section. If there is no key-name section, then 20 bytes of X'00'.
050	001	Key use flag bits. **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. **6** The key is translatable All other bits reserved, set to binary zero.
051	001	Reserved; set to binary zero.
052	048	Reserved; set to binary zero.
100	016	Reserved; set to binary zero.
116	002	Length of private exponent, d, in bytes: ddd.
118	002	Length of modulus, n, in bytes: nnn.
120	002	Length of padding field, in bytes: xxx.
122	002	Reserved; set to binary zero.
124	Start of the optionally-encrypted secure subsection.	
124	008	Random number, confounder.
132	ddd	Private-key exponent, d. $d=e^{-1} \bmod((p-1)(q-1))$, and $1<d<n$ where e is the public exponent.
132+ddd	xxx	X'00' padding of length xxx bytes such that the length from the start of the random number above to the end of the padding field is a multiple of eight bytes.
	End of the optionally-encrypted subsection; the confounder field and the private-key exponent field are enciphered for key confidentiality when the key format and security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the ede2 algorithm.	
132+ddd+xxx	nnn	Modulus, n. n=pq where p and q are prime and $1<n<2^{4096}$.

RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Form: This RSA private key token (up to 2048-bit modulus) is supported on the PCICC, PCIXCC, CEX2C, or CEX3C. The 4096-bit modulus private key token is supported on the z9 EC, z9 BC, z10 EC and z10 BC with the Nov. 2007 or later version of the licensed internal code installed on the CEX2C or CEX3C.

Table 62. RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Format

Offset (Dec)	Number of Bytes	Description
000	001	X'08', section identifier, RSA private key, CRT format (RSA-CRT)
001	001	X'00', version.
002	002	Length of the RSA private-key section, 132 + ppp + qqq + rrr + sss + uuu + xxx + nnn.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the end of the modulus.
024	004	Reserved; set to binary zero.
028	001	Key format and security: **X'40'** Unencrypted RSA private-key subsection identifier, Chinese Remainder form. **X'42'** Encrypted RSA private-key subsection identifier, Chinese Remainder form.
029	001	Reserved; set to binary zero.
030	020	SHA-1 hash of the optional key-name section and any following optional sections. If there are no optional sections, then 20 bytes of X'00'.
050	004	Key use flag bits. **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. **6** The key is translatable. All other bits reserved, set to binary zero.
054	002	Length of prime number, p, in bytes: ppp.
056	002	Length of prime number, q, in bytes: qqq.
058	002	Length of d_p, in bytes: rrr.
060	002	Length of d_q, in bytes: sss.
062	002	Length of U, in bytes: uuu.
064	002	Length of modulus, n, in bytes: nnn.
066	004	Reserved; set to binary zero.
070	002	Length of padding field, in bytes: xxx.
072	004	Reserved, set to binary zero.
076	016	Reserved, set to binary zero.
092	032	Reserved; set to binary zero.
124	Start of the optionally-encrypted secure subsection.	
124	008	Random number, confounder.
132	ppp	Prime number, p.
132 + ppp	qqq	Prime number, q
132 + ppp + qqq	rrr	d_p = d mod(p - 1)

Table 62. RSA Private Key Token, 4096-bit Chinese Remainder Theorem External Format (continued)

Offset (Dec)	Number of Bytes	Description
132 + ppp + qqq + rrr	sss	d_q = d mod(q - 1)
132 + ppp + qqq + rrr + sss	uuu	U = q^{-1}mod(p).
132 + ppp + qqq + rrr + sss + uuu	xxx	X'00' padding of length xxx bytes such that the length from the start of the random number above to the end of the padding field is a multiple of eight bytes.
	End of the optionally-encrypted secure subsection; all of the fields starting with the confounder field and ending with the variable length pad field are enciphered for key confidentiality when the key format-and-security flags (offset 28) indicate that the private key is enciphered. They are enciphered under a double-length transport key using the TDES (CBC outer chaining) algorithm.	
132 + ppp + qqq + rrr + sss + uuu + xxx	nnn	Modulus, n. n = pq where p and q are prime and $1<n<2^{4096}$.

RSA Private Internal Key Token

An RSA private internal key token contains the following sections:

- A required PKA token header, starting with the token identifier X'1F'
- basic record format of an RSA private internal key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format). All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

Table 63. RSA Private Internal Key Token Basic Record Format

Offset (Dec)	Number of Bytes	Description
Token Header (required)		
000	001	Token identifier. X'1F' indicates an internal token. The private key is enciphered with a PKA master key.
001	001	Version, X'00'.
002	002	Length of the key token structure excluding the internal information section.
004	004	Ignored; should be zero.
RSA Private Key Section and Secured Subsection (required)		
• For 1024-bit X'02' Modulus-Exponent form refer to “RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for Cryptographic Coprocessor Feature” on page 239 • For 1024-bit X'06' Modulus-Exponent form refer to “RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for PCICC, PCIXCC, CEX2C, or CEX3C” on page 240 • For 4096-bit X'08' Chinese Remainder Theorem form refer to “RSA Private Key Token, 4096-bit Chinese Remainder Theorem Internal Form” on page 241		
RSA Public Key Section (required)		
000	001	X'04', section identifier, RSA public key.
001	001	X'00', version.
002	002	Section length, 12+xxx.
004	002	Reserved field.
006	002	RSA public key exponent field length in bytes, “xxx”.
008	002	Public key modulus length in bits.

Table 63. RSA Private Internal Key Token Basic Record Format (continued)

Offset (Dec)	Number of Bytes	Description
010	002	RSA public key modulus field length in bytes, which is zero for a private token.
012	xxx	Public key exponent (this is generally a 1, 3, or 64 to 512 byte quantity), e. e must be odd and 1<e<n. (Frequently, the value of e is 2^{16}+1 (=65,537).
Private Key Name (optional)		
000	001	X'10', section identifier, private key name.
001	001	X'00', version.
002	002	Section length, X'0044' (68 decimal).
004	064	Private key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.
Internal Information Section (required)		
000	004	Eye catcher 'PKTN'.
004	004	PKA token type. **Bit** **Meaning When Set On** **0** RSA key. **1** DSS key. **2** Private key. **3** Public key. **4** Private key name section exists. **5** Private key unenciphered. **6** Blinding information present. **7** Retained private key.
008	004	Address of token header.
012	002	Total length of total structure including this information section.
014	002	Count of number of sections.
016	016	PKA master key hash pattern.
032	001	Domain of retained key.
033	008	Serial number of processor holding retained key.
041	007	Reserved.

RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for Cryptographic Coprocessor Feature:

Table 64. RSA Private Internal Key Token, 1024-bit ME Form for Cryptographic Coprocessor Feature

Offset (Dec)	Number of Bytes	Description
000	001	X'02', section identifier, RSA private key.
001	001	X'00', version.
002	002	Length of the RSA private key section X'016C' (364 decimal).

Table 64. RSA Private Internal Key Token, 1024-bit ME Form for Cryptographic Coprocessor Feature (continued)

Offset (Dec)	Number of Bytes	Description
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.
024	004	Reserved; set to binary zero.
028	001	Key format and security: **X'02'** RSA private key.
029	001	Format of external key from which this token was derived: **X'21'** External private key was specified in the clear. **X'22'** External private key was encrypted.
030	020	SHA-1 hash of the key token structure contents that follow the public key section. If no sections follow, this field is set to binary zeros.
050	001	Key use flag bits. **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. All other bits reserved, set to binary zero.
051	009	Reserved; set to binary zero.
060	048	Object Protection Key (OPK) encrypted under a PKA master key—can be under the Signature Master Key (SMK) or Key Management Master Key (KMMK) depending on key use.
108	128	Secret key exponent d, encrypted under the OPK. $d=e^{-1} \bmod((p-1)(q-1))$
236	128	Modulus, n. n=pq where p and q are prime and $1<n<2^{1024}$.

RSA Private Key Token, 1024-bit Modulus-Exponent Internal Form for PCICC, PCIXCC, CEX2C, or CEX3C:

Table 65. RSA Private Internal Key Token, 1024-bit ME Form for PCICC, PCIXCC, CEX2C, or CEX3C

Offset (Dec)	Number of Bytes	Description
000	001	X'06', section identifier, RSA private key modulus-exponent format (RSA-PRIV).
001	001	X'00', version.
002	002	Length of the RSA private key section X'0198' (408 decimal) + rrr + iii + xxx.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to and including the modulus at offset 236.
024	004	Reserved; set to binary zero.
028	001	Key format and security: **X'02'** RSA private key.
029	001	Format of external key from which this token was derived: **X'21'** External private key was specified in the clear. **X'22'** External private key was encrypted. **X'23'** Private key was generated using regeneration data. **X'24'** Private key was randomly generated.
030	020	SHA-1 hash of the optional key-name section and any following optional sections. If there are no optional sections, this field is set to binary zeros.

Table 65. RSA Private Internal Key Token, 1024-bit ME Form for PCICC, PCIXCC, CEX2C, or CEX3C (continued)

Offset (Dec)	Number of Bytes	Description
050	004	Key use flag bits. **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. All other bits reserved, set to binary zeros.
054	006	Reserved; set to binary zero.
060	048	Object Protection Key (OPK) encrypted under the Asymmetric Keys Master Key using the ede3 algorithm.
108	128	Private key exponent d, encrypted under the OPK using the ede5 algorithm. $d=e^{-1} \bmod ((p-1)(q-1))$, and $1<d<n$ where e is the public exponent.
236	128	Modulus, n. n=pq where p and q are prime and $2^{512}<n<2^{1024}$.
364	016	Asymmetric-Keys Master Key hash pattern.
380	020	SHA-1 hash value of the blinding information subsection cleartext, offset 400 to the end of the section.
400	002	Length of the random number r, in bytes: rrr.
402	002	Length of the random number r^{-1}, in bytes: iii.
404	002	Length of the padding field, in bytes: xxx.
406	002	Reserved; set to binary zeros.
408	Start of the encrypted blinding subsection	
408	rrr	Random number r (used in blinding).
408 + rrr	iii	Random number r^{-1} (used in blinding).
408 + rrr + iii	xxx	X'00' padding of length xxx bytes such that the length from the start of the encrypted blinding subsection to the end of the padding field is a multiple of eight bytes.
	End of the encrypted blinding subsection; all of the fields starting with the random number r and ending with the variable length pad field are encrypted under the OPK using TDES (CBC outer chaining) algorithm.	

RSA Private Key Token, 4096-bit Chinese Remainder Theorem Internal Form: This RSA private key token (up to 2048-bit modulus) is supported on the PCICC, PCIXCC, CEX2C, or CEX3C. The 4096-bit modulus private key token is supported on the z9 EC, z9 BC, z10 EC, z10 BC, or z196 with the Nov. 2007 or later version of the licensed internal code installed on the CEX2C or CEX3C.

Table 66. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format

Offset (Dec)	Number of Bytes	Description
000	001	X'08', section identifier, RSA private key, CRT format (RSA-CRT)
001	001	X'00', version.
002	002	Length of the RSA private-key section, 132 + ppp + qqq + rrr + sss + uuu + ttt + iii + xxx + nnn.
004	020	SHA-1 hash value of the private-key subsection cleartext, offset 28 to the end of the modulus.
024	004	Reserved; set to binary zero.

Table 66. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format (continued)

Offset (Dec)	Number of Bytes	Description
028	001	Key format and security: **X'08'** Encrypted RSA private-key subsection identifier, Chinese Remainder form.
029	001	Key derivation method: **X'21'** External private key was specified in the clear. **X'22'** External private key was encrypted. **X'23'** Private key was generated using regeneration data. **X'24'** Private key was randomly generated.
030	020	SHA-1 hash of the optional key-name section and any following sections. If there are no optional sections, then 20 bytes of X'00'.
050	004	Key use flag bits: **Bit** **Meaning When Set On** **0** Key management usage permitted. **1** Signature usage not permitted. All other bits reserved, set to binary zero.
054	002	Length of prime number, p, in bytes: ppp.
056	002	Length of prime number, q, in bytes: qqq.
058	002	Length of d_p, in bytes: rrr.
060	002	Length of d_q, in bytes: sss.
062	002	Length of U, in bytes: uuu.
064	002	Length of modulus, n, in bytes: nnn.
066	002	Length of the random number r, in bytes: ttt.
068	002	Length of the random number r^{-1}, in bytes: iii.
070	002	Length of padding field, in bytes: xxx.
072	004	Reserved, set to binary zero.
076	016	Asymmetric-Keys Master Key hash pattern.
092	032	Object Protection Key (OPK) encrypted under the Asymmetric-Keys Master Key using the TDES (CBC outer chaining) algorithm.
124	Start of the encrypted secure subsection, encrypted under the OPK using TDES (CBC outer chaining).	
124	008	Random number, confounder.
132	ppp	Prime number, p.
132 + ppp	qqq	Prime number, q
132 + ppp + qqq	rrr	$d_p = d \bmod (p - 1)$
132 + ppp + qqq + rrr	sss	$d_q = d \bmod (q - 1)$
132 + ppp + qqq + rrr + sss	uuu	$U = q^{-1} \bmod (p)$.
132 + ppp + qqq + rrr + sss + uuu	ttt	Random number r (used in blinding).
132 + ppp + qqq + rrr + sss + uuu + ttt	iii	Random number r^{-1} (used in blinding).

Table 66. RSA Private Internal Key Token, 4096-bit Chinese Remainder Theorem Internal Format (continued)

Offset (Dec)	Number of Bytes	Description
132 + ppp + qqq + rrr + sss + uuu + ttt + iii	xxx	X'00' padding of length xxx bytes such that the length from the start of the confounder at offset 124 to the end of the padding field is a multiple of eight bytes.
	End of the encrypted secure subsection; all of the fields starting with the confounder field and ending with the variable length pad field are encrypted under the OPK using TDES (CBC outer chaining) for key confidentiality.	
132 + ppp + qqq + rrr + sss + uuu + ttt + iii + xxx	nnn	Modulus, n. n = pq where p and q are prime and $1<n<2^{4096}$.

DSS Key Token Formats

DSS Public Key Token

A DSS public key token contains the following sections:

- A required token header, starting with the token identifier X'1E'
- A required DSS public key section, starting with the section identifier X'03'

Table 67 presents the format of a DSS public key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format).

Table 67. DSS Public Key Token

Offset (Dec)	Number of Bytes	Description
Token Header (required)		
000	001	Token identifier. X'1E' indicates an external token.
001	001	Version, X'00'.
002	002	Length of the key token structure.
004	004	Ignored. Should be zero.
DSS Public Key Section (required)		
000	001	X'03', section identifier, DSS public key.
001	001	X'00', version.
002	002	Section length, 14+ppp+qqq+ggg+yyy.
004	002	Size of p in bits. The size of p must be one of: 512, 576, 640, 704, 768, 832, 896, 960, or 1024.
006	002	Size of the p field in bytes, “ppp”.
008	002	Size of the q field in bytes, “qqq”.
010	002	Size of the g field in bytes, “ggg”.
012	002	Size of the y field in bytes, “yyy”.
014	ppp	Prime modulus (large public modulus), p.
014 +ppp	qqq	Prime divisor (small public modulus), q. $2^{159}<q<2^{160}$.
014 +ppp +qqq	ggg	Public key generator, g.
014 +ppp +qqq +ggg	yyy	Public key, y. $y=g^x \bmod(p)$; $1<y<p$.

DSS Private External Key Token

A DSS private external key token contains the following sections:

- A required PKA token header, starting with the token identifier X'1E'
- A required DSS private key section, starting with the section identifier X'01'
- A required DSS public key section, starting with the section identifier X'03'
- An optional private key name section, starting with the section identifier X'10'

Table 68 presents the format of a DSS private external key token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format). All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

Table 68. DSS Private External Key Token

Offset (Dec)	Number of Bytes	Description
Token Header (required)		
000	001	Token identifier. X'1E' indicates an external token. The private key is enciphered with a PKA master key.
001	001	Version, X'00'.
002	002	Length of the key token structure.
004	004	Ignored. Should be zero.
DSS Private Key Section and Secured Subsection (required)		
000	001	X'01', section identifier, DSS private key.
001	001	X'00', version.
002	002	Length of the DSS private key section, 436, X'01B4'.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.
024	004	Reserved; set to binary zero.
028	001	Key security: **X'00'** Unencrypted DSS private key subsection identifier. **X'81'** Encrypted DSS private key subsection identifier.
029	001	Padding, X'00'.
030	020	SHA-1 hash of the key token structure contents that follow the public key section. If no sections follow, this field is set to binary zeros.
050	010	Reserved; set to binary zero.
060	048	Ignored; set to binary zero.
108	128	Public key generator, g. $1<g<p$.
236	128	Prime modulus (large public modulus), p. $2^{L-1}<p<2^{L}$ and L (the modulus length) must be a multiple of 64.
364	020	Prime divisor (small public modulus), q. $2^{159}<q<2^{160}$.
384	004	Reserved; set to binary zero.
388	024	Random number, confounder. **Note:** This field and the next two fields are enciphered for key confidentiality when the key security flag (offset 28) indicates the private key is enciphered.
412	020	Secret DSS key, x; x is random. (See the preceding note.)
432	004	Random number, generated when the secret key is generated. (See the preceding note.)

Table 68. DSS Private External Key Token (continued)

Offset (Dec)	Number of Bytes	Description
DSS Public Key Section (required)		
000	001	X'03', section identifier, DSS public key.
001	001	X'00', version.
002	002	Section length, 14+yyy.
004	002	Size of p in bits. The size of p must be one of: 512, 576, 640, 704, 768, 832, 896, 960, or 1024.
006	002	Size of the p field in bytes, which is zero for a private token.
008	002	Size of the q field in bytes, which is zero for a private token.
010	002	Size of the g field in bytes, which is zero for a private token.
012	002	Size of the y field in bytes, “yyy”.
014	yyy	Public key, y. $y=g^x$ mod(p) **Note:** p, q, and y are defined in the DSS public key token.
Private Key Name (optional)		
000	001	X'10', section identifier, private key. name
001	001	X'00', version.
002	002	Section length, X'0044' (68 decimal).
004	064	Private key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.

DSS Private Internal Key Token

A DSS private internal key token contains the following sections:

- A required PKA token header, starting with the token identifier X'1F'
- A required DSS private key section, starting with the section identifier X'01'
- A required DSS public key section, starting with the section identifier X'03'
- An optional private key name section, starting with the section identifier X'10'
- A required internal information section, starting with the eyecatcher 'PKTN'

Table 69 presents the format of a DSS private internal token. All length fields are in binary. All binary fields (exponents, lengths, and so on) are stored with the high-order byte first (left, low-address, S/390 format). All binary fields (exponents, modulus, and so on) in the private sections of tokens are right-justified and padded with zeros to the left.

Table 69. DSS Private Internal Key Token

Offset (Dec)	Number of Bytes	Description
Token Header (required)		
000	001	Token identifier. X'1F' indicates an internal token. The private key is enciphered with a PKA master key.
001	001	Version, X'00'.
002	002	Length of the key token structure excluding the internal information section.
004	004	Ignored; should be zero.
DSS Private Key Section and Secured Subsection (required)		
000	001	X'01', section identifier, DSS private key.

