

z/OS  
Version 2 Release 4

*Cryptographic Services  
Integrated Cryptographic Service Facility  
Writing PKCS #11 Applications*



**Note**

Before using this information and the product it supports, read the information in [“Notices” on page 105.](#)

This edition applies to ICSF FMID HCR77D1 and Version 2 Release 4 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2020-10-01

© **Copyright International Business Machines Corporation 2007, 2020.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Tables.....</b>	<b>vii</b>
<b>About this document.....</b>	<b>ix</b>
Who should read this document.....	ix
How this document is organized.....	ix
How to use this document.....	ix
Where to find more information.....	ix
IBM Crypto Education.....	x
<b>How to send your comments to IBM.....</b>	<b>xi</b>
If you have a technical problem.....	xi
<b>Summary of changes.....</b>	<b>xiii</b>
Changes made in Cryptographic Support for z/OS V2R2 - z/OS V2R4 (FMID HCR77D1).....	xiii
Changes made in Cryptographic Support for z/OS V2R2 - z/OS V2R3 (FMID HCR77D0).....	xiv
Changes made in Cryptographic Support for z/OS V2R1 - z/OS V2R3 (FMID HCR77C1).....	xiv
Changes made in Cryptographic Support for z/OS V2R1 - z/OS V2R2 (FMID HCR77C0).....	xiv
<b>Chapter 1. Overview of z/OS support for PKCS #11.....</b>	<b>1</b>
Tokens.....	1
Secure key PKCS #11.....	2
The token data set (TKDS).....	2
Controlling token access and key policy.....	2
Managing tokens .....	7
Sample scenario for setting up z/OS PKCS #11 tokens.....	7
Sample scenario for controlling clear key processing .....	9
Auditing PKCS #11 functions.....	9
Component trace for PKCS #11 functions.....	9
Object types.....	10
Session objects.....	10
Token objects.....	10
Operating in compliance with FIPS 140-2.....	10
Requiring signature verification for ICSF module CSFINPV2.....	13
Requiring FIPS 140-2 compliance from all z/OS PKCS #11 applications.....	14
Requiring FIPS 140-2 compliance from select z/OS PKCS #11 applications.....	14
Preparing to use PKCS #11 applications.....	16
Tasks for the system programmer.....	16
Tasks for the security administrator.....	16
Tasks for the auditor.....	17
Tasks for application programmers.....	17
Optional Crypto Express adapters.....	17
<b>Chapter 2. The C API.....</b>	<b>19</b>
Using the C API.....	19
Deleting z/OS PKCS #11 tokens.....	19
Environment.....	19
Cross memory considerations.....	20
Key types and mechanisms supported.....	20
Additional manifest constants for Dilithium quantum-safe algorithm support.....	30

Objects and attributes supported.....	30
Library, slot, and token information.....	50
Functions supported.....	51
Standard functions supported .....	51
Non-standard functions supported.....	62
Non-standard mechanisms supported.....	63
Enterprise PKCS #11 coprocessors.....	64
Key algorithms/usages that are unsupported or disallowed by the Enterprise PKCS #11 coprocessors .....	64
PKCS #11 Coprocessor Access Control Points.....	65
Standard compliance modes.....	69
Function return codes.....	70
Troubleshooting PKCS #11 applications.....	71
<b>Chapter 3. Sample PKCS #11 C programs .....</b>	<b>73</b>
Running the pre-compiled version of testpkcs11.....	73
Steps for running the pre-compiled version of testpkcs11.....	73
Building sample PKCS #11 applications from source code.....	74
<b>Chapter 4. Regional cryptographic servers.....</b>	<b>77</b>
Regional cryptographic server key types and mechanisms supported.....	77
CKM_IBM_SM2.....	79
CKM_IBM_SM2_ENCRYPT.....	79
CKM_IBM_SM2_KEY_PAIR_GEN.....	80
CKM_IBM_SM2_SM3.....	80
CKM_IBM_SM3.....	81
CKM_IBM_SM4_CBC.....	81
CKM_IBM_SM4_ECB.....	81
CKM_IBM_SM4_ECB_ENCRYPT_DATA.....	82
CKM_IBM_SM4_ISO2_MAC.....	82
CKM_IBM_SM4_ISO2_MAC_GENERAL.....	83
CKM_IBM_SM4_KEY_GEN.....	83
CKM_IBM_SM4_MAC.....	83
CKM_IBM_SM4_MAC_GENERAL.....	84
CKM_XOR_BASE_AND_DATA.....	84
Additional manifest constants for regional cryptographic servers.....	84
API examples for regional cryptographic servers.....	84
<b>Chapter 5. ICSF PKCS #11 callable services.....</b>	<b>89</b>
<b>Appendix A. SMP/E installation data sets, directories, and files.....</b>	<b>91</b>
<b>Appendix B. Source code for the testpkcs11 sample program.....</b>	<b>93</b>
<b>Appendix C. Accessibility.....</b>	<b>101</b>
Accessibility features.....	101
Consult assistive technologies.....	101
Keyboard navigation of the user interface.....	101
Dotted decimal syntax diagrams.....	101
<b>Notices.....</b>	<b>105</b>
Terms and conditions for product documentation.....	106
IBM Online Privacy Statement.....	107
Policy for unsupported hardware.....	107
Minimum supported hardware.....	107
Trademarks.....	108