Table 69. DSS Private Internal Key Token (continued)

Offset (Dec)	Number of Bytes	Description
001	001	X'00', version.
002	002	Length of the DSS private key section, 436, X'01B4'.
004	020	SHA-1 hash value of the private key subsection cleartext, offset 28 to the section end. This hash value is checked after an enciphered private key is deciphered for use.
024	004	Reserved; set to binary zero.
028	001	Key security: X'01' DSS private key.
029	001	Format of external key token: **X'10'** Private key generated on an ICSF host. **X'11'** External private key was specified in the clear. **X'12'** External private key was encrypted.
030	020	SHA-1 hash of the key token structure contents that follow the public key section. If no sections follow, this field is set to binary zeros.
050	010	Reserved; set to binary zero.
060	048	The OPK encrypted under a PKA master key (Signature Master Key (SMK)).
108	128	Public key generator, g. $1<g<p$.
236	128	Prime modulus (large public modulus), p. $2^{L-1}<p<2^{L}$ for $512 \leq L \leq 1024$, and L (the modulus length) must be a multiple of 64.
364	020	Prime divisor (small public modulus), q. $2^{159}<q<2^{160}$.
384	004	Reserved; set to binary zero.
388	024	Random number, confounder. **Note:** This field and the two that follow are enciphered under the OPK.
412	020	Secret DSS key, x. x is random. (See the preceding note.)
432	004	Random number, generated when the secret key is generated. (See the preceding note.)
DSS Public Key Section (required)		
000	001	X'03', section identifier, DSS public key.
001	001	X'00', version.
002	002	Section length, 14+yyy.
004	002	Size of p in bits. The size of p must be one of: 512, 576, 640, 704, 768, 832, 896, 960, or 1024.
006	002	Size of the p field in bytes, which is zero for a private token.
008	002	Size of the q field in bytes, which is zero for a private token.
010	002	Size of the g field in bytes, which is zero for a private token.
012	002	Size of the y field in bytes, “yyy”.
014	yyy	Public key, y. $y=g^{x} \bmod(p)$; **Note:** p, g, and y are defined in the DSS public key token.
Private Key Name (optional)		
000	001	X'10', section identifier, private key name.
001	001	X'00', version.
002	002	Section length, X'0044' (68 decimal).

Table 69. DSS Private Internal Key Token (continued)

Offset (Dec)	Number of Bytes	Description
004	064	Private key name (in ASCII), left-justified, padded with space characters (X'20'). An access control system can use the private key name to verify that the calling application is entitled to use the key.
Internal Information Section (required)		
000	004	Eye catcher 'PKTN'.
004	004	PKA token type. **Bit** **Meaning When Set On** **0** RSA key. **1** DSS key. **2** Private key. **3** Public key. **4** Private key name section exists.
008	004	Address of token header.
012	002	Length of internal work area.
014	002	Count of number of sections.
016	016	PKA master key hash pattern.
032	016	Reserved.

ECC Key Token Format

The following table presents the format of the ECC Key Token.

Table 70. ECC Key Token Format

Offset (Dec)	Number of Bytes	Description
Token Header		
000	001	Token identifier. **X'00'** Null token **X'1E'** External token **X'1F'** Internal token; the private key is protected by the master key
001	001	Version, X'00'.
002	002	Length of the key token structure excluding the internal information section.
004	004	Ignored; should be zero.
ECC Token Private section		
000	001	X'20', section identifier, ECC private key
001	001	X'00', version.
002	002	Section length.

Table 70. ECC Key Token Format (continued)

Offset (Dec)	Number of Bytes	Description
004	001	Wrapping Method: This value indicates the wrapping method used to protect the data in the encrypted section. It is not the method used to protect the Object Protection Key (OPK). **X'00'** Clear – section is unencrypted. **X'01'** AESKW **X'02'** CBC Wrap - Other
005	001	Hash used for Wrapping **X'01'** SHA224 **X'02'** SHA256 **X'04'** Reserved. **X'08'** Reserved
006	002	Reserved Binary Zero
008	001	Key Usage: **X'C0'** Key Agreement **X'80'** Both signature generation and key agreement **X'00'** Signature generation only **X'02'** Translate allowed The two high-order bits indicate permitted key usage in the decryption of symmetric keys and in the generation of digital signatures. The bit in the second nibble indicates if the key is translatable. A key is translatable if it can be re-encrypted from one key encrypting key to another.
009	001	Curve type: **X'00'** Prime curve **X'01'** Brainpool curve
010	001	Key Format and Security Flag. External Token: **X'40'** Unencrypted ECC private key identifier **X'42'** Encrypted ECC private key identifier Internal Token: **X'08'** Encrypted ECC private key identifier
011	001	Reserved Binary Zero

Table 70. ECC Key Token Format (continued)

Offset (Dec)	Number of Bytes	Description
012	002	Length of p in bits **X'00C0'** Prime P-192 **X'00E0'** Prime P-224 **X'0100'** Prime P-256 **X'0180'** Prime P-384 **X'0209'** Prime P-521 **X'00A0'** Brainpool p-160 **X'00C0'** Brainpool P-192 **X'00E0'** Brainpool P-224 **X'0100'** Brainpool P-256 **X'0140'** Brainpool P-320 **X'0180'** Brainpool P-384 **X'0200'** Brainpool P-512)
014	002	IBM Associated Data length. The length of this field must be greater than or equal to 16
016	008	External Token: • Unencrypted – Reserved Binary 0x'00' • Encrypted – KVP of the AESKEK Internal Token: MKVP
024	048	External Token: reserved binary zeros. Internal Token: Object Protection Key (OPK), ICV (Integrity Check value), 8 byte confounder and a 256-bit AES key used with the AESKW algorithm to encrypt the ECC private key. The OPK is encrypted by the AES master key using AESKW as well. Example format for OPK data passed to AESKW: • 8 bytes = A6A6A6A6A6A60000 • 40 bytes = Confounder(8)/Key(32)
072	002	Associated data length, aa
074	002	Length of formatted section in bytes, bb
076	aa	Associated data (See Table 71 on page 250 for the Associated Data format).
076 + aa	Start of formatted section	If this section is in the clear it contains private key d. If it is encrypted it contains the AESKW wrapped payload.
76 + aa	bb	Formatted section which includes Private key d See Table 72 on page 251 for the format of the AESKW Wrapped Payload

Table 70. ECC Key Token Format (continued)

Offset (Dec)	Number of Bytes	Description
76 + aa + bb	End of formatted section	
ECC Token Public Section		
000	001	X'21', section identifier
001	001	X'00', version.
002	002	Section length
004	004	Reserved field, binary zero
008	001	Curve type **X'00'** Prime curve **X'01'** Brainpool curve
009	001	Reserved field, binary zero
010	002	Length of p in bits: **X'00C0'** Prime P-192 **X'00E0'** Prime P-224 **X'0100'** Prime P-256 **X'0180'** Prime P-384 **X'0209'** Prime P-521 **X'00A0'** Brainpool P-160 **X'00C0'** Brainpool P-192 **X'00E0'** Brainpool P-224 **X'0100'** Brainpool P-256 **X'0140'** Brainpool P-320 **X'0180'** Brainpool P-384 **X'0200'** Brainpool P-512
012	002	This field is the length of the public key q value in bytes, the maximum value could be up to 133 bytes, cc. The value includes the key material length and one byte to indicate if the key material is compressed or uncompressed.
014	cc	Public Key , q field

Associated Data Format for ECC Token

The table below defines the associated data as it is stored in the ECC token in the clear. Associated data is data whose integrity but not confidentiality is protected by a key wrap mechanism.

Table 71. Associated Data Format for ECC Private Key Token

Offset (Dec)	Number of Bytes	Description
000	001	Associated Data Version. 0 for ECC

Table 71. Associated Data Format for ECC Private Key Token (continued)

Offset (Dec)	Number of Bytes	Description
001	001	Length of Key Label, kl
002	002	IBM Associated Data length, 16 + kl + xxx
004	002	IBM Extended Associated Data length, xxx
006	001	User Definable Associated Data length, yyy. User definable lengths are from 0 bytes to 100 bytes.
007	001	Curve Type
008	002	Length of p in bits
010	001	Usage flag
011	001	Format and Security flag
012	004	reserved
016	kl	Key Label (optional)
016 + kl	xxx	IBM Extended Associated Data
016 + kl + xxx	yyy	User-definable Associated Data

AESKW Wrapped Payload Format for ECC Private Key Token

This table defines the contents of the AESKW payload: data will be copied into this format, then encrypted with the OPK according to the AESKW specification, and the result will be stored in the encrypted data section.

Table 72. AESKW Wrapped Payload Format for ECC Private Key Token

Offset (Dec)	Number of Bytes	Description
000	006	ICV (‘A6’....)
006	001	Length of padding in bits
007	001	Length of the hash of the associated data in bytes, ii
008	004	Hash options
012	ii	Hash of Associated Data
12+ii	mm	Key data
12+ii+mm	0-7	Padding to a multiple of 8 bytes

Trusted Block Key Token

A trusted block key-token (trusted block) is an extension of CCA PKA key tokens using new section identifiers. A trusted block was introduced to CCA beginning with Release 3.25. They are an integral part of a remote key-loading process.

Trusted blocks contain various items, some of which are optional, and some of which can be present in different forms. Tokens are composed of concatenated sections that, unlike CCA PKA key tokens, occur in no prescribed order.

As with other CCA key-tokens, both internal and external forms are defined:

- An external trusted block contains a randomly generated confounder and a triple-length MAC key enciphered under a DES IMP-PKA transport key. The MAC key is used to calculate an ISO 16609 CBC mode TDES MAC of the trusted block contents. An external trusted block is created by the Trusted_Block_Create verb. This verb can:

1. Create an inactive external trusted block
2. Change an external trusted block from inactive to active

- An internal trusted block contains a confounder and triple-length MAC key enciphered under a variant of the PKA master key. The MAC key is used to calculate a TDES MAC of the trusted block contents. A PKA master key verification pattern is also included to enable determination that the proper master key is available to process the key. The Remote_Key_Export verb only operates on trusted blocks that are internal. An internal trusted block must be imported from an external trusted block that is active using the PKA_Key_Import verb.

Note: Trusted blocks do not contain a private key section.

Trusted block sections

A trusted block is a concatenation of a header followed by an unordered set of sections. The data structures of these sections are summarized in the following table:

Section	Reference	Usage
Header	Table 73 on page 254	Trusted block token header
X'11'	Table 74 on page 255	Trusted block public key
X'12'	Table 75 on page 256	Trusted block rule
X'13'	Table 82 on page 263	Trusted block name (key label)
X'14'	Table 83 on page 263	Trusted block information
X'15'	Table 87 on page 266	Trusted block application-defined data

Every trusted block starts with a token header. The first byte of the token header determines the key form:

- An external header (first byte X'1E'), created by the Trusted_Block_Create verb
- An internal header (first byte X'1F'), imported from an active external trusted block by the PKA_Key_Import verb

Following the token header of a trusted block is an unordered set of sections. A trusted block is formed by concatenating these sections to a trusted block header:

- An optional public-key section (trusted block section identifier X'11')

  The trusted block trusted RSA public-key section includes the key itself in addition to a key-usage flag. No multiple sections are allowed.
- An optional rule section (trusted block section identifier X'12')

  A trusted block may have zero or more rule sections.

  1. A trusted block with no rule sections can be used by the PKA_Key_Token_Change and PKA_Key_Import callable services. A trusted block with no rule sections can also be used by the Digital_Signature_Verify verb, provided there is an RSA public-key section that has its key-usage flag bits set to allow digital signature operations.
  2. At least one rule section is required when the Remote_Key_Export verb is used to:
     - Generate an RKX key-token
     - Export an RKX key-token
     - Export a CCA DES key-token

– Encrypt the clear generated or exported key using the provided vendor certificate

3. If a trusted block has multiple rule sections, each rule section must have a unique 8-character Rule ID.

- An optional name (key label) section (trusted block section identifier X'13')

  The trusted block name section provides a 64-byte variable to identify the trusted block, just as key labels are used to identify other CCA keys. This name, or label, enables a host access-control system such as RACF to use the name to verify that the application has authority to use the trusted block. No multiple sections are allowed.

- A required information section (trusted block section identifier X'14')

  The trusted block information section contains control and security information related to the trusted block. The information section is required while the others are optional. This section contains the cryptographic information that guarantees its integrity and binds it to the local system. No multiple sections are allowed.

- An optional application-defined data section (trusted block section identifier X'15')

  The trusted block application-defined data section can be used to include application-defined data in the trusted block. The purpose of the data in this section is defined by the application. CCA does not examine or use this data in any way. No multiple sections are allowed.

Trusted block integrity

An enciphered confounder and triple-length MAC key contained within the required information section of the trusted block is used to protect the integrity of the trusted block. The randomly generated MAC key is used to calculate an ISO 16609 CBC mode TDES MAC of the trusted block contents. Together, the MAC key and MAC value provide a way to verify that the trusted block originated from an authorized source, and binds it to the local system.

An external trusted block has its MAC key enciphered under an IMP-PKA key-encrypting key. An internal trusted block has its MAC key enciphered under a variant of the PKA master key, and the master key verification pattern is stored in the information section.

Number representation in trusted blocks

- All length fields are in binary
- All binary fields (exponents, lengths, and so forth) are stored with the high-order byte first (left, low-address, z/OS format); thus the least significant bits are to the right and preceded with zero-bits to the width of a field
- In variable-length binary fields that have an associated field-length value, leading bytes that would otherwise contain X'00' can be dropped and the field shortened to contain only the significant bits

Format of trusted block sections

At the beginning of every trusted block is a trusted block header. The header contains the following information:

- A token identifier, which specifies if the token contains an external or internal key-token
- A token version number to allow for future changes
- A length in bytes of the trusted block, including the length of the header

The trusted block header is defined in the following table:

Table 73. Trusted block header

Offset (bytes)	Length (bytes)	Description
000	001	Token identifier (a flag that indicates token type) **X'1E'** External trusted block token **X'1F'** Internal trusted block token
001	001	Token version number (X'00').
002	002	Length of the key-token structure in bytes.
004	004	Reserved, binary zero.

Note: See “Number representation in trusted blocks” on page 253.

Following the header, in no particular order, are trusted block sections. There are five different sections defined, each identified by a one-byte section identifier (X'11' - X'15'). Two of the five sections have subsections defined. A subsection is a tag-length-value (TLV) object, identified by a two-byte subsection tag.

Only sections X'12' and X'14' have subsections defined; the other sections do not. A section and its subsections, if any, are one contiguous unit of data. The subsections are concatenated to the related section, but are otherwise in no particular order. Section X'12' has five subsections defined (X'0001' - X'0005'), and section X'14' has two (X'0001' and X'0002'). Of all the subsections, only subsection X'0001' of section X'14' is required. Section X'14' is also required.

The trusted block sections and subsections are described in detail in the following sections.

Trusted block section X'11': Trusted block section X'11' contains the trusted RSA public key in addition to a key-usage flag indicating whether the public key is usable in key-management operations, digital signature operations, or both.

Section X'11' is optional. No multiple sections are allowed. It has no subsections defined.

This section is defined in the following table:

Table 74. Trusted block trusted RSA public-key section (X'11')

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: **X'11'** Trusted block trusted RSA public key
001	001	Section version number (X'00').
002	002	Section length (16+*xxx*+*yyy*).
004	002	Reserved, must be binary zero.
006	002	RSA public-key exponent field length in bytes, *xxx*.
008	002	RSA public-key modulus length in bits.
010	002	RSA public-key modulus field length in bytes, *yyy*.
012	xxx	Public-key exponent, *e* (this field length is typically 1, 3, or 64 - 512 bytes). *e* must be odd and $1 \le e < n$. (*e* is frequently valued to 3 or $2^{16}+1$ (=65537), otherwise *e* is of the same order of magnitude as the modulus). **Note:** Although the current product implementation does not generate such a public key, you can import an RSA public key having an exponent valued to two (2). Such a public key (a Rabin key) can correctly validate an ISO 9796-1 digital signature.
012+*xxx*	*yyy*	RSA public-key modulus, *n*. $n=pq$, where *p* and *q* are prime and $2^{512} \le n < 2^{4096}$. The field length is 64 - 512 bytes.
012+*xxx*+*yyy*	004	Flags: **X'00000000'** Trusted block public key can be used in digital signature operations only **X'80000000'** Trusted block public key can be used in both digital signature and key management operations **X'C0000000'** Trusted block public key can be used in key management operations only

Note: See “Number representation in trusted blocks” on page 253.

Trusted block section X'12': Trusted block section X'12' contains information that defines a rule. A trusted block may have zero or more rule sections.

1. A trusted block with no rule sections can be used by the PKA_Key_Token_Change and PKA_Key_Import callable services. A trusted block with no rule sections can be used by the Digital_Signature_Verify verb, provided there is an RSA public-key section that has its key-usage flag set to allow digital signature operations.
2. At least one rule section is required when the Remote_Key_Export verb is used to:
   - Generate an RKX key-token
   - Export an RKX key-token
   - Export a CCA DES key-token
   - Generate or export a key encrypted by a public key. The public key is contained in a vendor certificate (section X'11'), and is the root certification key for the ATM vendor. It is used to verify the digital signature on public-key certificates for specific individual ATMs.
3. If a trusted block has multiple rule sections, each rule section must have a unique 8-character Rule ID.

Section X'12' is the only section allowed to have multiple sections. Section X'12' is optional. Multiple sections are allowed.

Note: The overall length of the trusted block may not exceed its maximum size of 3500 bytes.

Five subsections (TLV objects) are defined.

This section is defined in the following table:

Table 75. Trusted block rule section (X'12')

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: **X'12'** Trusted block rule
001	001	Section version number (X'00').
002	002	Section length in bytes (20+*yyy*).
004	008	Rule ID (in ASCII). An 8-byte character string that uniquely identifies the rule within the trusted block. Valid ASCII characters are: A...Z, a...z, 0...9, - (hyphen), and _ (underscore), left justified and padded on the right with space characters.
012	004	Flags (undefined flag bits are reserved and must be zero). **X'00000000'** Generate new key **X'00000001'** Export existing key
016	001	Generated key length. Length in bytes of key to be generated when flags value (offset 012) is set to generate a new key; otherwise ignore this value. Valid values are 8, 16, or 24; return an error if not valid.
017	001	Key-check algorithm identifier (all others are reserved and must not be used): **Value** **Meaning** **X'00'** Do not compute key-check value. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to zero. **X'01'** Encrypt an 8-byte block of binary zeros with the key. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to 8. **X'02'** Compute the MDC-2 hash of the key. In a call to CSNDRKX or CSNFRKX, set the key_check_length variable to 16.
018	001	Symmetric encrypted output key format flag (all other values are reserved and must not be used). Return the indicated symmetric key-token using the *sym_encrypted_key_identifier* parameter. **Value** **Meaning** **X'00'** Return an RKX key-token encrypted under a variant of the MAC key. **Note:** This is the only key format permitted when the flags value (offset 012) is set to generate a new key. **X'01'** Return a CCA DES key-token encrypted under a transport key. **Note:** This is the only key format permitted when the flags value (offset 012) is set to export an existing key.

Table 75. Trusted block rule section (X'12') (continued)

Offset (bytes)	Length (bytes)	Description
019	001	Asymmetric encrypted output key format flag (all other values are reserved and must not be used). Return the indicated asymmetric key-token in the asym_encrypted_key variable. **Value** **Meaning** **X'00'** Do not return an asymmetric key. Set the asym_encrypted_key_length variable to zero. **X'01'** Output in PKCS1.2 format. **X'02'** Output in RSAOAEP format.
020	*yyy*	Rule section subsections (tag-length-value objects). A series of 0 - 5 objects in TLV format.

Note: See “Number representation in trusted blocks” on page 253.

Section X'12' has five rule subsections (tag-length-value objects) defined. These subsections are summarized in the following table:

Table 76. Summary of trusted block rule subsection

Rule subsection tag	TLV object	Optional or required	Comments
X'0001'	Transport key variant	Optional	Contains variant to be exclusive-ORed into the cleartext transport key.
X'0002'	Transport key rule reference	Optional; required to use an RKX key-token as a transport key	Contains the rule ID for the rule that must have been used to create the transport key.
X'0003'	Common export key parameters	Optional for key generation; required for key export of an existing key	Contains the export key and source key minimum and maximum lengths, an output key variant length and variant, a CV length, and a CV to be exclusive-ORed with the cleartext transport key to control usage of the key.
X'0004'	Source key reference	Optional; required if the source key is an RKX key-token	Contains the rule ID for the rule used to create the source key. **Note:** Include all rules that will ever be needed when a trusted block is created. A rule cannot be added to a trusted block after it has been created.
X'0005'	Export key CCA token parameters	Optional; used for export of CCA DES key tokens only	Contains mask length, mask, and CV template to limit the usage of the exported key. Also contains the template length and template which defines which source key labels are allowed. The key type of a source key input parameter can be "filtered" by using the export key CV limit mask (offset 005) and limit template (offset 005+*yyy*) in this subsection.

Note: See “Number representation in trusted blocks” on page 253.

Trusted block section X'12' subsection X'0001': Subsection X'0001' of the trusted block rule section (X'12') is the transport key variant TLV object. This subsection is optional. It contains a variant to be exclusive-ORed into the cleartext transport key.

This subsection is defined in the following table:

Table 77. Transport key variant subsection (X'0001' of trusted block rule section (X'12')

Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: **X'0001'** Transport key variant TLV object
002	002	Subsection length in bytes (8+*nnn*).
004	001	Subsection version number (X'00').
005	002	Reserved, must be binary zero.
007	001	Length of variant field in bytes (*nnn*). This length must be greater than or equal to the length of the transport key that is identified by the *transport_key_identifier* parameter. If the variant is longer than the key, truncate it on the right to the length of the key prior to use.
008	*nnn*	Transport key variant. Exclusive-OR this variant into the cleartext transport key, provided: (1) the length of the variant field value (offset 007) is not zero, and (2) the symmetric encrypted output key format flag (offset 018 in section X'12') is X'01'. **Note:** A transport key is not used when the symmetric encrypted output key is in RKX key-token format.

Note: See “Number representation in trusted blocks” on page 253.

Trusted block section X'12' subsection X'0002': Subsection X'0002' of the trusted block rule section (X'12') is the transport key rule reference TLV object. This subsection is optional. It contains the rule ID for the rule that must have been used to create the transport key. This subsection must be present to use an RKX key-token as a transport key.

This subsection is defined in the following table:

Table 78. Transport key rule reference subsection (X'0002') of trusted block rule section (X'12')

Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: **X'0002'** Transport key rule reference TLV object
002	002	Subsection length in bytes (14).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.
006	008	Rule ID. Contains the rule identifier for the rule that must have been used to create the RKX key-token used as the transport key. The Rule ID is an 8-byte string of ASCII characters, left justified and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.

Trusted block section (X'12') subsection X'0003': Subsection X'0003' of the trusted block rule section (X'12') is the common export key parameters TLV object. This subsection is optional, but is required for the key export of an existing source key (identified by the *source_key_identifier* parameter) in either RKX key-token

format or CCA DES key-token format. For new key generation, this subsection applies the output key variant to the cleartext generated key, if such an option is desired. It contains the input source key and output export key minimum and maximum lengths, an output key variant length and variant, a CV length, and a CV to be exclusive-ORed with the cleartext transport key.

This subsection is defined in the following table:

Table 79. Common export key parameters subsection (X'0003') of trusted block rule section (X'12')

Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: **X'0003'** Common export key parameters TLV object
002	002	Subsection length in bytes (12+*xxx*+*yyy*).
004	001	Subsection version number (X'00').
005	002	Reserved, must be binary zero.
007	001	Flags (must be set to binary zero).
008	001	Export key minimum length in bytes. Length must be 8, 16, or 24. Also applies to the source key.
009	001	Export key maximum length in bytes (*yyy*). Length must be 8, 16, or 24. Also applies to the source key.
010	001	Output key variant length in bytes (*xxx*). Valid values are 0 or 8 - 255. If greater than 0, the length must be at least as long as the longest key ever to be exported using this rule. If the variant is longer than the key, truncate it on the right to the length of the key prior to use. **Note:** The output key variant (offset 011) is not used if this length is zero.
011	*xxx*	Output key variant. The variant can be any value. Exclusive-OR this variant into the cleartext value of the output.
011+*xxx*	001	CV length in bytes (*yyy*). • If the length is not 0, 8, or 16, return an error. • If the length is 0, and if the source key is a CCA DES key-token, preserve the CV in the symmetric encrypted output if the output is to be in the form of a CCA DES key-token. • If a non-zero length is less than the length of the key identified by the *source_key_identifier* parameter, return an error. • If the length is 16, and if the CV (offset 012+*xxx*) is valued to 16 bytes of X'00' (ignoring the key-part bit), then: 1. Ignore all CV bit definitions 2. If CCA DES key-token format, set the flag byte of the symmetric encrypted output key to indicate a CV value is present. 3. If the source key is 8 bytes in length, do not replicate the key to 16 bytes.

Table 79. Common export key parameters subsection (X'0003') of trusted block rule section (X'12') (continued)

Offset (bytes)	Length (bytes)	Description
012+*xxx*	*yyy*	CV. Place this CV into the output exported key-token, provided that the symmetric encrypted output key format selected (offset 018 in rule section) is CCA DES key-token. • If the symmetric encrypted output key format flag (offset 018 in section X'12') indicates return an RKX key-token (X'00'), then ignore this CV. Otherwise, exclusive-OR this CV into the cleartext transport key. • Exclusive-OR the CV of the source key into the cleartext transport key if the CV length (offset 011+*xxx*) is set to 0. If a transport key to encrypt a source key has equal left and right key halves, return an error. Replicate the key halves of the key identified by the *source_key_identifier* parameter whenever all of these conditions are met: 1. The Replicate Key command (offset X'00DB') is enabled in the active role 2. The CV length (offset 011+*xxx*) is 16, and both CV halves are non-zero 3. The *source_key_identifier* parameter (contained in either a CCA DES key-token or RKX key-token) identifies an 8-byte key 4. The key-form bits (40 - 42) of this CV do not indicate a single-length key (are not set to zero) 5. Key-form bit 40 of this CV does not indicate the key is to have guaranteed unique halves (is not set to 1). **Note:** A transport key is not used when the symmetric encrypted output key is in RKX key-token format.

Note: See “Number representation in trusted blocks” on page 253.

Trusted block section X'12' subsection X'0004': Subsection X'0004' of the trusted block rule section (X'12') is the source key rule reference TLV object. This subsection is optional, but is required if using an RKX key-token as a source key (identified by *source_key_identifier* parameter). It contains the rule ID for the rule used to create the export key. If this subsection is not present, an RKX key-token format source key will not be accepted for use.

This subsection is defined in the following table:

Table 80. Source key rule reference subsection (X'0004' of trusted block rule section (X'12')

Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: **X'0004'** Source key rule reference TLV object
002	002	Subsection length in bytes (14).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.
006	008	Rule ID. Rule identifier for the rule that must have been used to create the source key. The Rule ID is an 8-byte string of ASCII characters, left justified and padded on the right with space characters. Acceptable characters are A...Z, a...z, 0...9, - (X'2D'), and _ (X'5F'). All other characters are reserved for future use.

Note: See “Number representation in trusted blocks” on page 253.

Trusted block section X'12' subsection X'0005': Subsection X'0005' of the trusted block rule section (X'12') is the export key CCA token parameters TLV object. This subsection is optional. It contains a mask length, mask, and template for the export key CV limit. It also contains the template length and template for the source key label. When using a CCA DES key-token as a source key input parameter, its key type can be "filtered" by using the export key CV limit mask (offset 005) and limit template (offset 005+*yyy*) in this subsection.

This subsection is defined in the following table:

Table 81. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12')

Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: **X'0005'** Export key CCA token parameters TLV object
002	002	Subsection length in bytes (10+*yyy*+*yyy*+*zzz*).
004	001	Subsection version number (X'00').
005	002	Reserved, must be binary zero.
007	001	Flags (must be set to binary zero).
008	001	Export key CV limit mask length in bytes (*yyy*). Do not use CV limits if this CV limit mask length (*yyy*) is zero. Use CV limits if *yyy* is non-zero, in which case *yyy*: • Must be 8 or 16 • Must not be less than the export key minimum length (offset 008 in subsection X'0003') • Must be equal in length to the actual source key length of the key **Example**: An export key minimum length of 16 and an export key CV limit mask length of 8 returns an error.

Table 81. Export key CCA token parameters subsection (X'0005') of trusted block rule section (X'12') (continued)

Offset (bytes)	Length (bytes)	Description
009	*yyy*	Export key CV limit mask (does not exist if *yyy*=0). Indicates which CV bits to check against the source key CV limit template (offset 009+*yyy*). **Examples**: A mask of X'FF' means check all bits in a byte. A mask of X'FE' ignores the parity bit in a byte.
009+*yyy*	*yyy*	Export key CV limit template (does not exist if *yyy*=0). Specifies the required values for those CV bits that are checked based on the export key CV limit mask (offset 009). The export key CV limit mask and template have the same length, *yyy*. This is because these two variables work together to restrict the acceptable CVs for CCA DES key tokens to be exported. The checks work as follows: 1. If the length of the key to be exported is less than *yyy*, return an error 2. Logical AND the CV for the key to be exported with the export key CV limit mask 3. Compare the result to the export key CV limit template 4. Return an error if the comparison is not equal **Examples**: An export key CV limit mask of X'FF' for CV byte 1 (key type) along with an export key CV limit template of X'3F' (key type CVARENC) for byte 1 filters out all key types except CVARENC keys. **Note:** Using the mask and template to permit multiple key types is possible, but cannot consistently be achieved with one rule section. For example, setting bit 10 to 1 in the mask and the template permits PIN processing keys and cryptographic variable encrypting keys, and only those keys. However, a mask to permit PIN-processing keys and key-encrypting keys, and only those keys, is not possible. In this case, multiple rule sections are required, one to permit PIN-processing keys and the other to permit key-encrypting keys.
009+*yyy*+*yyy*	001	Source key label template length in bytes (*zzz*). Valid values are 0 and 64. Return an error if the length is 64 and a source key label is not provided.
010+*yyy*+*yyy*	*zzz*	Source key label template (does not exist if *zzz*=0). If a key label is identified by the *source_key_identifier* parameter, verify that the key label name matches this template. If the comparison fails, return an error. The source key label template must conform to the following rules: • The key label template must be 64 bytes in length • The first character cannot be in the range X'00' - X'1F', nor can it be X'FF' • The first character cannot be numeric (X'30' - X'39') • A key label name is terminated by a space character (X'20') on the right and must be padded on the right with space characters • The only special characters permitted are #, $, @, and * (X'23', X'24', X'40', and X'2A') • The wildcard X'2A' (*) is only permitted as the first character, the last character, or the only character in the template • Only alphanumeric characters (a...z, A...Z, 0...9), the four special characters (X'23', X'24', X'40', and X'2A'), and the space character (X'20') are allowed

Note: See “Number representation in trusted blocks” on page 253.

Trusted block section X'13': Trusted block section X'13' contains the name (key label). The trusted block name section provides a 64-byte variable to identify the trusted block, just as key labels are used to identify other CCA keys. This name, or label, enables a host access-control system such as RACF to use the name to verify that the application has authority to use the trusted block.

Section X'13' is optional. No multiple sections are allowed. It has no subsections defined. This section is defined in the following table:

Table 82. Trusted block key label (name) section X'13'

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: **X'13'** Trusted block name (key label)
001	001	Section version number (X'00').
002	002	Section length in bytes (68).
004	064	Name (key label).

Note: See “Number representation in trusted blocks” on page 253.

Trusted block section X'14': Trusted block section X'14' contains control and security information related to the trusted block. This information section is separate from the public key and other sections because this section is required while the others are optional. This section contains the cryptographic information that guarantees its integrity and binds it to the local system.

Section X'14' is required. No multiple sections are allowed. Two subsections are defined. This section is defined in the following table:

Table 83. Trusted block information section X'14'

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: **X'14'** Trusted block information
001	001	Section version number (X'00').
002	002	Section length in bytes (10+*xxx*).
004	002	Reserved, binary zero.
006	004	Flags: **X'00000000'** Trusted block is in the inactive state **X'00000001'** Trusted block is in the active state
010	*xxx*	Information section subsections (tag-length-value objects). One or two objects in TLV format.

Note: See “Number representation in trusted blocks” on page 253.

Section X'14' has two information subsections (tag-length-value objects) defined. These subsections are summarized in the following table:

Table 84. Summary of trusted block information subsections

Rule subsection tag	TLV object	Optional or required	Comments
X'0001'	Protection information	Required	Contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key, the ISO 16609 TDES CBC MAC value, and the MKVP of the PKA master key (computed using MDC4).
X'0002'	Activation and expiration dates	Optional	Contains flags indicating whether or not the coprocessor is to validate dates, and contains the activation and expiration dates that are considered valid for the trusted block.

Note: See “Number representation in trusted blocks” on page 253.

Trusted block section X'14' subsection X'0001': Subsection X'0001' of the trusted block information section (X'14') is the protection information TLV object. This subsection is required. It contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key, the ISO-16609 TDES CBC MAC value, and the MKVP of the PKA master key (computed using MDC4).

This subsection is defined in the following table:

Table 85. Protection information subsection (X'0001') of trusted block information section (X'14')

Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: **X'0001'** Trusted block information TLV object
002	002	Subsection length in bytes (62).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.
006	032	Encrypted MAC key. Contains the encrypted 8-byte confounder and triple-length (24-byte) MAC key in the following format: **Offset** **Description** **00 - 07** Confounder **08 - 15** Left key **16 - 23** Middle key **24 - 31** Right key
038	008	MAC. Contains the ISO-16609 TDES CBC message authentication code value.
046	016	MKVP. Contains the PKA master key verification pattern, computed using MDC4, when the trusted block is in internal form, otherwise contains binary zero.

Note: See “Number representation in trusted blocks” on page 253.

Trusted block section X'14' subsection X'0002': Subsection X'0002' of the trusted block information section (X'14') is the activation and expiration dates TLV object. This subsection is optional. It contains flags indicating whether or not the coprocessor is to validate dates, and contains the activation and expiration dates that are considered valid for the trusted block.

This subsection is defined in the following table:

Table 86. Activation and expiration dates subsection (X'0002') of trusted block information section (X'14')

Offset (bytes)	Length (bytes)	Description
000	002	Subsection tag: **X'0002'** Activation and expiration dates TLV object
002	002	Subsection length in bytes (16).
004	001	Subsection version number (X'00').
005	001	Reserved, must be binary zero.
006	002	Flags: **X'0000'** The coprocessor does not check dates. **X'0001'** The coprocessor checks dates. Compare the activation date (offset 008) and the expiration date (offset 012) to the coprocessor's internal real-time clock. Return an error if the coprocessor date is before the activation date or after the expiration date.
008	004	Activation date. Contains the first date that the trusted block can be used for generating or exporting keys. Format of the date is YYMD, where: **YY** Big-endian year (return an error if greater than 9999) **M** Month (return an error if any value other than X'01' - X'0C') **D** Day of month (return an error if any value other than X'01' - X'1F'; day must be valid for given month and year, including leap years) Return an error if the activation date is after the expiration date or is not valid.
012	004	Expiration date. Contains the last date that the trusted block can be used. Same format as activation date (offset 008). Return an error if date is not valid.

Note: See “Number representation in trusted blocks” on page 253.

Trusted block section X'15': Trusted block section X'15' contains application-defined data. The trusted block application-defined data section can be used to include application-defined data in the trusted block. The purpose of the data in this section is defined by the application; it is neither examined nor used by CCA in any way.

Section X'15' is optional. No multiple sections are allowed. It has no subsections defined. This section is defined in the following table:

Table 87. Trusted block application-defined data section X'15'

Offset (bytes)	Length (bytes)	Description
000	001	Section identifier: **X'15'** Application-defined data
001	001	Section version number (X'00').
002	002	Section length (6+*xxx*)
004	002	Application data length (*xxx*) The value of *xxx* can be from 0 bytes to a length that does not cause the trusted block to exceed its maximum size of 3500 bytes.
006	*xxx*	Application-defined data May be used to hold a public-key certificate for the trusted public key.

Note: See “Number representation in trusted blocks” on page 253.

Data Areas

These topics present the format of the Cryptographic Communication Vector Table (CCVT) and the Cryptographic Communication Vector Table Extension (CCVE) data areas.

The Cryptographic Communication Vector Table (CCVT)

The CCVT is the ICSF base control block and contains addresses of common areas for use by ICSF components. Indicators in the CCVT also provide ICSF status information. The CCVT is getmained in subpool 245 under the line. The ICSF CCVT is anchored off of SCVTCCVT in the SCVT macro.

Programming Interface information

CCVT

ONLY these fields are part of the programming interface:

- CCVTDACC
- CCVTCCVE
- CCVTHFLG
- CCVTSFLG
- CCVTPRPC
- CCVTINST
- CCVTINS2
- CCVTLNTH
- CCVTFMID
- CCVT_USERPARM

End of Programming Interface information

Table 88 on page 267 describes the contents of the Cryptographic Communication Vector Table. Any bits that are not described in the table are reserved.

Table 88. Cryptographic Communication Vector Table

Offset (Dec)	Number of Bytes	Field Name	Description
0	4	CCVTID	EBCDIC Cryptographic Communication Vector Table ID. This field must contain the character string `CCVT`.
4	2	CCVTVER	Version. The version of the CCVT. This field must contain the character string `04`
6	2	CCVTLEN	The length of the CCVT.
8	6	CCVTAUX	Auxilliary flags. **Bit** — **Meaning When Set On** **0** — ICSF is terminating. **1** — ICSF is abnormally terminating. **2** — ICSF initialization is performing master key validation. **3** — Coprocessor request interrupts enabled.
14	2	CCVTRLVL	ICSF level.
16	4	CCVTCCVE	Cryptographic Communication Vector Table Extension (CCVE) address. The address of a private area extension of the CCVT. You should place any fields not needed by other address spaces in the CCVE.
20	4	CCVT_CKT_BTMB	Address of anchor for CKT.
24	4	CCVTPC2	PC number for entry into module CSFASSPC.
28	4	CCVTPRPC	Entry point for the pre-PC processing module, CSFARPC.
32	4	CCVTINST	For installation use.
36	1	CCVTSFG1	Status byte. **Bit** — **Meaning When Set On** **0** — ICSF services are active. **1** — At least one Integrated Cryptographic Feature has a valid master key. **2** — ICSF initialization complete. **3** — ICSF is active and PCF is not active. **4** — Compatibility is permitted. COMPAT(YES) or COMPAT(COEXIST) is specified. **5** — At least one Integrated Cryptographic Feature is valid. **7** — Always set to 1.