<b>Glossary.....</b>	<b>109</b>
<b>Index.....</b>	<b>125</b>



---

# Tables

1. Token access levels.....	3
2. Resources in the CSFSERV class for token services.....	4
3. CLEARKEY.token-label resource access and key security policy.....	6
4. Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO).....	21
5. Mechanisms supported by specific cryptographic hardware.....	26
6. Restricted algorithms and uses when running in compliance with FIPS 140-2.....	29
7. Common footnotes for object attribute tables.....	31
8. Data object attributes that ICSF supports.....	31
9. X.509 certificate object attributes that ICSF supports.....	32
10. Secret key object attributes that ICSF supports.....	34
11. Public key object attributes that ICSF supports.....	39
12. RSA public key object attributes that ICSF supports.....	41
13. DSA public key object attributes that ICSF supports.....	42
14. Diffie-Hellman public key object attributes that ICSF supports.....	42
15. Elliptic Curve public key object attributes that ICSF supports.....	42
16. SM2 public key object attributes that ICSF supports.....	42
17. Private key object attributes that ICSF supports.....	43
18. RSA private key object attributes that ICSF supports.....	46
19. DSA private key object attributes that ICSF supports.....	47
20. Diffie-Hellman private key object attributes that ICSF supports.....	47
21. Elliptic Curve private key object attributes that ICSF supports.....	47
22. SM2 private key object attributes that ICSF supports.....	48
23. Domain parameter object attributes that ICSF supports.....	48

24. DSA domain parameter object attributes that ICSF supports.....	49
25. Diffie-Hellman domain parameter object attributes that ICSF supports.....	49
26. Dilithium public key object attributes that ICSF supports.....	49
27. Dilithium private key object attributes that ICSF supports.....	50
28. Standard PKCS #11 functions that ICSF supports.....	51
29. List of algorithms/uses not supported/disallowed by Enterprise PKCS #11 coprocessors.....	64
30. PKCS #11 Access Control Points.....	65
31. Environment variables for capturing trace data.....	71
32. Regional cryptographic server mechanisms and functions.....	78
33. CKM_IBM_SM2: Key and data length.....	79
34. CKM_IBM_SM2_ENCRYPT: Key and input length.....	80
35. CKM_IBM_SM2_SM3: Key and input length.....	81
36. CKM_IBM_SM3: Data and digest length.....	81
37. CKM_IBM_SM4_CBC: Key and data length.....	81
38. CKM_IBM_SM4_ECB: Key and data length.....	82
39. CKM_IBM_ISO2_SM4_MAC: Key and data length.....	83
40. CKM_IBM_SM4_ISO2_MAC_GENERAL: Key and data length.....	83
41. CKM_IBM_SM4_MAC: Key and data length.....	83
42. CKM_IBM_SM4_MAC_GENERAL: Key and data length.....	84



## About this document

---

This document describes the support for PKCS #11 provided by the z/OS® Integrated Cryptographic Service Facility (ICSF). ICSF is a component of z/OS Cryptographic Services, which includes the following components:

- z/OS Integrated Cryptographic Service Facility (ICSF)
- 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.

PKCS #11 is an industry-accepted standard that provides an application programming interface (API) to devices, referred to as *tokens*, that hold cryptographic information and perform cryptographic functions. PKCS #11 provides an alternative to IBM®'s Common Cryptographic Architecture (CCA).

## Who should read this document

---

This document is primarily intended for application programmers who want to write PKCS #11 applications for z/OS. It also contains information for security administrators, system programmers, and auditors in installations that use PKCS #11 applications.