Table 88. Cryptographic Communication Vector Table (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
37	1	CCVTFLAG	Flag byte. **Bit** — **Meaning When Set On** **3** — Cryptographic coprocessor hardware instructions available. **4** — At least one cryptographic coprocessor is active. **5** — Coprocessor request interrupt support active. **6** — At least one cryptographic coprocessor is online **7** — At least one coprocessor or accelerator is present.
38	1	CCVTOFLG	Operational flag byte. **Bit** — **Meaning When Set On** **0** — Configuration is under PR/SM. **2** — CKDS key record create, key record delete, and key record write disallowed. **3** — CKDS I/O subtask is available. **4** — CCVT_DEF_ALG bit. If on, CDMF is the system default algorithm; if off, DES is the default. **5** — CCVT_CDMF_ENA bit. If on, hardware is capable of performing CDMF. **6** — PKA master keys are valid. **7** — Use ICSF reason codes.
39	1	CCVTSVCM	SVC number for key management. This is the PCF compatibility SVC.
40	1		Reserved
41	1	CCVTSVCS	SVC number for DES interface SVC. This is the PCF compatibility SVC.
42	2	CCVTASID	ASID of ICSF address space.
44	4	CcvtPcGrs	Entry point to CSFMIDGR.
48	4	CCVTPC3	Entry point to CSFASSPA used by compatibility SVCs.
52	4	CCVTSRUT	Address of the access method module.
56	8	CCVTINS2	An 8-byte area for installation use.
64	4	CCVTMDS	Data space server PC. PC number for entry to data space server that adds and deletes the in-storage CKDS.
68	4	CCVTLNTH	Maximum installation data length.
72	4	CCVTASCB	ICSF ASCB address.
76	4		Reserved

Table 88. Cryptographic Communication Vector Table (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
80	1	CCVTHFLG	Flag bytes. **Bit** — **Meaning When Set On** **0** — Crypto assist instructions available. **1** — Additional secure Crypto device available. **2** — Support for 64-bit callers. **3** — ICSF Cross-System Services environment is active for CKDS **4** — ICSF Cross-System Services environment is active for TKDS **5** — RSA 4096-bit function enabled and the RNGL service is available **6** — Secure key AES is available **7** — AES master key is active
81	1	CCVTSFLG	Flag bytes. **Bit** — **Meaning When Set On** **0** — ICSF during initialization. **1** — ICSF was able to complete cleanup, so no EOM cleanup is needed. **2** — PKCS #11 operating in FIPS standard mode. **3** — PKCS #11 operating in FIPS compatibility mode.
82	1	CCVT1FLG	Flag byte. **Bit** — **Meaning When Set On** **0** — ECC master key is active / ECC Secure Key Functions available **1** — ECC clear key functions are supported **2** — AES KEKs, TR-31, and ECC Diffie-Hellman are supported. **3** — Dynamic RSA master key change enabled.
83	1		Reserved
84	4	CCVTENF	ECB for ENF listen.
88	4	CCVTTCB	ICSF maintask TCB address.
92	4	CCVTTRC	ECB for component trace.
96	4		Reserved
100	4	CCVT_ABTERM_ECB	Abnormal termination ECB.
104	8	CCVT_CKDS_FIXED	Offsets and lengths for fixed length CKDS.
112	8	CCVT_CKDS_VAR	Offsets and lengths for variable length CKDS.
120	4	CCVTLFDE	ECB to start the “look for disabled crypto” task.

Table 88. Cryptographic Communication Vector Table (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
124	4	CCVTIOSE	ECB to post to use I/O subtask.
128	4	CCVTPCTRP	PC for CSFKSTRP entry.
132	4	CCVT_ACT_DURING_TERM	Activity count during term.
136	8	CCVTFMID	ICSF FMID.
144	8	CCVT_USERPARM	ICSF user parameter.
152	1	CCVTPKAF	PKA register clear key entry processing flags. **Bit** — **Meaning When Set On** **0** — KMMK is valid for CP0. **1** — SMK is valid for CP0. **2** — KMMK has been reset for CP0. **3** — SMK has been reset for CP0. **4** — KMMK is valid for CP1. **5** — SMK is valid for CP1. **6** — KMMK has been reset for CP1. **7** — SMK has been reset for CP1.
153	1	CCVTPKAR	**Bit** — **Meaning When Set On** **0 and 1** — SMK status for KSU0. **2 and 3** — KMMK status for KSU0. **4 and 5** — SMK status for KSU1. **6 and 7** — KMMK status for KSU1.
154	1	CCVTPKAX	PKA register status (reserved).
155	1	CCVTPKAZ	PKA register status (reserved).
156	16	CCVTCCC	Cryptographic configuration control (CCC).
172	4	CCVTSPKB	Address of public key build.
176	4	CCVTSPKX	Address of public key extract.
180	4	CCVTPIOE	ECB for PKDS I/O subtask.
184	8		Reserved
192	4	CCVTGiveAway	Recovery token for cell pools.
196	1	CCVTPKDF	PKDS processing flags. **Bit** — **Meaning When Set On** **0** — PKDS available. **2** — At least one PCICA is active. **3** — ICSF Cross-System Services environment is active for PKDS.
197	1	CCVTCICS	CICS processing flags. **Bit** — **Meaning When Set On** **0** — CSFVCCPP installed. **1** — CSFACKWL installed.

Table 88. Cryptographic Communication Vector Table (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
198	1	CCVTYAFF	**Bit** **Meaning When Set On** **0** ZKA compliance environment.
199	1	CSFTTKDF	TKDS processing flags **Bit** **Meaning When Set On** **0** TKDS available
200	4	CCVTPRPD	Address of CSFVCCPP.
204	4	CCVTCKWL	Address of CSFVCCKW.
208	12	CcvtSdtTcb	Address of CSFMISDT TCBs.
220	4	CCVTENFP	ECB for PCI Cryptographic Coprocessor online event.
224	6	CcvtSdtAsid	Asids owning SYSZxKT
230	10		Reserved.
240	4	CCVTPC6	PC6 (CSFMWCFS entry).
244	16	CCVT_KXMD	Hardware feature status. **Bit** **Meaning When Set On** **1** SHA-1 enabled. **2** SHA-256 enabled. **3** SHA-512 enabled. Bytes 2–16 are reserved.
260	4		Reserved
264	4		Reserved
268	4	CCVTCSVG	Address of CSFSCVG.
272	4		Reserved
276	4	CCVTDACC	ICSF DAC instructions control block for RMF.
280	16	CCVT_KMC_EXPORT	Hardware feature status. **Bit** **Meaning When Set On** **1** KMC DES enabled. **3** KMC TDES enabled. **9** KMC encrypted DES enabled. **11** KMC encrypted TDES enabled. **18** KMC AES 128 key enabled. **19** KMC AES 192 key enabled. **20** KMC AES 256 key enabled. **26** KMC encrypted AES-128 enabled **27** KMC encrypted AES-192 enabled. **28** KMC encrypted AES-256 enabled Change bytes 5-16 are reserved.
296	4	CCVTPC7	PC7 (CSFMGARM entry)
300	4	CCVTPC8	PC8 (CSFMGTRM entry)

Table 88. Cryptographic Communication Vector Table (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
304	8	CCVTGART	Token of CSFMGARC resource manager
312	8	CCVTGTRT	Token of CSFMGTRC resource manager
320	4	CCVTGARC	Address of CSFMGARC resource manager
324	4	CCVTGTRC	Address of CSFMGTRC resource manager
328	4	CCVT_IDENTITY	Identifier
332	4		Reserved
336	8	CCVT_PSMID	Last used PSMID
344	4	CCVTEPRP	Address of CSFVCPC6
348	4	CCVTPKB6	Address of CSFSPKB6
352	4	CCVTVRET	Address of CSFVRET
356	4	CCVTWRET	Address of CSFWRET
360	4	CCVTSRET	Address of CCVTSRET
364	4	CCVTGSRET	Address of CCVTGSRET
368	4	CCVTCVG6	Address of CCVTCVG6
372	4	Ccvt_CKDS_PREPMSG_ORIGIN	SYSid of XCF message originator for prepare message for CKDS
376	4	Ccvt_PKDS_PREPMSG_ORIGIN	SYSid of XCF message originator for prepare message for PKDS
380	4	Ccvt_TKDS_PREPMSG_ORIGIN	SYSid of XCF message originator for prepare message for TKDS
384	4	CCVTTIOE	ECB for TKDS I/O subtask
388	4	CCVT_CKDS_VALUES_ACTIVE	Address of CCVT_CKDS_VALUES structures for the active CKDS
392	32		Reserved
424	8	CCVTTDS	Definition space information (TKDS)
432	20		Reserved
452	2	CCVT_PKDS_MAXLRECL	Maximum logical record length for PKDS records
454	2		Reserved
456	8	CCVT_PKDS_DS	Current PKDS data space
464	16		Reserved
480	4	CCVTPUPD_ECB	ECB to post for PKDS update
484	44	CCVT_PKDSN	PKDS data set name
528	4	CCVTNAMES	Address of CSFNAMES
532	4	CCVTSNAMES	Address of CCVTSNAMES
536	4	CCVTGNAMES	Address of CCVTGNAMES
540	4	Ccvt_MTLen	Size of module table.
544	4	Ccvt_MTSP	Module table subpool Id
548	48	Ccvt_Mt	Module Table
596	4	CCVTRNAMES	Address of CSFRNAMES
600	8		Reserved

Table 88. Cryptographic Communication Vector Table (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
608	32	Ccvt_ModuleAddrs	Module address.
640	8		Reserved
648	4	CCVTCSS	Address of CKDS Cross-System Services block.
652	4	CCVTCSST	Address of TKDS Cross-System Services block.
656	4	CCVTCSSP	Address of PKDS Cross-System Services block .
660	32	CCVT_Keep	Area preserved across a restart of ICSF.
692	8	CCVTGIRT	Token of CSFMGIRT resource manager.
700	4	CCVTSMF82_14Ctr	counter to control writing of SMF records.
704	44	CCVT_CKDSN	CKDS data set name currently in use .
748	44	CCVT_TKDSN	TKDS data set name currently in use.
792	24		Reserved
816	0		

The Cryptographic Communication Vector Table Extension (CCVE)

The CCVE is an extension of the CCVT that contains fields that can exist. The CCVE exists in ICSF extended private. It should contain any ICSF base control block fields that are not needed by other address spaces.

Programming Interface information

CCVE

ONLY these fields are part of the programming interface:

- CCVEINPP
- CCVEINPL
- CCVESECC

End of Programming Interface information

Table 89 describes the contents of the Cryptographic Communication Vector Table Extension. Any bits that are not described in the table are reserved.

Table 89. Cryptographic Communication Vector Table Extension

Offset (Dec)	Number of Bytes	Field Name	Description
0	4	CCVEID	Cryptographic Communication Vector Table Extension ID. This field must contain the character string `CCVE`.
4	2	CCVEVER	Version. The version number of the CCVE. This field must contain the character string `04`.
6	2	CCVELEN	The length of the CCVE.
8	8		Reserved.

Table 89. Cryptographic Communication Vector Table Extension (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
16	4	CCVESTAT	Status word First status byte – CCVESTA1 **Bit** **Meaning When Set On** **0** Special secure mode allowed. **1** Special secure mode enabled. **3** Authentication required for key retrieval. **4** The hardware has gone from active to inactive. **5** First start of ICSF during this IPL. **6** Security Server (RACF) checking required for authorized callers. **7** PCF coexistence. Second status byte – CCVESTA2 **0** Dynamic CKDS updates disallowed. **1** PKA callable services disabled from panel. **2** Dynamic PKDS updates disabled from panel. **3** Include CKT in dump of ICSF private space. **6** PKA callable services disallowed. **7** Authenticate the CKT when bit is one. Third status byte – CCVESTA3 **1** PKDS write, create, and delete not permitted. **2** SYSPLEXCKDS(YES) was specified in Install Options Data Set. **3** SYSPLEXCKDS(YES,FAIL(YES)) was specified in Install Options Data Set. **4** SYSPLEXTKDS(YES) was specified in Install Options Data Set. **5** SYSPLEXTKDS(YES,FAIL(YES)) was specified in Install Options Data Set. **6** TKDS refresh requested. **7** TKDS empty at initialization Fourth status byte – CCVESTA4 **Bit** **Meaning When Set On** **0** PKDS dataspace needs refresh. **1** PKDS dataspace can't be updated. **2** Reserved **3** Reserved **4** SYSPLEXPKDS(YES) **5** SYSPLEXPKDS(YES,FAIL(YES)) **6** CKDS MAC record authentication **7** Sysplex running in sysplex mode (not XCF-local mode)
20	4	CCVECAMQ	Pointer to MCAMQ.
24	4	CCVEEXIT	Pointer to the installation exit router (CSFEXIT).
28	4	CCVECLIC	Software Crypto control block

Table 89. Cryptographic Communication Vector Table Extension (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
32	4	CCVE_ENQ_TIMEOUT	XCF Failure detection interval in 0.01 seconds used for Sysplex ENQ timeout interval.
36	4	CCVETRCB	Pointer to the current trace buffer. **Bit** **Meaning When Set On** **0** Trace is active.
40	4	CCVECPRM	Address of CPRM.
44	4	CCVEMGST	Address of the generic service table. See “Generic Service Table (CSFMGST)” on page 278 for a description of the generic service table.
48	4	CCVEENT	Address of the exit name table.
52	4	CCVETSKT	Address of task table.
56	4	CCVEMKVN	Master key version numbers. Byte 1: Current master key version number. Bytes 2 and 3: Reserved. Byte 4: Cryptographic domain index.
60	54	CCVEWLDS	Dataset name of WaitList dataset.
114	1	CCVEIBMR	IBM reserved byte.
115	1	CCVEHFL2	Hardware flags **Bit** **Meaning When Set On** **0** CCA level 3.41 detected **1** CCA level 4.00 detected **2** Reserved **3** AP-special-command facility available **4** AP 4096-bit ME facility available **5** AP 4096-bit CRT facility available
116	4	CCVE_EXTRAFALGS	Status word. **Bit** **Meaning When Set On** **0** The default wrapping for internal tokens is enhanced. **1** The default wrapping for external tokens is enhanced.
120	4	CCVE_NOPKA_MSGID	WTO message ID saved when PKA callable services are not available at startup
124	12	CCVEDCTLARR	DCTL address array.
136	4	CCVESERBCPID	SERB cell pool ID
140	4	CCVEFIXS	Address of the fixed area storage used as dynamic storage for the RISGNL routines.
144	4	CCVEFIXL	Length of the fixed area storage.
148	4	CCVECPUF	CPUF routine — used to manipulate the control register.

Table 89. Cryptographic Communication Vector Table Extension (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
152	4	CCVERFMK	RFOMK routine — used to RFOMK keys on specific CPs.
156	4	CCVERMKV	MKV RISGNL routine — used by MKV to validate a CP.
160	4	CCVESTHW	STHW routine — used to obtain the current status of the hardware.
164	4	CCVEKEYM	KEYM routine — used to manipulate keys from the key entry hardware.
168	4	CCVEDKEF	DKEF routine — used to manipulate keys for clear key entry.
172	16	CCVE_PKA_KMMK_HP	KMMK hash pattern
188	16	CCVE_PKA_SMK_HP	SMK hash pattern
204	4	CCVELFDD	ECB for look for disabled Cryptographic Coprocessor Feature task termination (LFD Done).
208	4	CCVELFDT	Pointer to TCB for CSFMLFDT.
212	4	CCVEENFS	ECB for *Issue ENF SIGNAL*.
216	4	CVESMCA	Address of SMCA
220	4	CCVE_SUBPOOL	Subpool for storage
224	4	CCVE_SRRW_EXIT	Single read/write exit addr
228	4	CCVEMKVB	Pointer to the current Master Key Verification Pattern (MKVP) block. See “DES Master Key Verification Pattern Block (MKVB)” on page 278 for a description of the MKVP block.
232	32	CCVEMKB1	First MKVP block.
264	32	CCVEMKB2	Second MKVP block.
296	32	CCVEMKB3	Third MKVP block.
328	4	CCVEINPP	Pointer to installation optional parameter.
332	4	CCVEINPL	Length of the installation optional parameter.
336	4	CCVETRCN	Number of trace entries.
340	4	CCVEIOPB_PKDS	Address of PKDS IO subtask data.
344	4	CCVEIOST_TKDS	Address of TKDS IO subtask TCB.
348	4	CCVEIOPB_TKDS	Address of TKDS IO subtask data.
352	4	CCVEIOPB	Address of IO subtask data.
356	4	CCVECCPD	Pointer to CAJP Data.
360	4	CCVECCPV	Pointer to private CAJP Data .
364	4	CCVEWKAR	Work area for services.
368	4	CCVEMUST	Address of UDX service table.
372	8	CCVESECC	Reserved for security exit.
380	4	CCVEENTK	ENTE for security keys exit.
384	4	CCVEENTS	ENTE for security service exit.
388	4	CCVEMIQIH	Address of interrupt handler
392	4	CCVE_TKE_KEY_CACHE@	Address of TKE key cache

Table 89. Cryptographic Communication Vector Table Extension (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
396	4	CCVEDSCB	Control block for the data manager.
400	12	CCVE_CKDS_HASH_TABLES	CKDS hash tables.
412	12	CCVE_PKDS_HASH_TABLES	PKDS hash tables.
424	4	CCVE_KEY_ STORE_POLICY	**Bit** — **Meaning When Set On** **0** — CKDS key store policy enabled **1** — CKDS control in fail mode **2** — CKDS control in warn mode **3** — CKDS default control enabled **4** — No duplicates in CKDS **8** — PKDS key store policy enabled **9** — PKDS control in fail mode **10** — PKDS control in warn mode **11** — PKDS default control enabled **12** — No duplicates in PKDS **16** — Granular keylabel access controls enabled in fail mode **17** — Granular keylabel access controls enabled in warn mode **18** — Enhanced export restrictions enabled for AES keys **19** — Enhanced export restrictions enabled for DES keys **24** — PKA key extensions enabled. **25** — PKCS #11 Token used for trusted certificate repository (SAF keyring when this bit is 0). **26** — PKA key extensions in WARNONLY mode.
428	4	CCVE_PLEX_SYSID	System sysplex token
432	4	CCVEINQKP_ECB	INQKP ECB for waking up
436	4	CCVE_KSP_PKAKE_DATA_PTR	Address of PKAKE data
440	1	CCVE_FIPS	FIPS policy flags. **Bit** — **Meaning When Set On** **1** — FIPS startup known answer tests failed disabling PKCS#11. **2** — FIPSMODE(xxx,FAIL(YES)) specified **3** — Known answer test executed on accelerator for private key operation **4** — Known answer test executed on accelerator for public key operation

Table 89. Cryptographic Communication Vector Table Extension (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
441	3		Reserved.
444	8	CCVE_ECC_MKVP	ECC MK verification pattern This field will contain zeros unless the ECC MK is valid.
452	16	CCVE_KMF_QUERY	Results of CPACF KMF-Query
468	16	CCVE_KMCTR_QUERY	Results of CPACF KMCTR-Query
484	16	CCVE_KMO_QUERY	Results of CPACF KMO-Query
492	8	CCVE_AES_MKVP	AES MK verification pattern.
500	8	CCVE_DES_MKVP	DES MK verification pattern
508	32	CCVE_KDS_MKVPS	MKVPs from key data sets
540	4	Ccve_MaxSys	Maximum number of systems possible in sysplex
544	4	CCVEMWT_EBC	ECB to attach CSFPLMWT
548	4		reserved
552	4	CCVE_ABTERM_EBC	ECB to terminate ICSF
556	4	CCVE_HCHK_PTR	Pointer to Health Check blocks
560	28		reserved

DES Master Key Verification Pattern Block (MKVB)

Table 90 describes the contents of the MKVB.

Table 90. DES Master Key Verification Pattern Block Format

Offset (Dec)	Number of Bytes	Description
0	4	Pointer to the next element or zero.
4	4	Pointer to the next element — this field for use by CSFMMKV.
8	4	Reserved.
12	1	DES master key version number for this verification pattern.
13	1	Flag. **Bit** **Meaning When Set On** **0** This element is on the active queue.
14	2	Reserved.
16	8	DES Master Key Verification Pattern.
24	8	DES Master Key Authentication Pattern.

Generic Service Table (CSFMGST)

Table 91 on page 279 describes the format of the generic service table, a control block that is used to control the call of installation-defined services.

Table 91. Generic Service Table Block Format

Offset (Dec)	Number of Bytes	Description
0	4	EBCDIC ID.
4	2	Version number.
6	2	Length of the MGST.
8	4	Number of entries in the array.
12	4	Subpool this table is in.
16	4	Reserved.
20	4	Reserved.
24	4	Reserved.
28	4	Reserved.
Variable Section of the MGST		
32	8	IBM-assigned name.
40	8	Installation-assigned name.
48	4	Flags. **Bit** — **Meaning When Set On** **0** — Service has been requested by the installation. **1** — Service has been loaded. **2** — Service is active. **3** — Service is required.
52	4	Address of the service.
56	4	Installation-assigned service number.
60	4	Reserved.

RMF Measurements Table

Table 92 describes the contents of the performance measurements for RMF. The count fields are double-word length.

Table 92. RMF Measurements Record Format

Offset (Dec)	Number of Bytes	Field Name	Description
0	4	DACC_ID	The DACC ID.
4	4	DACC_VER	The version.
8	4	DACC_LEN	The control block length.
12	4	DACC_ENT_CNT	Number of entries.
16	4	DACC_ENT_LEN	Length of each entry.
20	8	DACC_ENT_ID	Identifier of count array - character 'ENCSDES'. The Encipher service will collect data as follows: • Collection for single DES is done separately. The number of service calls, number of bytes of data enciphered, and the number of hardware instructions used to encipher the data will be collected.
28	8	DACC_ENT_SVC_CNT	Count of ENCSDES service calls.

Table 92. RMF Measurements Record Format (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
36	8	DACC_ENT_BYT_CNT	Count of ENCSDES bytes processed.
44	8	DACC_ENT_INT_CNT	Count of ENCSDES instructions.
52	8	DACC_ENT_ID	Identifier of count array - character 'ENCTDES'. The Encipher service will collect data as follows: • Double and triple DES will be counted together. The number of service calls, number of bytes of data enciphered, and the number of hardware instructions used to encipher the data will be collected.
60	8	DACC_ENT_SVC_CNT	Count of ENCTDES service calls.
68	8	DACC_ENT_BYT_CNT	Count of ENCTDES bytes processed.
76	8	DACC_ENT_INT_CNT	Count of ENCTDES instructions.
84	8	DACC_ENT_ID	Identifier of count array - character 'DECSDES'. The Decipher service will collect data as follows: • Collection for single DES is done separately. The number of service calls, number of bytes of data deciphered, and the number of hardware instructions used to decipher the data will be collected.
92	8	DACC_ENT_SVC_CNT	Count of DECSDES service calls.
100	8	DACC_ENT_BYT_CNT	Count of DECSDES bytes processed.
108	8	DACC_ENT_INT_CNT	Count of DECSDES instructions.
116	8	DACC_ENT_ID	Identifier of count array - character 'DECTDES'. The Decipher service will collect data as follows: • Double and triple DES will be counted together. The number of service calls, number of bytes of data deciphered, and the number of hardware instructions used to decipher the data will be collected.
124	8	DACC_ENT_SVC_CNT	Count of DECTDES service calls.
132	8	DACC_ENT_BYT_CNT	Count of DECTDES bytes processed.
140	8	DACC_ENT_INT_CNT	Count of DECTDES instructions.
148	8	DACC_ENT_ID	Identifier of count array - character 'MACGEN'. The MAC Generate service will collect data as follows: • Single and various double key MAC will be gathered together. The number of service calls, number of bytes of data MAC'd, and the number of instructions will be collected.
156	8	DACC_ENT_SVC_CNT	Count of MACGEN service calls.
164	8	DACC_ENT_BYT_CNT	Count of MACGEN bytes processed.
172	8	DACC_ENT_INT_CNT	Count of MACGEN instructions.
180	8	DACC_ENT_ID	Identifier of count array - character 'MACVER'. The MAC Verify service will collect data as follows: • Single and various double key MAC will be gathered together. The number of service calls, number of bytes of data MAC'd, and the number of instructions will be collected.
188	8	DACC_ENT_SVC_CNT	Count of MACVER service calls.
196	8	DACC_ENT_BYT_CNT	Count of MACVER bytes processed.

Table 92. RMF Measurements Record Format (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
204	8	DACC_ENT_INT_CNT	Count of MACVER instructions.
212	8	DACC_ENT_ID	Identifier of count array - character 'OWH'. The One Way Hash service will collect data as follows: • For SHA-1, the number of service calls, number of bytes of bytes of data hashed, and the number of instructions will be collected.
220	8	DACC_ENT_SVC_CNT	Count of OWH service calls.
228	8	DACC_ENT_BYT_CNT	Count of OWH bytes processed.
236	8	DACC_ENT_INT_CNT	Count of OWH instructions.
244	8	DACC_ENT_ID	Identifier of count array - character 'PTR'. The PIN Translate service will collect data as follows: • Collect the number of service calls only.
252	8	DACC_ENT_SVC_CNT	Count of PTR service calls.
260	16		Reserved.
276	8	DACC_ENT_ID	Identifier of count array - character 'PVR'. The PIN Verify service will collect data as follows: • Collect the number of service calls only.
284	8	DACC_ENT_SVC_CNT	Count of PVR service calls.
292	16		Reserved.
308	8	DACC_ENT_ID	Identifier of count array - character 'OWH256'. The One Way Hash service will collect data as follows: • For SHA-224 and SHA-256, the number of service calls, number of bytes of data hashed, and the number of instructions will be collected.
316	8	DACC_ENT_SVC_CNT	Count of OWH service calls for SHA-224 and SHA-256.
324	8	DACC_ENT_BYT_CNT	Count of OWH bytes processed for SHA-224 and SHA-256.
332	8	DACC_ENT_INT_CNT	Count of OWH instructions for SHA-224 and SHA-256.
340	8	DACC_ENT_ID	Identifier of count array - character 'OWH512'. The One Way Hash service will collect data as follows: • For SHA-384 and SHA-512, the number of service calls, number of bytes of data hashed, and the number of instructions will be collected.
348	8	DACC_ENT_SVC_CNT	Count of OWH service calls for SHA-384 and SHA-512.
356	8	DACC_ENT_BYT_CNT	Count of OWH bytes processed for SHA-384 and SHA-512.
364	8	DACC_ENT_INT_CNT	Count of OWH instructions for SHA-384 and SHA-512.
372	8	DACC_ENT_ID	Identifier of count array - character ‘ENCAES’. The Symmetric algorithm encipher service will collect data as follows: The number of service calls, number of bytes of data enciphered, and the number of instructions used to encipher the data will be collected.
380	8	DACC_ENT_SVC_CNT	Count of SAE service calls
388	8	DACC_ENT_BYT_CNT	Count of ENCAES bytes processed
396	8	DACC_ENT_INT_CNT	Count of ENCAES instruction

Table 92. RMF Measurements Record Format (continued)

Offset (Dec)	Number of Bytes	Field Name	Description
404	8	DACC_ENT_ID	Identifier of count array - character 'DECAES'. The Symmetric algorithm decipher service will collect data as follows: the number of service calls, number of bytes of data deciphered, and the number of instructions used to decipher the data will be collected.
412	8	DACC_ENT_SVC_CNT	Count of SAD service calls
420	8	DACC_ENT_BYT_CNT	Count of DECAES bytes processed
428	8	DACC_ENT_INT_CNT	Count of DECAES instruction

Appendix B. ICSF SMF Records

SMF records are documented in *z/OS MVS System Management Facilities (SMF)* and published on release boundaries of z/OS. As a migration aid for ICSF Web Deliverables, which are often made available between releases or z/OS, the ICSF SMF records are also documented here.

Record Type 82 (52) — ICSF Record

Record type 82 is used to record information about the events and operations of the Integrated Cryptographic Service Facility (ICSF) program product. Record type 82 is written to the SMF data set at the completion of certain cryptographic functions:

- **Subtype 1** — is written whenever ICSF is started.
- **Subtype 3** — is written whenever there is a change in the number of available processors with the cryptographic feature
- **Subtype 4** — is written whenever ICSF handles error conditions for cryptographic feature failure (CC3, Reason Code 1) or cryptographic tampering (CC3 Reason Code 3).
- **Subtype 5** — is written whenever a change to special security mode is detected.
- **Subtype 6 and 7** — are written whenever a key part is entered via the key entry unit (KEU).
- **Subtype 8** — is written whenever the in-storage copy of the CKDS is refreshed.
- **Subtype 9** — is written whenever the CKDS is updated by a dynamic CKDS update service.
- **Subtype 10** — is written when a clear key part is entered for one of the PKA master keys.
- **Subtype 11** — is written when a clear key part is entered for the DES master key.
- **Subtype 12** — is written for each request and reply from calls to the CSFSPKSC service by TKE.
- **Subtype 13** — is written whenever the PKDS is updated by a dynamic PKDS update service.
- **Subtype 14** — is written when a clear key part is entered for any of the PCI Cryptographic Coprocessor master keys.
- **Subtype 15** — is written whenever a PCI Cryptographic Coprocessor retained key is created or deleted.
- **Subtype 16** — is written for each request and reply from calls to the CSFPCI service by TKE.
- **Subtype 17** — is written periodically to provide some indication of PCI Cryptographic Coprocessor usage.
- **Subtype 18** — is written when a PCI Cryptographic Coprocessor, PCI Cryptographic Accelerator, PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express2 Accelerator comes online or offline.
- **Subtype 19** — is written when a PCI X Cryptographic Coprocessor operation begins or ends.
- **Subtype 20** — is written by ICSF to record processing times for PCIXCCs and CEX2Cs.
- **Subtype 21** — is written when ICSF issues IXCJOIN to join the ICSF sysplex group or issues IXCLEAVE to leave the sysplex group.

- **Subtype 22** — is written when the Trusted Block Create Callable services are invoked.
- **Subtype 23** — is written when the token data set (TKDS) is updated
- **Subtype 24** — is written when duplicate tokens are found.
- **Subtype 25** — is written when the key store policy is activated.
- **Subtype 26** — is written when the public key data set is refreshed.
- **Subtype 27** — is written for information about PKA Key Management Extensions.
- **Subtype 28** — is written for information about High Performance Encrypted Key.
- **Subtype 29** — is written for each TKE workstation audit record received from a TKE workstation.

Macro to Symbolically Address Record Type 82: The SMF record mapping macro for ICSF type 82 record is CSFSMF82.

The mapping macro, CSFSMF82, resides in SYS1.MACLIB.

Record Environment

The following conditions exist for the generation of each of the subtypes of this record:

Macro

Subtype	Macro
1	SMFWTM (record exit: IEFU83)
3,4,5,6,7,8	SMFEWTM,BRANCH=YES,MODE=XMEM (record exit: IEFU85)

Record Mapping

Header/Self-defining Section

This section contains the common SMF record headers fields and the triplet fields (offset/length/number), if applicable, that locate the other sections on the record.

Offsets		Name	Length	Format	Description
0	0	SMF82LEN	2	binary	Record length. This field and the next field (total of four bytes) form the RDW (record descriptor word).
2	2	SMF82SEG	2	binary	Segment descriptor (see record length field).
4	4	SMF82FLG	1	binary	System indicator: **Bit** **Meaning When Set** **0-2** Reserved **3-6** Version indicators **7** Reserved.
5	5	SMF82RTY	1	binary	Record type 82 (X'52').
6	6	SMF82TME	4	binary	Time since midnight, in hundredths of a second, that the record was moved into the SMF buffer.
10	A	SMF82DTE	4	packed	Date when the record was moved into the SMF buffer, in the form 0*cyydddF*.
14	E	SMF82SID	4	EBCDIC	System identification (from the SID parameter).
18	12	SMF82SSI	4	EBCDIC	Subsystem identification.
22	16	SMF82STY	2	binary	Record subtype.

Server User or End User Audit Section

Provides server user or end user audit information when the subtype is one that logs state changes. When auditing information is supplied, there will be a server user section and, optionally, an end user section. The SMF82AUD_HDR_NUM_SECTIONS field of the Auditing Header section will indicate whether only a server user section is provided, or if an end user section is also provided. If both a server user section and an end user section are present, they can appear in either order.

Table 93. SMF type 82 server user or end user audit section

Offsets		Name	Length	Format	Description
0	0	SMF82AUD_SECTION_TYPE	4	EBCDIC	Type of the section that follows. Either: • 'SERV' (for server user) • 'USER' (for end user)
4	4	SMF82AUD_SECTION_NUM_FLDS	2	binary	Number of triples in this section
6	6	SMF82AUD_SECTION_TOTAL_LEN	2	binary	Overall length of this section, including this header
8	8	Tag-Length-Value (TLV) triplets start here and are defined in Table 94. These repeat as many times as the SMF82AUD_SECTION_NUM_FLDS field indicates.			

Each Tag-Length-Value (TLV) triplet is a structure called SMF82AUD_TRIPLET and is defined as follows. The values for the tags and the format and maximum length of the data are defined in Table 95.

Table 94. Tag-Length-Value (TLV) triplet structure (SMF82AUD_TRIPLET)

Offsets		Name	Length	Format	Description
0	0	SMF82AUD_TRIPL_TAG	2	binary	Tag of the information in this TLV
2	2	SMF82AUD_TRIPL_LENGTH	2	binary	Length of this TLV including these first two fixed fields
4	4	SMF82AUD_TRIPL_DATA	*	varies	Data for this TLV

The tag values and their corresponding information are described in the following table. The tag value is defined in the constant SMF82AUD_TAG_*xxx* and the maximum length in SMF82AUD_MAXLEN_*xxx*. For example, the tag for X500_IDN is SMF82AUD_TAG_X500_IDN and maximum length of the associated data is SMF82AUD_MAXLEN_X500_IDN.

Table 95. TLV triplet tag values

Tag Value		Name	Length	Format	Description
1	1	X500_IDN	0-255	EBCDIC	X.500 Certificate Issuer's Distinguished Name (ACEEX5PR->IDN)
2	2	X500_SDN	0-255	EBCDIC	X.500 Certificate Subject's Distinguished Name (ACEEX5PR->SDN)
10	A	IDID_USRI	1-246	UTF-8	X.500 Distinguished Name of distributed client end user (ACEEIDID-> IDID1UDN)
11	B	IDID_USRF	1	binary	Format of IDID_USRI (ACEEIDID->IDID1NMF) **0** Undetermined **1** Straight string **2** X.500 format
12	C	IDID_REG	1-255	UTF-8	Name of the registry that authenticated the user (ACEEIDID->IDID1RN)
14	E	USRI	8	EBCDIC	RACF user ID (ACEEUSRI)

Table 95. TLV triplet tag values (continued)

Tag	Value	Name	Length	Format	Description
15	F	GRPN	8	EBCDIC	Connect group (ACEEGRPN)
16	10	TRM_USER	8	EBCDIC	Terminal ID (ACEETRM)
17	11	JOB_JBN	8	EBCDIC	Job name (JMRJOB)
18	12	JOB_RST	4	binary	Job entry time (JMRENTRY) in hundredths of a second that the reader recognized the JOB statement for this job. This field can be zero.
26	1A	JOB_RSD	4	binary	Job entry date (JMREDATE) that the reader recognized the JOB statement for this job in the form 0CYYDDDF. This field can be zero.
34	22	JOB_UID	8	binary	User-defined identification field (JMRUSEID)
42	2A	SEC	8	EBCDIC	Security label (TOKSCL)

Subtype 1

Initialization Section

Offsets		Name	Length	Format	Description
0	0	SMF82INI	4	binary	Cryptographic communication vector table extension (CCVE) status bits **Bit** **Meaning When Set** **0** Special security mode allowed **1** Special security mode enabled **2** Reserved **3** Key authentication **4-5** Reserved **6** RACF checking **7-14** Reserved **15** CKT authentication **16** Default wrapping for internal tokens is the enhanced method **17** Default wrapping for external tokens is the enhanced method **18-31** Reserved
4	4	SMF82VTS	1	binary	Cryptographic communication vector table (CCVT) status bits **Bit** **Meaning When Set** **0-3** Reserved **4** Compatible with CUSP and PCF **5-7** Reserved.
5	5	SMF82IDO	1	binary	Current crypto domain index.
6	6		2		Reserved.
8	8	SMF82ITE	4	binary	Number of trace entries.
12	C	SMF82CKD	44	EBCDIC	Name of the cryptographic key data set (CKDS) that was read into storage.
56	38	SMF82IML	4	binary	Maximum length for data.
60	3C	SMF82USR	8	EBCDIC	USERPARM specifies installation use in the installation options data set.
68	44	SMF82PKD	44	EBCDIC	PKDS name.
112	70	SMF82TKS	44	EBCDIC	TKDS name.

Subtype 3

Status Change Section

Offsets		Name	Length	Format	Description
0	0	SMF82SNS	4	binary	Number of sections following.
4	4	SMF82SPR	4	binary	Processor number.
8	8	SMF82KSU	4	binary	Key storage unit (KSU) number.
12	C	SMF82SDX	4	binary	Current crypto domain index.
16	10	SMF82VER	4	binary	Current master key version.
20	14	SMF82SSW	4	binary	Zero, if no error condition exists with the processor. Otherwise, the ICSF status word.
24	18	SMF82STI	4	binary	**Bit** **Meaning When Set** **0** Current master key verification pattern valid **1** New master key authentication pattern valid **2** New master key verification pattern valid **3** Old master key verification pattern valid **4-31** Reserved.
28	1C	SMF82CVP	8	EBCDIC	Current master key verification pattern.
36	24	SMF82NAP	8	EBCDIC	New master key authentication pattern.
44	2C	SMF82NVP	8	EBCDIC	New master key verification pattern.
52	34	SMF82OVP	8	EBCDIC	Old master key verification pattern.

Subtype 4

Condition Code Three Section

Offsets		Name	Length	Format	Description
0	0	SMF823SW	4	binary	Status word from CC3.
4	4	SMF823PR	1	binary	Processor number.
5	5	SMF823DX	1	binary	Current crypto domain index.
6	6		2		Reserved.

Subtype 5

Special Security Mode Section

Offsets		Name	Length	Format	Description
0	0	SMF82SSB	8	binary	Special security mode (SSM) bits **Bit** **Meaning When Set** **0** SSM mode is enabled **1-63** Reserved.

Subtype 6

Master Key Part Section

Offsets		Name	Length	Format	Description
0	0	SMF82MKB	4	binary	Master key part (MKPART) bits **Bit** **Meaning When Set** **0** New master key verification pattern valid **1** Old master key verification pattern valid **2-31** Reserved.
4	4	SMF82NMV	8	EBCDIC	New master key verification pattern.
12	C	SMF82OMV	8	EBCDIC	Verification pattern for the key part.
20	14	SMF82MKS	1	binary	KSU number.
21	15	SMF82MDX	1	binary	Current crypto domain index.
22	16		2		Reserved.

Subtype 7

KEU Key Part Entry Section

Offsets		Name	Length	Format	Description
0	0	SMF82KPB	4	binary	Key part (KPART) bits **Bit** **Meaning When Set** **0** Key part verification pattern valid. **1** Coprocessor is a PCIXCC. **2** Coprocessor is a CEX2C. **3** Coprocessor is a CEX3C. **4-31** Reserved.
4	4	SMF82KV	8	EBCDIC	Key part verification pattern.
12	C	SMF82KKS	1	binary	KSU number.
13	D	SMF82KDX	1	binary	Current crypto domain index.
14	E		2		Reserved.
16	10	SMF82KCK	44	EBCDIC	Name of the CKDS containing the key part.
60	3C	SMF82KCL	72	EBCDIC	CKDS entry being modified.

Subtype 8

Cryptographic Key Data Set Refresh Section

Offsets		Name	Length	Format	Description
0	0	SMF82ROC	44	EBCDIC	Name of the CKDS being replaced.
44	2C	SMF82RNC	44	EBCDIC	Name of the CKDS to replace the current CKDS.

Subtype 9

Dynamic CKDS Update

Offsets		Name	Length	Format	Description
0	0	SMF82UCB	4	binary	Update CKDS bits **Bit** **Meaning When Set** **0** CKDS record added **1** CKDS record changes **2** CKDS record deleted **3-31** Reserved.
4	4	SMF82UCN	44	EBCDIC	CKDS name.
48	30	SMF82UCL	72	EBCDIC	CKDS entry being modified.

Subtype 10

PKA Key Part Entry

Offsets		Name	Length	Format	Description
0	0	SMF82PKB	4	binary	PKA part bits **Bit** **Meaning When Set** **0** Key management master key processed **1** Signature master key processed **2** Key part hash pattern valid **3** Master key hash pattern valid **4-31** Reserved.
4	4	SMF82PHP	16	EBCDIC	Master key hash pattern.
20	14	SMF82KPH	16	EBCDIC	Key part hash pattern.
36	24	SMF82PKS	1	binary	KSU number.
37	25	SMF82PKX	1	binary	Current crypto domain index.
38	26		2		Reserved.