## How this document is organized

---

- Chapter 1, “Overview of z/OS support for PKCS #11,” on page 1 provides an overview of ICSF support for PKCS #11. It discusses tokens, the token data set (TKDS), auditing and tracing PKCS #11 functions, session objects, and tasks that must be performed before using PKCS #11 applications.
- Chapter 2, “The C API,” on page 19 discusses the PKCS #11 C API provided by ICSF, highlighting differences between the z/OS implementation and the PKCS #11 standard.
- Chapter 3, “Sample PKCS #11 C programs ,” on page 73 discusses how to build and run the testpkcs11 sample.
- Chapter 4, “Regional cryptographic servers,” on page 77 provides an introduction to the PKCS #11 extensions or mechanisms to be used with regional cryptographic servers.
- Chapter 5, “ICSF PKCS #11 callable services,” on page 89 provides a brief introduction to the PKCS #11 callable services, which are documented in *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

## How to use this document

---

Application programmers should read the entire book.

Security administrators should read the section “[Tasks for the security administrator](#)” on page 16 and the information that it references.

System programmers should read the section “[Tasks for the system programmer](#)” on page 16 and the information that it references.

Auditors should read the section “[Tasks for the auditor](#)” on page 17 and the information that it references.

## Where to find more information

---

Before using this document, application programmers must be familiar with the PKCS #11 specification. The PKCS #11 standard can be found at [PKCS#11: Cryptographic Token Interface Standard](#)

([www.cryptsoft.com/pkcs11doc](http://www.cryptsoft.com/pkcs11doc)). Application programmers should also be familiar with the ICSF library and C programming.

Security administrators should be familiar with *[z/OS Security Server RACF Security Administrator's Guide](#)*.

Auditors should be familiar with *[z/OS Security Server RACF Auditor's Guide](#)*.

The callable services for PKCS #11 functions are documented in *[z/OS Cryptographic Services ICSF Application Programmer's Guide](#)*.

The format of the token data set is documented in *[z/OS Cryptographic Services ICSF System Programmer's Guide](#)*.

## **IBM Crypto Education**

The [IBM Crypto Education](http://www.ibm.com/developerworks/community/groups/community/crypto) ([www.ibm.com/developerworks/community/groups/community/crypto](http://www.ibm.com/developerworks/community/groups/community/crypto)) community provides detailed explanations and samples pertaining to IBM cryptographic technology.

# How to send your comments to IBM

---

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

**Important:** If your comment regards a technical question or problem, see instead [“If you have a technical problem”](#) on page xi.

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

## **Feedback on z/OS function**

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

## **Feedback on IBM Knowledge Center function**

If your comment or question is about the IBM Knowledge Center functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Knowledge Center Support at [ibmkc@us.ibm.com](mailto:ibmkc@us.ibm.com).

## **Feedback on the z/OS product documentation and content**

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

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

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS ICSF Writing PKCS #11 Applications, SC14-7510-07
- The section title of the specific information to which your comment relates
- The text of your comment.

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

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

## **If you have a technical problem**

---

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

- Go to the [IBM Support Portal \(support.ibm.com\)](http://support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.



# Summary of changes

---

ICSF is an element of z/OS, but provides independent ICSF releases as web deliverables. These web deliverables are identified by their FMID. Each release of z/OS includes a particular ICSF FMID level as part of its base.

ICSF publications can be obtained from:

- The [Resource Link home page \(www.ibm.com/servers/resourcelink\)](http://www.ibm.com/servers/resourcelink). (Select Publications and then select the release that you are interested in under ICSF Publications by FMID.)
- [IBM z/OS downloads \(www.ibm.com/systems/z/os/zos/downloads\)](http://www.ibm.com/systems/z/os/zos/downloads) for Cryptographic Support.

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

## Changes made in Cryptographic Support for z/OS V2R2 - z/OS V2R4 (FMID HCR77D1)

---

This document contains information previously presented in *z/OS ICSF Writing PKCS #11 Applications*, SC14-7510-06.

This document is for ICSF FMID HCR77D1. This release of ICSF runs on z/OS V2R2, z/OS V2R3, and z/OS V2R4 and only on zSeries hardware.

The most recent updates are listed at the top of each section.

### New

#### Prior to the June 2020 refresh

- Information about IBM z15.
- [“Additional manifest constants for Dilithium quantum-safe algorithm support” on page 30](#)

### Changed

#### June 2020 refresh

- [“Key types and mechanisms supported” on page 20 \(APAR OA58880\)](#)

#### Prior to the June 2020 refresh

- [“Key types and mechanisms supported” on page 20 \(APAR OA58358\)](#)
- [“Objects and attributes supported” on page 30 \(APAR OA58358\)](#)
- [“Standard functions supported ” on page 51 \(APAR OA58358\)](#)
- [“PKCS #11 Coprocessor Access Control Points” on page 65 \(APAR OA58358\)](#)

### Deleted

No content was removed from this information.