Subtype 11

Clear New Master Key Part Entry

Offsets		Name	Length	Format	Description
0	0	SMF82CMB	4	binary	Clear new master key bits **Bit** **Meaning When Set** **0** Clear new master key hash pattern valid **1** Clear new master key verification pattern valid **2** Clear new key part hash pattern valid **3** Clear new key part verification pattern valid.
4	4	SMF82CHP	16	EBCDIC	Clear new master key hash pattern.
20	14	SMF82CNP	8	EBCDIC	Clear new master key verification pattern.
28	1C	SMF82CPH	16	EBCDIC	Key part hash pattern.
44	2C	SMF82CPV	8	EBCDIC	Key part verification pattern.
52	34	SMF82CKS	1	binary	KSU number.
53	35	SMF82CDX	1	binary	Current crypto domain index.
54	36		2		Reserved.

Subtype 12

PKSC Commands

Offsets		Name	Length	Format	Description
0	0	SMF82PSQ	1024	EBCDIC	Request.
1024	400	SMF82PSP	1024	EBCDIC	Response.

Subtype 13

Dynamic PKDS Update

Offsets		Name	Length	Format	Description
0	0	SMF_PKDS_BITS	4	binary	Update PKDS bits **Bit** **Meaning When Set** **0** PKDS record added **1** PKDS record changed **2** PKDS record deleted **3-31** Reserved.
4	4	SMF_PKDS_NAME	44	EBCDIC	PKDS name.
48	30	SMF_PKDS_KEY_LABEL	72	EBCDIC	PKDS entry being modified.

Subtype 14

PCI Cryptographic Coprocessor Master Key Entry

Offsets		Name	Length	Format	Description
0	0	SMF82AAB	4	binary	Flag bytes **Bit** **Meaning When Set** **0** DES NMK verification pattern is valid. **1** RSA NMK verification pattern is valid. **2** DES Key key part verification pattern is valid. **3** RSA Key Key part verification pattern is valid. **4** AES NMK verification pattern is valid. **5** AES key part verification pattern is valid. **6** ECC NMK verification pattern is valid. **7** ECC key part verification pattern is valid. **8** Coprocessor is not a PCI Cryptographic Coprocessor. **9** Coprocessor is a PCI X Cryptographic Coprocessor. **10** Coprocessor is a CEX2C. **11** Coprocessor is a CEX3C. **12-31** Reserved.
4	4	SMF82ANV	16	EBCDIC	New master key register verification pattern.
20	14	SMF82AKV	16	EBCDIC	Key part verification pattern.
36	24	SMF82APN	1	binary	PCI Cryptographic Processor number.
37	25	SMF82ASN	8	EBCDIC	PCI Cryptographic Processor serial number.
45	2D	SMF82ADM	1	binary	PCI Cryptographic Coprocessor domain.
46	2E		2		Reserved.

Subtype 15

PCI Cryptographic Coprocessor Master Key Entry

Offsets		Name	Length	Format	Description
0	0	SMF82RKF	4	binary	First flag byte **Bit** **Meaning When Set** **0** Retained key created. **1** Retained key deleted on coprocessor. **2** Retained key deleted from PKDS. **3-7** Reserved. **8** Coprocessor is *not* a PCI Cryptographic Coprocessor. **9** Coprocessor is a PCI X Cryptographic Coprocessor. **10** Coprocessor is a CEX2C. **11** Coprocessor is a CEX3C. **12-31** Reserved.
4	4	SMF82RKN	64	EBCDIC	Label of Retained private key.
68	44	SMF82RKP	1	binary	PCI Cryptographic Coprocessor number.
69	45	SMF82RKS	8	EBCDIC	PCI Cryptographic Coprocessor serial number.
77	4D	SMF82RDM	1	binary	PCI Cryptographic Coprocessor domain.
78	4E		2		Reserved.

Subtype 16

PCI Cryptographic Coprocessor TKE

Offsets		Name	Length	Format	Description
0	0	SMF82PFL	4	binary	Flag bytes **Bit** **Meaning When Set** **0** Request command. **1** Reply response. **2-7** Reserved. **8** Coprocessor is *not* a PCI Cryptographic Coprocessor. **9** Coprocessor is a PCI X Cryptographic Coprocessor. **10** Coprocessor is a CEX2C. **11** Coprocessor is a CEX3C. **12-31** Reserved.
4	4	SMF82PPN	1	binary	PCI Cryptographic Coprocessor number.
5	5	SMF82PSN	8	EBCDIC	PCI Cryptographic Coprocessor serial number.
13	D	SMF82PDM	1	binary	PCI Cryptographic Coprocessor domain.
14	E		2		Reserved.
16	10	SMF82PBL	4	binary	Parameter block length, "xxx".
20	14	SMF82PDL	4	binary	Parameter data block length, "yyy".
24	18	SMF82PBK			Parameter block of length "xxx" followed by parameter data block of length "yyy".

Subtype 17

PCI Cryptographic Coprocessor Timing

Offsets		Name	Length	Format	Description
0	0	SMF82CTN	8	EBCDIC	Time just before the PCI Cryptographic Coprocessor operation begins. This is in time-of-day (TOD) format.
8	8	SMF82CTD	8	EBCDIC	Time just after PCI Cryptographic Coprocessor operation ends. This is in time-of-day (TOD) format.
16	10	SMF82CTW	8	EBCDIC	Time just after results have been communicated to caller address space. This is in time-of-day (TOD) format.
20	14	SMF82CTQ	4	binary	Number of processes waiting to submit work to the same PCI Cryptographic Coprocessor and domain, using the same reference number.
24	18	SMF82CTF	2	EBCDIC	Function code of service.
26	1A	SMF82CTX	1	binary	PCI Cryptographic Coprocessor number.
27	1B	SMF82CTS	8	EBCDIC	PCI Cryptographic Coprocessor serial number.
35	23	SMF82CTM	1	binary	PCI Cryptographic Coprocessor domain.
36	24	SMF82CTR	1	binary	PCI Cryptographic Coprocessor reference number.
37	25		3		Reserved.

Subtype 18

Cryptographic Coprocessor Configuration

Offsets		Name	Length	Format	Description
0	0	SMF82CGB	4	binary	Flag bytes **Bit** **Meaning When Set** **0** A Cryptographic Coprocessor has been brought online. **1** A Cryptographic Coprocessor has been taken offline. **2-7** Reserved. **8** Coprocessor is *not* a PCI Cryptographic Coprocessor. **9** Coprocessor is a PCI X Cryptographic Coprocessor. **10** Coprocessor is a CEX2C. **11** Coprocessor is a CEX2A. **12** Coprocessor is a CEX3C. **13** Coprocessor is a CEX3A. **14–31** Reserved.
4	4	SMF82CGX	1	binary	Cryptographic Coprocessor number.
5	5	SMF82CGS	8	EBCDIC	Cryptographic Coprocessor serial number.
13	D		3		Reserved.

Subtype 19

PCI X Cryptographic Coprocessor Timing

Offsets		Name	Length	Format	Description
0	0	SMF82XTN	8	EBCDIC	Time just before the PCI X Cryptographic Coprocessor operation begins.
8	8	SMF82XTD	8	EBCDIC	Time just after PCI X Cryptographic Coprocessor operation ends.

Offsets		Name	Length	Format	Description
16	10	SMF82XTW	8	EBCDIC	Time just after results have been communicated to caller address space.
24	18	SMF82XTQ	4	binary	Number of processes waiting to submit work to the same PCI X Cryptographic Coprocessor and domain, using the same reference number.
28	1C	SMF82XTF	2	EBCDIC	Function code of service.
30	1E	SMF82XTX	1	binary	PCI X Cryptographic Coprocessor number.
31	1F	SMF82XTS	8	EBCDIC	PCI X Cryptographic Coprocessor serial number.
39	27	SMF82XTM	1	binary	PCI X Cryptographic Coprocessor domain.
40	28	SMF82XTR	1	binary	PCI X Cryptographic Coprocessor reference number.
41	29		3		Reserved.

Subtype 20

Cryptographic Coprocessor Processing Times

Offsets		Name	Length	Format	Description
0	0	SMF82TFL	4	binary	Flag bytes **Bit** **Meaning When Set** **0** Coprocessor is a PCI X Cryptographic Coprocessor. **1** Coprocessor is a CEX2C. **2** Coprocessor is a CEX2A. **3** Coprocessor is a CEX3C. **4** Coprocessor is a CEX3A. **5–31** Reserved.
4	4	SMF82TNQ	8	EBCDIC	Coprocessor time before NQAP.
12	C	SMF82TDQ	8	EBCDIC	Coprocessor time after DQAP.
20	14	SMF82TWT	8	EBCDIC	Coprocessor time after WAIT.
28	1C	SMF82TQU	4	binary	Coprocessor queue length.
32	20	SMF82TSF	2	EBCDIC	Coprocessor sub function code.
34	22	SMF82TIX	1	binary	Coprocessor index.
35	23	SMF82TSN	8	EBCDIC	Coprocessor serial number.
43	2B	SMF82TDM	1	binary	Domain.
44	2C	SMF82TRN	1	binary	Reference number.
45	2D		3		Reserved.

Subtype 21

ICSF Sysplex Group Change Section

Offsets		Name	Length	Format	Description
0	0	SMF82SXG	8	EBCDIC	Name of ICSF Sysplex group.
8	8	SMF82SXM	8	EBCDIC	Name of sysplex member.
16	F	SMF82SXA	1	binary	ICSF Sysplex member status flags **Bit** **Meaning When Set** **0** Member joined the ICSF sysplex group. **1** Member left the ICSF sysplex group. **2–7** Reserved.

Offsets		Name	Length	Format	Description
17	11	SMF82SXR	1	binary	ICSF Sysplex member conditions of status flags **Bit** **Meaning When Set** **0** Member joined or left the ICSF sysplex due to normal initialization/termination processing **1** Member left the ICSF sysplex due to error **2–7** Reserved.
18	12		2		Reserved.
20	14	SMF82SXT	8	EBCDIC	Time of ICSF sysplex join/leave index.
28	1C	SMF82SXC	44	EBCDIC	Name of active CKDS.

Subtype 22

Trusted Block Create Callable Services Section

Offsets		Name	Length	Format	Description
0	0	SMF82TBF	4	binary	Process Flag bytes **Bit** **Meaning When Set** **0** Created Inactive Trusted Block. **1** Activate an Inactive Block. **2** Trusted Block has Public Key. **3–31** Reserved.
4	4	SMF82TBS	2	EBCDIC	ASID of caller.
6	6	SMF82TBN	64	EBCDIC	Label of Input Trusted Block.
70	46	SMF82TBO	64	EBCDIC	Label of Output Trusted Block.
134	86	SMF82TBX	64	EBCDI	Label of Transport Key.

Subtype 23

Token Data Set Update

Offsets		Name	Length	Format	Description
0	0	SMF82TKF	4	binary	TKDS bits **Bit** **Meaning When Set** **0** TKDS record added **1** TKDS record changed **2** TKDS record deleted **3–31** Reserved.
4	4	SMF82TKN	44	EBCDIC	TKDS name
48	30	SMF82TKH	44	EBCDIC	TKDS handle being processed

Subtype 24

Duplicate Tokens Found

Offsets		Name	Length	Format	Description
0	0	SMF82DCNTSTRT	4	binary	Start of duplicate labels.
4	4	SMF82DCNTEND	4	binary	End of duplicate labels.
8	8	SMF82DCNT	4	binary	Number of duplicate labels.
12	C	SMF82DRSVD	4	binary	Reserved.
16	10	SMF82DNAM	44	binary	Name of key data set.

Offsets		Name	Length	Format	Description
The following field is repeated *count* (SMF82DCNT) number of times.					
60	3C	SMF82_Label	64	EBCDIC	A key label.

Subtype 25

Key Store Policy

The key store policy must be activated before this SMF record subtype is logged. The subtype is logged when the callable service request meets the following requirements:

- The key store policy allows the request to complete with a warning.
- The key store policy indicates that the request should complete with a failure. The "warning" flag is not set in the failure case.

Offsets		Name	Length	Format	Description
0	0	SMF82KDS	44	EBCDIC	Data set name.
44	2C	SMF82KLF	4	binary	Key store policy flags: **Bit** **Meaning When Set** **0** Warning. **1** List is incomplete. **2** List is from CKDS. **3** List is from PKDS. **4-31** Reserved.
48	30	SMF82KLC	4	binary	Number of key labels following.
The following field is repeated *count* (SMF82KLC) number of times.					
52	34	SMF82DKL	72	EBCDIC	Unauthorized duplicate key label and key type.

Subtype 26

Public Key Data Set Refresh

Offsets		Name	Length	Format	Description
0	0	SMF82PREF_FLAG	4	binary	Flags: **Bit** **Meaning When Set** **0** Data space was refreshed. **1-31** Reserved.
4	4	SMF82_PREF_OLDDS	44	EBCDIC	Old PKDS Name.
48	30	SMF82_PREF_NEWDS	44	EBCDIC	New PKDS Name.

Subtype 27

PKA Key Management Extensions

Offsets		Name	Length	Format	Description
24	18	SMF82PKE_FLAGS	4	binary	PKA Key Management Extension flags: **Bit** **Meaning When Set** **0** PKA token may not be used for requested function. **1** SYM token may not be exported by the provided PKA token. **2** PKA label list is imcomplete. **3** SYM label list is incomplete. **24** Trusted certificate repository has changed. **25** PKA Key Management Extensions in WARNONLY mode. **26** An error was detected during processing. **27** Trusted cert repository was empty. **28** An error was detected while extracting APPLDATA. **29** The repository wasn't found. **30** One or more certs couldn't be parsed. Bits 0-3 are set during callable services. Bits 24-30 are set during repository parsing. Bits 4-23 and 31 are reserved.
28	1C	SMF82PKE_FUNCTION	8	EBCDIC	Name of the service that issued this SMF record. The name is in the form CSFzzz.
36	24	SMF82PKE_APPLDATALEN	1	binary	Length of the enablement profile APPLDATA or current repository name.
37	25	SMF82PKE_APPLDATA	247	EBCDIC	Enablement profile APPLDATA or current repository name.
284	11C	SMF82PKE_FUNCSPEC	0	binary	Function-specific section of the record.
284	11C	SMF82PKE_APPLDATA_PARSING	0	binary	APPLDATA parsing results section.
284	11C	SMF82PKE_SAF_RC	2	binary	SAF_RC or 'FFFF'X.
286	11E	SMF82PKE_SERV_RC	2	binary	RACF RC or ICSF RC.
288	120	SMF82PKE_SERV_RS	4	binary	RACF RS or ICSF RS.
284	11C	SMF82PKE_SERVICE_SECTION	0	binary	Callable services section.
284	11C	SMF82PKE_PKA_REC_CNT	4	binary	Number of PKA labels present in this record.
288	120	SMF82PKE_SYM_REC_CNT	4	binary	Number of SYM labels present in this record.
The following is repeated SMF82PKE_PKA_REC_CNT number of times.					
292	124	SMF82PKE_PKA_LABELS	64	EBCDIC	PKA key label.
The following is repeated SMF82PKE_SYM_REC_CNT number of times.					
292+ zzz	124+ zzz	SMF82PKE_SYM_LABELS	72	EBCDIC	SYM key label.

Subtype 28

High Performance Encrypted Key

Offsets		Name	Length	Format	Description
24	18	SMF82HPSK_FLAGS	4	binary	High Performance Encrypted Key flags: **Bit** **Meaning When Set** **0** Rewrapping operation is not permitted for this symmetric key. **1** Rewrapping operation was permitted for this symmetric key. **2** The list of labels is incomplete. Bits 3–31 are reserved.
28	1C	SMF82HPSK_FUNCTION	8	EBCDIC	Name of the service that issues this SMF record. The name is in the form of CSFzzzz.
36	24	SMF82HPSK_SYM_LABEL_CNT	4	binary	Number of SYM labels present in this record.
The following is repeated SMF82HPSK_SYM_LABEL_CNT number of times.					
40	28	SMF82HPSK_SYM_LABELS	72	EBCDIC	SYM key label and type.

Subtype 29

TKE Workstation Audit Record

Offsets		Name	Length	Format	Description
24	18	SMF82TKEAR_FLAGS	4	binary	Flags -- reserved
28	1C	SMF82TKEAR_NAMELEN	2	binary	TKE workstation name length
30	24	SMF82TKEAR_RCDLEN	2	binary	TKE audit record data length
32	20	SMF82TKEAR_NAME	VAR	EBCDIC	TKE workstation name
VAR	VAR		VAR	EBCDIC	TKE audit record data

Appendix C. CICS-ICSF Attachment Facility

The purpose of the CICS-ICSF Attachment Facility is to enhance the performance of CICS transactions in the same region as a transaction using long-running ICSF services such as the PKA services and CKDS or PKDS update services. You must have CICS Transaction Server for z/OS Version 3.1 or higher.

Without the CICS-ICSF Attachment Facility, the application that requests a long-running ICSF service is placed into an OS WAIT. With the CICS-ICSF Attachment Facility, a long running service is transferred to an L8, and the CICS application is placed into a CICS WAIT, rather than an OS WAIT, for the duration of the operation.

Installing the CICS-ICSF Attachment Facility

Before you can use the CICS-ICSF Attachment Facility, the ICSF system programmer, or the CICS administrator needs to install it. This involves:

- Relinking the ICSF enabling routine, CSFATREN, and the ICSF TRUE, CSFATRUE, if ICSF was previously installed in an environment without the CICS-ICSF Attachment Facility
- Installing the proper load libraries in the PROC used to start CICS
- Updating the CICS System Definitions (CSD) data set to define the programs to CICS
- Enabling these programs

For information about CICS TRUE programs, refer to *CICS Customization Guide*, SC33-1683.

Steps for installing the CICS-ICSF attachment facility

1. If ICSF was previously installed in an environment without the CICS-ICSF Attachment Facility (i.e., without being linked with the CICS SDFHLOAD data set), the ICSF system programmer will need to relink the ICSF TRUE, CSFATRUE, and the ICSF enabling routine, CSFATREN. This would be the case if, for example, (a) the DDDEF entries for ICSF do not have the SDFHLOAD DDDEF pointing to the CICS SDFHLOAD data set but instead have it pointing to an empty data set, or (b) z/OS (and hence ICSF) was installed using a ServerPac.