## Changes made in Cryptographic Support for z/OS V2R2 - z/OS V2R3 (FMID HCR77D0)

---

This document contains information previously presented in *z/OS ICSF Writing PKCS #11 Applications*, SC14-7510-05.

This document is for ICSF FMID HCR77D0. This release of ICSF runs on z/OS V2R2 and z/OS V2R3 and only on zSeries hardware.

The most recent updates are listed at the top of each section.

### Changed

- [“Requiring signature verification for ICSF module CSFINPV2” on page 13](#)
- [“Key types and mechanisms supported” on page 20](#)
- [“Objects and attributes supported” on page 30](#)
- [“PKCS #11 Coprocessor Access Control Points” on page 65](#)
- [“Standard compliance modes” on page 69](#)

### Deleted

No content was removed from this information.

## Changes made in Cryptographic Support for z/OS V2R1 - z/OS V2R3 (FMID HCR77C1)

---

This document contains information previously presented in *z/OS ICSF Writing PKCS #11 Applications*, SC14-7510-04.

This document is for ICSF FMID HCR77C1. This release of ICSF runs on z/OS V2R1, V2R2, and V2R3 and only on zSeries hardware.

The most recent updates are listed at the top of each section.

### New

- Information about IBM z14 and IBM z14 ZR1.

### Changed

- [“Objects and attributes supported” on page 30 \(APAR OA54346\)](#).
- [“Key types and mechanisms supported” on page 20](#).

### Deleted

No content was removed from this information.

## Changes made in Cryptographic Support for z/OS V2R1 - z/OS V2R2 (FMID HCR77C0)

---

This document contains information previously presented in *z/OS ICSF Writing PKCS #11 Applications*, SC14-7510-03.

This document is for ICSF FMID HCR77C0. This release of ICSF runs on z/OS z/OS V2R1 and z/OS V2R2 and only on zSeries hardware.

The most recent updates are listed at the top of each section.

### **New**

- [“Optional Crypto Express adapters” on page 17](#) is new.

### **Changed**

- [“Objects and attributes supported” on page 30](#) has been updated (APAR OA54346).
- [“Regional cryptographic server key types and mechanisms supported” on page 77](#) has been updated.
- [Table 4 on page 21](#)
- [Table 5 on page 26](#)
- [Table 6 on page 29](#)
- [Table 28 on page 51](#)

### **Deleted**

No content was removed from this information.





---

# Chapter 1. Overview of z/OS support for PKCS #11

PKCS #11, also known as Cryptoki, is the cryptographic token interface standard. It specifies an application programming interface (API) to devices, referred to as *tokens*, that hold cryptographic information and perform cryptographic functions. The PKCS #11 API is an industry-accepted standard commonly used by cryptographic applications. ICSF supports PKCS #11, providing an alternative to IBM's Common Cryptographic Architecture (CCA) and broadening the scope of cryptographic applications that can make use of zSeries cryptography. PKCS #11 applications developed for other platforms can be recompiled and run on z/OS.

The PKCS #11 standard can be found at [PKCS#11: Cryptographic Token Interface Standard \(www.cryptsoft.com/pkcs11doc\)](http://www.cryptsoft.com/pkcs11doc). This document describes how ICSF supports that standard. The support includes the following:

- A token data set (TKDS) that serves as a repository for persistent cryptographic keys and certificates used by PKCS #11 applications.
- Instore memory that serves as a repository for temporary (session-only) cryptographic keys and certificates used by PKCS #11 applications.
- A C application programming interface (API) that supports a subset of the V2.20 level of the PKCS #11 specification
- PKCS #11 specific ICSF callable services. The C API uses these callable services.

---

## Tokens

On most single-user systems, a token is a smart card or other plug-installed cryptographic device, accessed through a card reader or *slot*. The PKCS #11 specification assigns numbers to slots, known as *slot IDs*. An application identifies the token that it wants to access by specifying the appropriate slot ID. On systems that have multiple slots, it is the application's responsibility to determine which slot to access.

z/OS must support multiple users, each potentially needing a unique key store. In this multiuser environment, the system does not give users direct access to the cryptographic cards installed as if they were personal smart cards. Instead, z/OS PKCS #11 tokens are virtual, conceptually similar to RACF (SAF) key rings. An application can have one or more z/OS PKCS #11 tokens, depending on its needs.

Typically, PKCS #11 tokens are created in a factory and initialized either before they are installed or upon their first use. In contrast, z/OS PKCS #11 tokens can be created using system software such as RACF, the `gskkyman` utility, or by applications using the C API. Each token has a unique token name, or label, that is specified by the end user or application at the time that the token is created.

**Rules:** A token name must follow these rules:

- Up to 32 characters in length
- Permitted characters are:
  - Alphanumeric
  - National: @ (X'5B'), # (X'7B'), or \$ (X'7C')
  - Period: . (X'4B')
- The first character must be alphabetic or national
- Lowercase letters can be used, but are folded to uppercase
- The IBM1047 code page is assumed

In addition to any tokens your installation may create, ICSF will also create a token that will be available to all applications. This "omnipresent" token is created by ICSF in order to enable PKCS #11 services when no other token has been created. This token supports session objects only. Session objects are

objects that do not persist beyond the life of a PKCS #11 session. The omnipresent token is always mapped to slot ID #0, and its token label is SYSTOK-SESSION-ONLY.

**Tip:** To reference the omnipresent token by label, use the constant `SESS_ONLY_TOK`, which is defined in `csnpdefs.h`.

Because PKCS #11 tokens are typically physical hardware devices, the PKCS #11 specification provides no mechanism to delete tokens. However, because z/OS PKCS #11 tokens are virtual, z/OS must provide a way to delete them. For information on how to delete tokens using the C API, see [“Deleting z/OS PKCS #11 tokens”](#) on page 19.

## Secure key PKCS #11

z/OS PKCS #11 supports two different keying models, Secure versus Clear. A secure key is one where the sensitive key material is always in wrapped form whenever it is outside the cryptographic device. The key is wrapped by using a master key that has been established in the cryptographic device and is not available in its entirety outside that device. A clear key does not have this extra protection. A clear key's sensitive key material appears in the virtual storage of ICSF in-the-clear and might even appear outside ICSF in-the-clear. Obviously, secure keys provide an extra layer of security. However, clear keys are more versatile than secure keys as they are not bound to any particular cryptographic device. They can even be used via software, when no cryptographic device is available.

The decision on whether to create a clear or secure key happens at the time the key is created. Absent any direction from the applications themselves (through new vendor-defined attributes), ICSF uses the context of the request along with a new RACF profile setting to decide. See [“Controlling token access and key policy”](#) on page 2.

z/OS PKCS #11 tokens can contain clear keys, secure keys, or a mixture of both. Secure keys require a secure coprocessor. Optional cryptographic hardware features can be configured as a cryptographic accelerator, a secure CCA cryptographic coprocessor, or a secure PKCS #11 cryptographic coprocessor. A secure PKCS #11 cryptographic coprocessor is also known as an Enterprise PKCS #11 coprocessor or EP11. An Enterprise PKCS #11 coprocessor with an active master key must be available to generate and use secure PKCS #11 keys. Clear keys do not require a coprocessor.

## The token data set (TKDS)

The token data set (TKDS) is a VSAM data set that serves as the repository for persistent cryptographic keys and certificates used by PKCS #11 applications. The system programmer creates the TKDS and updates the ICSF installation options data set to identify the data set name of the TKDS.

A TKDS is not required in order to run PKCS #11 applications. If ICSF is started without a TKDS, however, only the omnipresent token will be available.

A TKDS is required to utilize Secure Key PKCS #11.

**Rules:** The token data set must follow these rules:

- It must be a key-sequenced VSAM data set with spanned variable length records.
- It must be allocated on a permanently resident volume.

Clear keys in the token data set are not encrypted. Therefore, it is important that the security administrator create a RACF profile to protect the token data set from unauthorized access.

For the format of the TKDS, see 'Creating the TKDS' in [z/OS Cryptographic Services ICSF System Programmer's Guide](#).

To optimize performance, ICSF utilizes in-storage copy of the TKDS.

## Controlling token access and key policy

The PKCS #11 standard was designed for systems that grant access to token information based on a PIN. The standard defines two types of users, the standard user (*User*) and the security officer (*SO*), each having its own personal identification number (PIN). The SO can initialize a token (zero the contents) and set the User's PIN. The SO can also access the public objects on the token, but not the private ones. The

User has access to the private objects on a token and has the power to change his or her own PIN. The User cannot reinitialize a token. The PIN that a user enters determines which role that user takes. A user can fill both roles by having knowledge of both PINs.

z/OS does not use PINs. Instead, profiles in the SAF CRYPTOZ class control access to tokens. For each token, there are two resources in the CRYPTOZ class for controlling access to tokens:

- The resource `USER.token-name` controls the access of the User role to the token.
- The resource `SO.token-name` controls the access of the SO role to the token.