   To relink the ICSF modules, first manually update the ICSF DDDEF for SDFHLOAD to point to the CICS SDFHLOAD data set. (Refer to ICSF sample CSFDDDEF shipped in SAMPLIB.) Then submit a job to relink the ICSF modules. This is an example of job control language for the relink.

```
//STEP01        EXEC PGM=IEWL,
//   PARM='LIST,XREF,LET,DCBS,AMODE(31),RMODE(24)'
//SYSLMOD   DD DISP=SHR,DSN=yyy.SCSFMOD0   (the ICSF load library)
//SYSLIB    DD DISP=SHR,DSN=xxxxxx.SDFHLOAD
//SDFHLOAD  DD DISP=SHR,DSN=xxxxxx.SDFHLOAD
//SCSFMOD0  DD DISP=SHR,DSN=yyy.SCSFMOD0   (the ICSF load library)
//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(10,10))
//SYSPRINT  DD SYSOUT=*
//SYSLIN    DD *
       INCLUDE SDFHLOAD(DFHEAI)
       REPLACE CSFDHEAI(DFHEAI),CSF0EAI
       INCLUDE SCSFMOD0(CSFATREN)
         ENTRY DFHEAI
     NAME CSFATREN(R)
```

```
       INCLUDE SDFHLOAD(DFHEAI)
       REPLACE CSFDHEAI(DFHEAI),CSF0EAI
       INCLUDE SCSFMOD0(CSFATRUE)
         ENTRY DFHEAI
     NAME CSFATRUE(R)
/*
```

2. Include the ICSF load module data set in the CICS startup job control language as shown in this example.

```
//DFHRPL    DD DISP=SHR,DSN=xxxxx.SDFHLOAD
//          DD DISP=SHR,DSN=yyy.SCSFMOD0 (The ICSF load library)
//          DD ...
...
//SYSIN     DD DISP=SHR,DSN=xxxxx.SYSIN(DFH$SIPx)
...
```

   In the previous sample code, DFH$SIPx includes the entry:

```
PLTPI=yy,
```

3. Customize the Program Load Table (PLT), to include the ICSF enabling routine CSFATREN in second stage initialization.

   This is an example input deck for compiling a PLT for automatic enablement of the CICS-ICSF link. This is ASM code. Assemble it with the CICS macro library, but **without** the CICS translator.

```
//SYSIN DD *
*
* List of programs to be executed sequentially during system
* initialization.  Required system initialization parm: PLTPI=yy
* DFHPLTCS should be defined in the CSD by CEDA or DFHCSDUP job
*
DFHPLT TYPE=INITIAL,SUFFIX=yy
*
* -------- Second stage of initialization -----------------
*
DFHPLT TYPE=ENTRY,PROGRAM=CSFATREN  (Run enable of CSFATRUE)
*
* ---------- Delimiter between Stages 2 and 3 ------------
*
DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
*
* --------- Third stage of initialization -----------------
* (none)
*
DFHPLT TYPE=FINAL
END
/*
```

   The previous code is an example only. Your CICS administrator can use it as a guide in customizing the PLT. For more information about coding the PLT, refer to *CICS Resource Definition Guide*.

4. Link edit the PLT with these controls:

```
INCLUDE OBJLIB(DFHPLTyy)
NAME    DFHPLTyy(R)
```

5. The CICS administrator should customize the system CSD to include:
   - CSFATRUE
   - CSFATREN
   - A PLT to indicate that initialization is to call CSFATREN to enable the ICSF TRUE, CSFATRUE

   This is an example of the job control language and input. In this example, xxxxx represents the local CICS prefix, and zzzzzzzz represents the PLT entry that was compiled previously.

```
//UPDATE  JOB  ...
//*- - - - - - - - - - - - - - - - - - - - -
//DEFINES EXEC PGM=DFHCSDUP,REGION=2M
//STEPLIB   DD DISP=SHR,DSN=xxxxxx.SDFHLOAD
//          DD DISP=SHR,DSN=zzzzzzzz
//DFHCSD    DD DISP=SHR,DSN=xxxxxx.DFHCSD
//SYSPRINT  DD SYSOUT=A
//SYSIN     DD *
*
DEFINE PROGRAM(CSFATREN) GROUP(ICSF)
                         DESCRIPTION(TRUE enablement routine)
                         LANGUAGE(ASSEMBLER)
*
DEFINE PROGRAM(CSFATRUE) GROUP(ICSF)
                         DESCRIPTION(ICSF interface TRUE)
                         LANGUAGE(ASSEMBLER)
                         CONCURRENCY(THREADSAFE)
                         API(OPENAPI)
*
DEFINE PROGRAM(DFHPLTyy) GROUP(ICSF)
                         DESCRIPTION(PLT Program Init for CSFATRUE)
                         LANGUAGE(ASSEMBLER)
```

The PLT in the example runs the program CSFATREN during CICS initialization. CSFATREN automatically enables the ICSF TRUE, CSFATRUE. If CICS is already started, use a CICS Command Level Interpreter Transaction (CECI) to enable CSFATRUE. To do this, go into CECI and issue this statement:

```
ENABLE PROGRAM('CSFATRUE') TALENGTH(250) LINKEDITMODE START
```

You can also do this in a single step with this statement:

```
CECI ENABLE PROGRAM('CSFATRUE') TALENGTH(250) LINKEDITMODE START
```

6. If you have any existing CICS applications which invoke any of the ICSF services in the Wait List, then these applications must be re-linked with the current ICSF stubs.

Implementing the CICS wait list

The CICS Wait List can be implemented by means of a customer modifiable data set, pointed to by the Installation Options Data Set (WAITLIST parameter). The default WAITLIST includes all services which can complete asynchronously (for example, those services which perform I/O to a cryptographic key data set and those services which are routed to a PCICC or PCIXCC). If the option is not specified, the default CICS Wait List will be utilized by ICSF when a CICS application invokes an ICSF callable service. If WAITLIST is specified, the data set specified by this parameter will be used to determine the names of the services to be placed on the CICS Wait List. A sample data set is provided by ICSF via member CSFWTL00 (for CCF systems with PCICCs) and CSFWTL01 (for systems with PCIXCCs) of SYS1.SAMPLIB. The sample data set contains the same entries as the default ICSF CICS Wait List -- for example, the data set contains the names of all ICSF callable services which, by default, will be driven through the CICS TRUE.

The WAITLIST option should be added to the Installation Options data set under these conditions.

- Non-CICS customers will not specify a WAITLIST keyword.
- CICS customers who want to use the default CICS Wait List shipped with ICSF will not specify a WAITLIST keyword. If you have any existing CICS applications which invoke any of the ICSF services in the Wait List, then these applications must be re-linked with the current ICSF stubs.

- CICS customers who do not want to make use of CICS TRUE must either not enable the TRUE or specify a WAITLIST keyword and point to an empty wait list data set or you can specify WAITLIST(DUMMY) in the Installation Options data set.
- CICS customers who wish to modify the ICSF default CICS Wait List should modify the sample Wait List data set supplied in member CSFWTL00 (for CCF systems with PCICCs) or member CSFWTL01 (for systems with PCIXCCs, CEX2Cs, or CEX3Cs) of SYS1.SAMPLIB. The WAITLIST keyword in the Installation Options Data Set should be set to point to this data set. If you have any existing CICS applications which invoke any of the ICSF services in the Wait List, then these applications must be re-linked with the current ICSF stubs.

If you already have the CICS-ICSF Attachment facility installed, there are a number of callable services which may potentially be routed to the PCICC, PCIXCC, CEX2C, or CEX3C or may perform other asynchronous processing. If you have a modified CICS Wait List, you should ensure that the wait list data set includes all such services, and any CICS applications which invoke any of these services are re-linked with the current ICSF stubs. As a model, you can use the default CICS Wait List that is shipped with ICSF which includes all services which have an asynchronous interface to ICSF or you can use a sample Wait List data set that is also shipped with ICSF. The sample CICS Wait List data set is contained in member CSFWTL00 (for CCF systems with PCICCs) or in member CSFWTL01 (for systems with PCIXCC, CEX2C, or CEX3C) of SYS1.SAMPLIB. The sample data set contains the same entries as the default ICSF CICS Wait List. You can modify the sample data set to add and/or delete items from the Wait List. Here are some examples of why you might want to modify the sample data set.

For CCF Systems:

- If you do not have a PCI Cryptographic Coprocessor installed, you can delete all of the services identified with an "*" that are in the sample wait list.
- If you have a PCI Cryptographic Coprocessor installed, you can examine the services your applications invoke in a CICS environment and determine, based upon the routing information provided for each service in *z/OS Cryptographic Services ICSF Application Programmer's Guide*, SA22-7522, that the service will never be routed to a PCI Cryptographic Coprocessor. In this case (except for the CKDS/PKDS access services) the service can be deleted from the list.

For CCF systems with a PCICC or z990/z890 systems with a PCIXCC/CEX2C:

- If you have an application which invokes a UDX while running under CICS, then the name of the UDX generic service should be added to the CICS Wait List.

If you use a CICS Wait List data set, you need to identify the data set to ICSF through the WAITLIST(data_set_name) option in the ICSF Installation Options data set. The data set can be a member of a PARMLIB, a member of a partitioned data set, or a sequential data set. The data set should be allocated on a permanently resident volume and should adhere to:

- The format of each record in the data set must be fixed length or fixed block length.
- A physical line in the data set must be a LRECL of 80 characters long. The system ignores any characters in positions 73 to 80 of the line.
- You can delimit comments by "/*" and "*/" and include them anywhere in the text. A comment cannot span physical records.
- Only one service may be specified on a logical line.

Note: You can use the WAITLIST(DUMMY) parameter to specify a null CICS Wait List data set, or you can disable the CICS TRUE if you do not want to utilize the CICS TRUE. See “Parameters in the installation options data set” on page 38 for additional information.

Appendix D. Helpful Hints for ICSF First Time Startup

The purpose of this topic is to provide some helpful hints and resolutions for the problems that you may encounter when starting ICSF for the first time.

See Appendix F, “z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, or z196 without a PCIXCC, CEX2C, or CEX3C,” on page 319 if you're running in this environment.

Checklist for First-Time Startup of ICSF

This is a checklist for the first-time startup of ICSF.

Note: ALL crypto coprocessors cards must be loaded with the same level of code. Otherwise, unpredictable results can occur.

Step 1. Hardware Setup - CCF Systems

Process Crypto Enablement Diskette Load

PCICC FCV Load (if applicable)

Power-on Reset

Responsible CE or Client Operator Representative

Where Support Element

Verify Via Cryptographic Coprocessor Configuration Task

- Status for CP0 and/or CP1 is "Initialized"

Via PCI Cryptographic Coprocessor Configuration Task

- Status for Pxx is "Configured"

References *Support Element Operations Guide*

Completed

Step 1. Hardware Setup - PCIXCC/CEX2C/CEX3C Systems

Note: The CP Assist for Cryptographic Functions feature is required for selection of the PCIXCC/CEX2C/CEX3C in the activation profiles.

Process LIC installed for CP Assist for Cryptographic Functions

Note: If using TKE V4.0 or higher and your z890/z990 system is running with May 2004 or higher version LIC or your z9 EC or z9 BC system is running with MCL 029 Stream J12220 or higher version of LIC, you must Permit each PCIXCC/CEX2C for TKE Commands.

Responsible CE or Client Operator Representative

Where Support Element

Verify Via CPC details

- CP Assist for Cryptographic Functions is 'Installed'
- CP Assist for Cryptographic Functions DES/TDES enablement (feature 3863) is 'Installed'

Via PCI Cryptographic Configuration Task

- Status for each PCICA, PCIXCC, CEX2C, CEX2A, CEX3C, or CEX3A is 'Configured'

  Note: If using TKE V4.0 or higher, the status for each PCIXCC/CEX2C is 'Permitted' when May 2004 or higher LIC is installed on a z890/z990 or MCL 029 Stream J12220 or higher is installed on a z9 EC or z9 BC.

References *Support Element Operations Guide*

Completed

Step 2. LPAR Activation Profiles - CCF Systems

Process Crypto Page Setup

PCICC Page Setup

Processor Page Setup

Responsible CE or Client Operator Representative

Where Support Element

Verify On Crypto Page of the Activation Profile

- Enable Public Key Algorithm
- Enable Cryptographic Functions
- Enable PKSC and ICSF
- Enable Cryptographic Facility (ICRF) Key Entry
- Enable Special Secure Mode

If using TKE, also

- Enable Modify Authority (Only (1) LPAR - TKE Host)
- Enable Query Signature Controls (TKE Host)
- Enable Query Transport Controls (TKE Host)
- For the TKE Host, the Control Domain must include ALL the domains that will be controlled by the TKE Host

On the PCICC Page of the Activation Profile

- PCI Cryptographics Coprocessor Candidate List includes all PCICC's and PCICA's that CAN be online
- PCI Cryptographics Coprocessor Online List includes all PCICC's and PCICAs that WILL be online when activation is complete (Selections in the Online List MUST be selected in the Candidate List)

On the Processor Page Setup

- Cryptographic Coprocessor(s) are enabled for that LPAR

References *Support Element Operations Guide*

z/OS Cryptographic Services ICSF TKE Workstation User's Guide, SA23-2211 (LPAR Considerations)

zSeries PR/SM Planning Guide

Completed

Step 2. LPAR Activation Profiles - PCIXCC, CEX2C, and CEX3C Systems

Process PCI Crypto Page Setup

Responsible CE or Client Operator Representative

Where Support Element

Verify Control Domain Index

Usage Domain Index

PCI Cryptographic Candidate List includes all PCICAs, PCIXCCs, CEX2Cs, CEX2As, CEX3Cs, and CEX3As that CAN be online

PCI Cryptographic Online List includes all PCICAs, PCIXCCs, CEX2Cs, CEX2As, CEX3Cs, and CEX3As that WILL be online when activation is complete (Selections in the Online List MUST be selected in the Candidate List)

References *Support Element Operations Guide*

z/OS Cryptographic Services ICSF TKE Workstation User's Guide, SA23-2211 (LPAR Considerations)

zSeries PR/SM Planning Guide

Completed

Note: If TKE is to be used, then ALL PCIXCCs, CEX2Cs, and CEX3Cs that you want TKE to be able to control MUST be defined in the Online and Candidate Lists. Also, the Usage Domain for the TKE Host LPAR **MUST** be unique. While the same domain may be used by other LPARs as long as these LPARs do not share any of the same PCIXCC/CEX2C/CEX3C cards, the TKE Host domain must have access to all the PCIXCC/CEX2C/CEX3C cards so that prohibits any other LPAR from using the same domain.

Step 3. ICSF Setup

Process Install and Customize ICSF

Responsible System Programmer and ICSF Administrator

Where TSO and ISPF Panels

Verify Customize SYS1.PARMLIB

- Add CSF.SCSFMOD0 to the LNKLST concatenation
- Update PROGxx to APF authorize CSF.SCSFMOD0
- Update IKJTSOxx for ICSF by adding CSFDAUTH and CSFDPKDS to the AUTHPGM and AUTHTSF parameter lists. To change the active IKJTSOxx member of SYS1.PARMLIB, use the PARMLIB UPDATE command.

CKDS and PKDS created

ICSF Startup Procedure created

Installation Options Dataset created

- The DOMAIN parameter in the installation options data set is optional. It is required if more than one domain is specified as the usage domain on the PR/SM panels or if running in native mode.

- CKDS and PKDS names specified
- COMPAT(NO) and SSM(YES)

  Note: SSM(YES) is not required with a PCIXCC, CEX2C, or CEX3C

Access provided to the ICSF panels

References Chapter 2, “Installation, Initialization, and Customization,” on page 13

Completed

Step 4. TKE Setup

If you are not using TKE, proceed to the next step.

Process Initialize the TKE Workstation

Configure TCP/IP on the Host and the TKE Workstation

Perform passphrase or smart card setup

Setup the TKE Host Transaction Program

- Create JCL to start the TKE Host Transaction Program
- RACF Security Setup
- Start the TKE Host Transaction Program

Responsible Network Programmer, System Programmer and TKE Administrator

Where ISPF Panels, TKE Workstation

Verify CSFTTKE is authorized in the AUTHCMD list of IKJTSOxx in SYS1.PARMLIB

TKE Host Transaction Program (CSFTTCP) is defined in the RACF STARTED class (If your installation has a Generic Userid associated to all started procedures, this is not necessary)

CSFTTKE profile is defined in the RACF FACILITY and RACF APPL classes

References *z/OS Cryptographic Services ICSF TKE Workstation User's Guide*, SA23-2211 (See Topics: TKE Workstation Setup and Customization and TKE TCP/IP and Host Considerations)

Completed

Step 5. ICSF Startup

Process Start ICSF

Responsible Client Operator Representative or System Programmer

Where Operator Console

References Chapter 2, “Installation, Initialization, and Customization,” on page 13

Completed

Step 6. Loading Master Keys and Initializing the CKDS through ICSF Panels

Notes:

1. If sharing a CKDS between CCF systems and non-CCF systems, the CKDS must be initialized on a CCF system.
2. When defining a master key by specifying master key parts, **make sure the key parts are recorded and saved in a secure location.** When you are entering the key parts for the first time, be aware that **you may need to reenter these same key values at a later date to restore master key values that have been cleared.** If defining a master key using a pass phrase, realize that the same pass phrase will always produce the same master key values, and is therefore as critical and sensitive as the master key values themselves. Make sure you save the pass phrase so that you can later reenter it if needed. Because of the sensitive nature of the pass phrase, make sure you secure it in a safe place.

If you are using TKE, proceed to the next step.

Process Passphrase Initialization to load and SET master keys and initialize CKDS and PKDS

- Create NOCV, ANSI, and ESYS keys as applicable for your installation

  Note: These system keys are not valid on a PCIXCC, CEX2C, or CEX3C system.

- OR -

Clear Master Key Entry

- Load DES New Master Key
- Load AES New Master Key
- Load PKA Signature Master Key (SMK)
- Load PKA Key Management Master Key (KMMK)
- Load New Symmetric Master Key (if applicable)
- Load New Asymmetric Master Key (if applicable)

  Note: Using the Coprocessor Management panel, the master keys can be loaded into all the coprocessors (CCF, PCICC, PCIXCC, CEX2C, and CEX3C) at the same time. It is recommended that the SMK and KMMK keys be set to the same value.

- Initialize CKDS
- Create NOCV, ANSI, and ESYS keys as applicable for your installation - CCF systems only
- Initialize the PKDS
- Enable PKA Services
- Enable PKDS Read Access
- Enable PKDS Write, Create, and Delete Access

Responsible ICSF Administrator and Key Officers

Where ICSF Panels

Verify In System Log (CCF Systems):

```
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
IEE504I CRYPTO(0),ONLINE
IEE504I CRYPTO(1),ONLINE  (if applicable)
CSFM116I BOTH MASTER KEYS CORRECT ON PCI CRYPTOGRAPHIC
COPROCESSOR Pnn, SERIAL NUMBER nn-nnnn  (if applicable)
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
CSFM400I CRYPTOGRAPHY SERVICES ARE NOW AVAILABLE
```

In System Log (PCIXCC, CEX2C, or CEX3C Systems):

```
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM124I MASTER KEY DES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY AES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY ECC ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY RSA ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CKDS IS NOT INITIALIZED.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.PKDS IS NOT INITIALIZED.
CSFM012I NO ACCESS CONTROL AVAILABLE FOR CRYPTOZ RESOURCES. ICSF PKCS11
SERVICES DISABLED.
CSFM122I PKA SERVICES WERE NOT ENABLED DURING ICSF INITIALIZATION
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM009I NO ACCESS CONTROL AVAILABLE FOR ICSF SERVICES OR KEYS
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
```

Message CSFM440I will be issued for each active PCIXCC.

Message CSFM124I will be issued for each CEX2C/CEX3C online. The ECC master key is available only on the CEX3C.

Message CSFM122I will not be issued when your system has any CEX3C coprocessors (with the Sep. 2011 or later LIC) online. The PKA callable services control will not be active. The availability of RSA callable services will depend on the status of the RSA master key. CSFM130I is issued when the RSA master key is active and RSA callable services are available.

In System Log (CEX2C or CEX3C without CEX2A or CEX3A Systems):

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM124I MASTER KEY DES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY AES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY ECC ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
```

```
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY RSA ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM129I MASTER KEY mk ON coprocessor-name cii, SERIAL
NUMBER nnnnnnn, IS CORRECT.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM400I CRYPTOGRAPHY - SERVICES ARE NOW AVAILABLE.
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
CSFM127I CRYPTOGRAPHY - AES SERVICES ARE AVAILABLE.
```

Message CSFM129I will be issued for each CEX2C/CEX3C online.

In System Log (CEX2C/CEX3C and CEX2A/CEX3A Systems):

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM101E PKA KEY DATA SET, CSF.PKDS IS NOT INITIALIZED.
CSFM124I MASTER KEY DES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY AES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY ECC ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY RSA ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CKDS IS NOT INITIALIZED.
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
CSFM400I CRYPTOGRAPHY - SERVICES ARE NOW AVAILABLE.
CSFM127I CRYPTOGRAPHY - AES SERVICES ARE AVAILABLE.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS3 COPROCESSOR
xxx, SERIAL NUMBER nnnnnnn
```

Message CSFM124I will be issued for each CEX2C/CEX3C online. The ECC master key is available only on the CEX3C.