A user's access level to each of these resources (read, update, or control) determines the user's access level to the token.

There are six possible token access levels. Three are defined by the PKCS #11 standard, and three are unique to z/OS. The PKCS #11 token access levels are:

- User R/O: Allows the user to read the token including its private objects, but the user cannot create new token or session objects or alter existing ones.
- User R/W: Allows the user read/write access to the token object including its private objects.
- SO R/W: Allows the user to act as the security officer for the token and to read, create, and alter public objects on the token.

The token access levels unique to z/OS are:

- Weak SO: A security officer that can modify the CA certificates contained in a token but not initialize the token. (For example, a system administrator who determines the trust policy for all applications on the system.)
- Strong SO: A security officer that can add, generate or remove private objects in a token. (For example, a server administrator.)
- Weak User: A User that cannot change the trusted CAs contained in a token. (For example, to prevent an end-user from changing the trust policy of his or her token.)

Table 1 on page 3 shows how a user's access level to a token is derived from the user's access level to a resource in the SAF CRYPTOZ class.

<i>Table 1. Token access levels</i>			
<b>CRYPTOZ resource</b>	<b>READ (SAF access level)</b>	<b>UPDATE (SAF access level)</b>	<b>CONTROL (SAF access level)</b>
<b>SO.token-label</b>	Weak SO Can read, create, delete, modify, and use public objects	SO R/W Same ability as Weak SO plus can create and delete tokens	Strong SO Same ability as SO R/W plus can read but not use (see Note "2" on page 4) private objects; create, delete, and modify private objects
<b>USER.token-label</b>	User R/O Can read and use (see Note "2" on page 4) public and private objects	Weak User Same ability as User R/O plus can create, delete, and modify private and public objects. Cannot add, delete, or modify certificate authority objects	User R/W Same ability as Weak User plus can add, delete, and modify certificate authority objects

**Note:**

1. The *USER.token-name* and *SO.token-name* profiles will **not** be checked to determine access to the omnipresent token SYSTOK-SESSION-ONLY. ICSF creates this token to provide PKCS #11 support even if no other token is available to an application. All users will always be considered to have R/W access to this token.

2. "Use" is defined as any of the following:

- Performing any cryptographic operation involving the key object; for example C\_Encrypt
- Searching for key objects using sensitive search attributes
- Retrieving sensitive key object attributes.

The sensitive attribute for a secret key is CKA\_VALUE. The sensitive attribute for Diffie Hellman, DSA, and Elliptic Curve private key objects is CKA\_VALUE. The sensitive attributes for RSA private key objects are CKA\_PRIVATE\_EXPONENT, CKA\_PRIME\_1, CKA\_PRIME\_2, CKA\_EXPONENT\_1, CKA\_EXPONENT\_2, and CKA\_COEFFICIENT.

3. The CRYPTOZ resources can be defined as "RACF-DELEGATED" if required. For information about delegated resources, see the topic on delegated resources in *z/OS Security Server RACF Security Administrator's Guide*.

4. Although the use of generic profiles in the CRYPTOZ class is permitted, you should not use a single generic profile to cover both the *SO.token-label* and *USER.token-label* resources. You should not do this, because there are additional resources in the class controlling key policy. (See Guidelines in this topic for FIPSEXEMPT.token-label and CLEARKEY.token-label.) Creating a generic profile that uses generic characters to match both the SO and USER portion of the resource name (for example \*.token-label) will also inadvertently match these other resources and can have unintended consequences.

5. If the CSFSERV class is active, ICSF performs access control checks on the underlying callable services. The user must have READ access to the appropriate CSFSERV class resource. [Table 2 on page 4](#) lists the resources in the CSFSERV class for token services.

Name of resource	Service	Called by
<b>CSF1TRC</b>	Token or object creation	C_InitToken, C_CreateObject, C_CopyObject
<b>CSF1TRD</b>	Token or object deletion	C_InitToken, C_DestroyObject
<b>CSF1TRL</b>	Token or object find	C_Initialize, C_FindObjects, CSN_FindAllObjects
<b>CSF1SAV</b>	Set object attributes	C_SetAttributeValue
<b>CSF1GAV</b>	Get object attributes	C_GetAttributeValue
<b>CSF1GSK</b>	Generate secret key	C_GenerateKey
<b>CSF1GKP</b>	Generate key pair	C_GenerateKeyPair
<b>CSF1PKS</b>	Private key sign	C_Decrypt, C_DecryptUpdate, C_DecryptFinal, C_Sign, C_SignFinal
<b>CSF1PKV</b>	Public key verify	C_Encrypt, C_EncryptUpdate, C_EncryptFinal, C_Verify, C_VerifyFinal
<b>CSF1SKD</b>	Secret key decrypt	C_Decrypt, C_DecryptUpdate, C_DecryptFinal

Table 2. Resources in the CSFSERV class for token services (continued)

Name of resource	Service	Called by
<b>CSF1SKE</b>	Secret key encrypt	C_Encrypt, C_EncryptUpdate, C_EncryptFinal
<b>CSFOWH</b>	One-way hash	C_Digest, C_DigestUpdate, C_DigestFinal, C_Sign, C_SignUpdate, C_SignFinal, C_Verify, C_VerifyUpdate, C_VerifyFinal
<b>CSF1WPK</b>	Wrap key	C_WrapKey
<b>CSF1UWK</b>	Unwrap key	C_UnwrapKey
<b>CSF1HMG</b>	Generate MAC	C_Sign
<b>CSF1HMV</b>	Verify MAC	C_Verify
<b>CSF1DVK</b>	Derive key	C_DeriveKey
<b>CSF1DMK</b>	Derive multiple keys	C_DeriveKey
<b>CSFIQA</b>	PKCS #11 initialization	C_Initialize
<b>CSFRNG</b>	Random number generate	C_GenerateRandom

**Guidelines:**

1. If your organization controls access to ICSF callable services using the CSFSERV class, define the resources listed in [Table 2 on page 4](#) and grant access accordingly.  
  
**Tip:** Define generic profiles. For example, a profile named CSF\* covers all the ICSF services. A profile named CSF1\* covers the PKCS #11 subset of the ICSF services, with the exception of those covered by the CSFOWH, CSFIQF, and CSFRNG resources.
2. The CRYPTOZ class supports generic profiles. Take advantage of this by creating a token naming convention for your organization and enforce it with generic profiles. For example, require users and applications to prefix their token names with their user IDs, as with data set names. (See [“Sample scenario for setting up z/OS PKCS #11 tokens” on page 7.](#))
3. For server applications, grant security officers (server administrators) Strong SO access and their end-users (server daemon user IDs) Weak User or User R/W access.
4. For applications for which you do not wish to separate the security officer and end-user roles, grant the appropriate user IDs access to both the SO and USER profiles.

In addition to these two resources for controlling access to tokens, each token also has two additional resources in the CRYPTOZ class: *FIPSEXEMPT.token-name* and *CLEARKEY.token-name*.

The *FIPSEXEMPT.token-name* resource is used for identifying applications that are subject to FIPS 140 restrictions when ICSF is running in FIPS compatibility mode. Refer to [“Operating in compliance with FIPS 140-2” on page 10](#) for more information.

The *CLEARKEY.token-name* resource will be queried to determine the policy for creating a clear in contrast to a secure key when CKA\_IBM\_SECURE=TRUE has not been specified for key generation. The following table indicates the significance of the different access levels. When there is no matching profile defined, the row indicating RACF access of UPDATE or No Decision is considered, the policy is to base the decision on the key’s sensitivity and whether an Enterprise PKCS #11 coprocessor is available or not.

Table 3. CLEARKEY.token-label resource access and key security policy

Key Security Objective	RACF ACCESS	Action taken when PKCS #11 coprocessor not available or algorithm not supported	Action taken when PKCS #11 coprocessor available and algorithm supported
Generate no secure keys. Stay compatible with earlier releases	CONTROL	Sensitive - Clear Key Non-sensitive - Clear Key	Sensitive - Clear Key Non-sensitive - Clear Key
Use key sensitivity and environment to determine security	UPDATE or No Decision	Sensitive - Clear Key Non-sensitive - Clear Key	Sensitive - Secure Key Non-sensitive - Clear Key
Ensure keys explicitly marked sensitive are always secure keys	READ	Sensitive - Denied Non-sensitive - Clear Key	Sensitive - Secure Key Non-sensitive - Clear Key
Prevent generation or creation of any clear keys	NONE	Sensitive - Denied Non-sensitive - Denied	Sensitive - Secure Key Non-sensitive - Secure Key

**Service specific notes:**

1. For generate key and generate key-pair, CLEARKEY.token-label checking is always performed as described previously.
2. For create object, no CLEARKEY.token-label checking is performed. By default, all keys created via create object are clear keys. To get an encrypted key, the caller must specify CKA\_IBM\_SECURE=TRUE. Such keys are not true secure keys since the sensitive key material has appeared in-the-clear outside the bounds of the secure coprocessor.
3. For copy object, no CLEARKEY.token-label checking is performed. By default, the source key's security is carried forward to the target key. Clear keys may be upgraded to encrypted keys by specifying CKA\_IBM\_SECURE=TRUE.
4. For unwrap key, the security of the base key always determines the security of the unwrapped key. However, in the case of clear key unwrap, CLEARKEY.token-label checking is performed to see if clear keys are permitted.
5. For derive key, the base key can be clear or secure. The resulting derived key will be clear. CLEARKEY.token-label checking is performed to see if clear keys are permitted.