Message CSFM111I will be issued for each active CEX2C/CEX3C.

In System Log (CPACF only system):

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
```

In System Log (CPACF, CEX2A, and CEX3A)

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS3 COPROCESSOR
xxx, SERIAL NUMBER nnnnnnn
```

```
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
```

Message CSFM111I will be issued for each active CEX2A/CEX3A.

References For information on using the Pass Phrase Initialization Utility and managing master keys, refer to *z/OS Cryptographic Services ICSF Administrator's Guide*, SA22-7521.

Completed

Step 7. Customizing TKE and Loading Master Keys

If you are not using TKE, proceed to the next step.

Process - CCF Systems

TKE Administrator's and Key Officers

- Define Host IDs
- Define CCF Authorities
- Define Access Controls (Signature Requirements for CCF)
- Define Roles (if applicable)
- Define PCI Cryptographic Coprocessor Authorities (if applicable)
- Load DES New Master Key
- Load PKA Signature Master Key (SMK)
- Load PKA Key Management Master Key (KMMK)
- Load New Symmetric Master Key (if applicable)
- Load and SET New RSA Master Key (if applicable)

Note: If you have more than one crypto module or PCI Cryptographic Coprocessor, repeat the process for each, unless Groups have been defined. It is recommended that the SMK and KMMK keys be set to the same value.

Process - PCIXCC/CEX2C/CEX3C Systems

TKE Administrator's and Key Officers

- Define Host IDs
- Define Roles
- Define PCIXCC/CEX2C/CEX3C Authorities
- Load New DES-MK
- Load New AES-MK (if running on z10 or z196 servers with a CEX2C or CEX3C and the Nov. 2008 or later licensed internal code (LIC))
- Load and SET New RSA-MK or ECC-MK

Note: If you have more than one PCIXCC, CEX2C, or CEX3C, repeat the process for each, unless Groups have been defined.

Responsible ICSF Administrator

- Initialize CKDS and SET the DES/SYM-MK New Master Key
- Create NOCV, ANSI, and ESYS keys as applicable for your installation - CCF Systems only
- Load PKA/RSA-MK/ECC-MK Master Keys

- SET RSA-MK (PCICC, PCIXCC, CEX2C, and CEX3C) and/or ECC-MK (CEX3C)
- Initialize the PKDS
- Enable PKA Services
- Enable PKDS Read Access
- Enable PKDS Write, Create, and Delete Access

Where TKE Workstation and ICSF Panels

Verify In System Log (CCF Systems):

```
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
IEE504I CRYPTO(0),ONLINE
IEE504I CRYPTO(1),ONLINE  (if applicable)
CSFM116I BOTH MASTER KEYS CORRECT ON PCI CRYPTOGRAPHIC
COPROCESSOR Pnn, SERIAL NUMBER nn-nnnn  (if applicable)
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
CSFM400I CRYPTOGRAPHY SERVICES ARE NOW AVAILABLE
```

In System Log (PCIXCC, CEX2C, or CEX3C Systems):

```
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM124I MASTER KEY DES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY AES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY ECC ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY RSA ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CKDS IS NOT INITIALIZED.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.PKDS IS NOT INITIALIZED.
CSFM012I NO ACCESS CONTROL AVAILABLE FOR CRYPTOZ RESOURCES. ICSF PKCS11
SERVICES DISABLED.
CSFM122I PKA SERVICES WERE NOT ENABLED DURING ICSF INITIALIZATION
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM009I NO ACCESS CONTROL AVAILABLE FOR ICSF SERVICES OR KEYS
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
```

Message CSFM440I will be issued for each active PCIXCC.

Message CSFM124I will be issued for each CEX2C/CEX3C online. The ECC master key is available only on the CEX3C.

| Message CSFM122I will not be issued when your system has any CEX3C coprocessors (with the Sep. 2011 or later LIC) online. The PKA callable services control will not be active. The availability of RSA callable services will depend on the status of the RSA master key. CSFM130I is issued when the RSA master key is active and RSA callable services are available.

In System Log (CEX2C or CEX3C without CEX2A or CEX3A Systems):

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM124I MASTER KEY DES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY AES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY ECC ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY RSA ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM129I MASTER KEY mk ON coprocessor-name cii, SERIAL
NUMBER nnnnnnn, IS CORRECT.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM400I CRYPTOGRAPHY - SERVICES ARE NOW AVAILABLE.
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
CSFM127I CRYPTOGRAPHY - AES SERVICES ARE AVAILABLE.
```

Message CSFM129I will be issued for each CEX2C/CEX3C online.

In System Log (CEX2C/CEX3C and CEX2A/CEX3A Systems):

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM101E PKA KEY DATA SET, CSF.PKDS IS NOT INITIALIZED.
CSFM124I MASTER KEY DES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY AES ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY ECC ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM124I MASTER KEY RSA ON CRYPTO EXPRESS3 COPROCESSOR xxx, SERIAL
NUMBER nnnnnnnn, NOT INITIALIZED.
CSFM100E CRYPTOGRAPHIC KEY DATA SET, CSF.CKDS IS NOT INITIALIZED.
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
CSFM400I CRYPTOGRAPHY - SERVICES ARE NOW AVAILABLE.
CSFM127I CRYPTOGRAPHY - AES SERVICES ARE AVAILABLE.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS3 COPROCESSOR
xxx, SERIAL NUMBER nnnnnnn
```

Message CSFM124I will be issued for each CEX2C/CEX3C online. The ECC master key is available only on the CEX3C.

Message CSFM111I will be issued for each active CEX2C/CEX3C.

In System Log (CPACF only system):

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.
CSFM508I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC ACCELERATORS ONLINE.
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
```

In System Log (CPACF, CEX2A, and CEX3A)

```
S CSF
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM111I CRYPTOGRAPHIC FEATURE IS ACTIVE. CRYPTO EXPRESS3 COPROCESSOR
xxx, SERIAL NUMBER nnnnnnn
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM126I CRYPTOGRAPHY - FULL CPU-BASED SERVICES ARE AVAILABLE.
```

Message CSFM111I will be issued for each active CEX2A/CEX3A.

References

For information on managing master keys, refer to *z/OS Cryptographic Services ICSF Administrator's Guide*, SA22-7521.

Completed

Step 8. CICS-ICSF Attachment Facility Setup

If you are not using CICS, proceed to the next topic.

Process Follow the instructions in Appendix C, “CICS-ICSF Attachment Facility,” on page 299 if desired.

Responsible System Programmer

Where Sample Jobs

References Appendix C, “CICS-ICSF Attachment Facility,” on page 299

Completed

Step 9. Complete ICSF initialization

See “Steps for initializing ICSF” on page 31

Responsible System Programmer

Where Operator Console

Completed

Commonly Encountered ICSF First Time Setup/initialization Messages

These ICSF messages are commonly encountered during initialization and first time startup of ICSF.

- **CSFM105E CRYPTOGRAPHY - DOMAIN '***domain*' **IS NOT ACCESSIBLE** - A domain mismatch exists between the domain you have selected in your LPAR activation profile and the domain option specified in your ICSF options data set. You must decide which domain is the one you want and correct it in the appropriate location.
- **CSFM107E CRYPTOGRAPHY - CRYPTO MODULES CONFIGURED DIFFERENTLY** - The CCC values on both of your crypto coprocessors (CCFs) must be the same. One of the cryptos may not have been loaded with an enablement diskette yet, or selected for next activation with the force zeroize option. Ensure that both crypto coprocessors are loaded with the same configuration. An IPL will be required.

This message is only if you have a CCF System.

- **CSFM120E PUBLIC KEY SECURE CABLE (PKSC) FACILITY IS NOT ENABLED** - The Enable cryptographic functions option and/or the Enable public key secure cable (PKSC) and integrated cryptographic service facility (ICSF) option is not enabled in the LPAR activation profile. Check the appropriate boxes to enable the options.

  This message is only if you have a CCF System.
- **CSFM124I MASTER KEY** *mk* ON *coprocessor-name cii*, **SERIAL NUMBER** *nnnnnnn*, **NOT INITIALIZED** - The cryptographic coprocessor does not have the master key. When a master key is not set, then the cryptographic coprocessor may not be used for operations with the master key until the system administrator has provided the master key. This may be a normal situation for your installation. Have the system administrator enter the correct master key if appropriate.
- **CSFM410E ERROR IN OPTIONS DATA SET** - ICSF could not interpret the options data set. Check the CSF job output for diagnostic messages.

CCF Systems ONLY: Before trying to start ICSF, ensure that the crypto coprocessors have been initialized with the enablement diskette. If the coprocessors have been loaded, a configuration should be available to select for next activation from the Cryptographic Coprocessors Configuration panels. If the crypto coprocessors have not been loaded with the enablement diskette and ICSF is started, message CSFM107E will be issued. This message will only be issued if you have 2 Cryptographic Coprocessor Features and they do not contain the same CCC. If the CCCs have not been initialized (are all zeroes) you will receive an X'18F' reason code 4a abend.

A PKDS is required. The PKDS data set name must be specified in the options data set with the PKDSN option. If a PKDS is not specified, you will receive these messages:

```
CSFM408A NO PKDS NAME WAS SPECIFIED IN THE OPTIONS DATA SET.
CSFM401I CRYPTOGRAPHY - SERVICES NO LONGER AVAILABLE.
```

Appendix E. Using AMS REPRO Encryption

This appendix provides information on using IDCAMS REPRO ENCIPHER and DECIPHER options with ICSF.

Steps for setting up ICSF

Perform these tasks to use the ENCIPHER and DECIPHER parameters with ICSF:

1. Define the key value that is used to encrypt and decrypt the data key. To define the key value, use one of these ICSF key administrative options:
   - Trusted Key Entry (TKE) workstation. For information about how to define the key value using the TKE workstation, see *z/OS Cryptographic Services ICSF TKE Workstation User's Guide*.
   - Key generator utility program (KGUP). Use the KGUP panel "ICSF - Create ADD, UPDATE, or DELETE Key Statement" to define the key value. For more information about how to use KGUP panels, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

     Be aware of the following restrictions:
     - The length of the data encryption key is limited to 8 bytes, or 56-bit DES. Triple DES support is not available.
     - Key labels are limited to 8 characters because of the fixed size of REPRO storage areas.
     - The REPRO command's encryption algorithm variables are not documented, so you cannot use them to write decryption applications on another system. Therefore, cross-platform exchange is not possible.
2. Refresh ICSF's cryptographic key data set (CKDS) so that the key value can be used by REPRO.
3. Ensure that ICSF can support PCF macro calls by specifying COMPAT(YES) in the ICSF installation options. For more information about how to specify ICSF installation options, see Chapter 2, “Installation, Initialization, and Customization,” on page 13.

   If you had to change the ICSF installation options, you must restart ICSF.
4. Run the REPRO ENCIPHER or DECIPHER job.

Restrictions: The REPRO command's encryption algorithm variables are not documented, so you cannot use them to write decryption applications on another system. Therefore, cross-platform exchange is not possible.

Recommendation: Do not specify the REPRO parameter PRIVATEKEY, because it exposes the clear data key value. Instead, specify either EXTERNALKEY or INTERNALKEY, and STOREDATAKEY

Appendix F. z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, or z196 without a PCIXCC, CEX2C, or CEX3C

This topic describes the processing of the z890, z990, z9 EC, z9 BC, z10 EC and z10 BC environment without a PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor. Note that these servers do not support the Cryptographic Coprocessor Feature or the PCI Cryptographic Coprocessor.

Applications and programs

Applications requiring secure cryptography using encrypted keys will not be able to execute on the z890, z990, z9 EC, z9 BC, z10 EC, z10 BC, and z196 without a PCI X Cryptographic Coprocessor Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor. All cryptographic keys must be clear keys.

These applications and programs are not supported:

- Access Method Services Cryptographic option
- CICS attachment facility
- CKDS Conversion program
- CSFEUTIL program for CKDS reencipher, refresh, change master key, and passphrase initialization functions
- CSFPUTIL program for PKDS reencipher and refresh functions.
- Distributed Key Management System (DKMS)
- Key Generation Utility Program (KGUP)
- PCF applications
- UDX (User Defined Extension) support
- VTAM Session Level Encryption
- 4753-HSP applications
- Applications that access ICSF services through the BSAFE interfaces
- If only the CPACF feature is installed you will not be able to:
  1. Set master keys (SYM-MK and ASYM-MK)
  2. Initialize the CKDS and PKDS
  3. Store keys in the CKDS and PKDS.

Callable services

These services are available when running on a z990, z890, z9 EC, z9 BC, z10 EC, z10 BC, and z196 without a PCI X Cryptographic Coprocessor, Crypto Express2 Coprocessor, or Crypto Express3 Coprocessor:

- Character/Nibble Conversion (CSNBXBC and CSNBXCB)
- Code Conversion (CSNBXEA and CSNBXAE)
- Control Vector Generate (CSNBCVG)
- Decode (CSNBDCO) - This service requires CP Assist for Cryptographic Functions.
- Digital Signature Verify (CSNDDSV) - This service requires a PCI Cryptographic Accelerator.
- Encode (CSNBECO) - This service requires CP Assist for Cryptographic Functions.

- ICSF Query Sevice (CSFIQF) - The only part of this service available without a PCIXCC/CEX2C/CEX3C is the ICSFSTAT function.
- ICSF Query Algorithm (CSFIQA)
- MDC Generate (CSNBMDG and CSNBMDG1) - This service requires CP Assist for Cryptographic Functions.
- One-Way Hash Generate (CSNBOWH and CSNBOWH1) - This service requires CP Assist for Cryptographic Functions.
- PKA Decrypt (CSNDPKD) - This service requires a PCI Cryptographic Accelerator.
- PKA Encrypt (CSNDPKE) ZERO-PAD formatting only - This service requires a PCI Cryptographic Accelerator.
- PKA Key Token Build (CSNDPKB)
- PKA Public Key Extract (CSNDPKX)
- PKCS #11 Derive multiple keys (CSFPDMK)
- PKCS #11 Derive key (CSFPDVK)
- PKCS #11 Get attribute value (CSFPGAV)
- PKCS #11 Generate key pair (CSFPGKP)
- PKCS #11 Generate secret key (CSFPGSK)
- PKCS #11 Generate HMAC (CSFPHMG)
- PKCS #11 Verify HMAC (CSFPHMV)
- PKCS #11 One-way hash generate (CSFPOWH)
- PKCS #11 Private key sign (CSFPPKS)
- PKCS #11 Public key verify (CSFPPKV)
- PKCS #11 Pseudo-random function (CSFPPRF)
- PKCS #11 Set attribute value (CSFPSAV)
- PKCS #11 Secret key decrypt (CSFPSKD)
- PKCS #11 Secret key encrypt (CSFPSKE)
- PKCS #11 Token record create (CSFPTRC)
- PKCS #11 Token record delete (CSFPTRD)
- PKCS #11 Token record list (CSFPTRL)
- PKCS #11 Unwrap key (CSFPUWK)
- PKCS #11 Wrap key (CSFPWPK)
- Symmetric Key Decipher (CSNBSYD and CSNBSYD1) - This service requires CP Assist for Cryptographic Functions.
- Symmetric Key Encipher (CSNBSYE and CSNBSYE1) - This service requires CP Assist for Cryptographic Functions.
- Symmetric MAC Generate (CSNBSMG, CSNBSMG1, CSNESMG, and CSNESMG1)
- Symmetric MAC Verify (CSNBSMV, CSNBSMV1, CSNESMV, and CSNESMV1)
- X9.9 Data Editing (CSNB9ED)

Installation defined callable services are supported only if you're using clear keys and using one of the supported callable services.

ICSF Setup and Initialization

Starting ICSF on a z990, z890, or z9 EC or z9 BC without a PCI Cryptographic Accelerator or PCIXCC/CEX2C or starting ICSF on a z9 EC, z9 BC, z10 EC, z10 BC, or z196 without a CEX2C, CEX2A,CEX3C, or CEX3A:

- Starting ICSF on a z990, z890, z9 EC, z9 BC, z10 EC, z10 BC, or z196 without a PCICA, PCIXCC, CEX2C, or CEX3C:

```
S CSF
$HASP100 CSF ON STCINRDR
IEF695I START CSF WITH JOBNAME CSF IS ASSIGNED TO USER
++++++++
$HASP373 CSF STARTED
IEF403I CSF - STARTED - TIME=11.07.28
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM506I CRYPTOGRAPHY - THERE IS NO ACCESS TO ANY CRYPTOGRAPHIC COPROCESSORS OR
ACCELERATORS.
CSFM122I PKA SERVICES WERE NOT ENABLED DURING ICSF INITIALIZATION
CSFM001I ICSF INITIALIZATION COMPLETE
```

- Starting ICSF on a z990 or z890 with a PCI Cryptographic Accelerator and without a PCI X Cryptographic Coprocessor or Crypto Express2 Coprocessor, you'll receive message CSFM411I for each PCI Cryptographic Accelerator that is Active. Starting ICSF on a z9 EC, z9 BC, z10 EC, z10 BC, or z196 with a Crypto Express2 Accelerator, and without a Crypto Express2 Coprocessor, you'll receive message CSFM435I for each Crypto Express2 Accelerator that is Active.

```
S CSF
$HASP100 CSF ON STCINRDR
IEF695I START CSF WITH JOBNAME CSF IS ASSIGNED TO USER
++++++++
$HASP373 CSF STARTED
IEF403I CSF - STARTED - TIME=11.08.15
CSFM607I A CKDS KEY STORE POLICY IS NOT DEFINED.
CSFM607I A PKDS KEY STORE POLICY IS NOT DEFINED.
CSFM610I GRANULAR KEYLABEL ACCESS CONTROL IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR AES IS DISABLED.
CSFM611I XCSFKEY EXPORT CONTROL FOR DES IS DISABLED.
CSFM612I PKA KEY EXTENSIONS CONTROL IS DISABLED.
CSFM411I PCI CRYPTOGRAPHIC ACCELERATOR Ann is ACTIVE (for z890/z990)
CSFM435I CRYPTO EXPRESS2 ACCELERATOR Fpp IS ACTIVE (for z9 EC/z9 BC)
CSFM507I CRYPTOGRAPHY - THERE ARE NO CRYPTOGRAPHIC COPROCESSORS ONLINE.
CSFM122I PKA SERVICES WERE NOT ENABLED DURING ICSF INITIALIZATION
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM400I CRYPTOGRAPHY - SERVICES ARE NOW AVAILABLE.
```

Secure Sockets Layer (SSL)

System SSL applications are supported on the z990, z890, z9 EC, z9 BC, z10 EC, z10 BC, or z196. SSL defines methods for data encryption, server authentication, message integrity, and client authentication for a TCP/IP connection. Security is provided on the link and callable services have been enhanced for DES, TDES and SHA-1 services.

TKE workstation

The Trusted Key Entry (TKE) workstation is not available with this hardware configuration.

Appendix G. Accessibility

Publications for this product are offered in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when using PDF files, you may view the information through the z/OS Internet Library Web site or the z/OS Information Center. If you continue to experience problems, send an e-mail to mhvrcfs@us.ibm.com or write to:

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
U.S.A.

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS® enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer or Library Server versions of z/OS books in the Internet library at:

`http://www.ibm.com/systems/z/os/zos/bkserv/`

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 give 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 character set (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:

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

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 information 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.

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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This book primarily documents information that is NOT intended to be used as a Programming Interface of ICSF.

This book also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of ICSF. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

Programming Interface information

End of Programming Interface information

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.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Index

Numerics

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Product Number: 5964-A01

Printed in USA

SA22-7520-16