**General notes:**

1. You should avoid setting a discrete or generic profile that would restrict clear key creation in the omnipresent token. This token is used by other z/OS components to create session keys only. It is typical for session keys to be clear keys. The clear key resource checked for the omnipresent token is CLEARKEY.SYSTOK-SESSION-ONLY.
2. If no CLEARKEY profile is created to protect a given token, the No Decision row governs the action taken for that token for key creation and generation requests. The action taken depends on whether a PKCS #11 coprocessor is active or not. If no PKCS #11 coprocessor is active, all key creation and generate requests result in clear keys. If a PKCS #11 coprocessor is made active, sensitive keys (CKA\_SENSITIVE=TRUE) may be generated as secure keys, depending on the algorithm. This could have an unexpected effect on existing programs.

## Managing tokens

z/OS provides several facilities to manage tokens:

- A C language application programming interface (API) that implements a subset of the PKCS #11 specification. For a description of this API, see [Chapter 2, “The C API,” on page 19](#).
- PKCS #11 specific ICSF callable services. The C API uses these callable services. For information about these callable services, see [Chapter 5, “ICSF PKCS #11 callable services,” on page 89](#).
- ISPF panels. The ICSF ISPF panels provide the capability to see a formatted view of TKDS objects, and make limited updates to them.
- The RACF RACDCERT command supports the certificate, public key, and private key objects, and provides the following subfunctions to manage these objects:
  - ADDTOKEN - creates a new empty token
  - DELTOKEN - deletes an existing token and everything in it
  - LISTTOKEN - displays information on the certificate objects in a token and whether associated public and private key objects exist
  - BIND - connects a RACF certificate, its public key, and potentially its private key to an existing token
  - UNBIND - removes a certificate and its keys from a token
  - IMPORT - defines a token certificate to RACF

For information about the RACDCERT command, see [z/OS Security Server RACF Command Language Reference](#) and [z/OS Security Server RACF Security Administrator's Guide](#).

- The SAF CRYPTOZ class controls access to tokens. For information about this class, see [“Controlling token access and key policy” on page 2](#).
- The RACF R\_Datalib callable service (IRRSDL00) allows applications to read tokens by providing a user ID of \*TOKEN\* to indicate that the key ring name is really a token name. For information about R\_Datalib, see [z/OS Security Server RACF Callable Services](#).

**Note:** IRRSDL00 was originally created to allow applications to read RACF (SAF) key rings, but has been enhanced to read PKCS #11 tokens as well. Thus applications written to read key rings can also read tokens without being modified.

## Sample scenario for setting up z/OS PKCS #11 tokens

The following examples show how to control access to z/OS PKCS #11 tokens. In this scenario, a company wants to use z/OS PKCS #11 tokens as the key stores for its FTP and Web servers. The company has established a naming convention for their tokens requiring that all tokens have the owning user ID as the high-level qualifier. The owning user IDs for the FTP and Web server tokens are the daemons FTPSRV and WEBSRV, respectively. User ABIGAIL is the administrator for the servers.

The security administrator, who has the RACF SPECIAL attribute, creates the protection profiles for the tokens. The security administrator's goal is to give user ABIGAIL the Security Officer role for these profiles, and to give the daemon user IDs the User role. To do this, the security administrator issues RACF TSO commands. First, the security administrator activates the CRYPTOZ class with generics and RACLISTS it:

```
SETROPTS CLASSACT(CRYPTOZ) GENERIC(CRYPTOZ) RACLIST(CRYPTOZ)
```

Next, the security administrator creates profiles for the security officer's access to the FTP and Web Server tokens:

```
RDEFINE CRYPTOZ SO.FTPSRV.* UACC(NONE)  
RDEFINE CRYPTOZ SO.WEBSRV.* UACC(NONE)
```

Then, the security administrator creates profiles for the standard user's access to the FTP and Web Server tokens:

```
RDEFINE CRYPTOZ USER.FTPSRV.* UACC(NONE)
RDEFINE CRYPTOZ USER.WEBSRV.* UACC(NONE)
```

The security administrator now gives user ABIGAIL Strong SO power for the tokens by giving her CONTROL access to the profiles that protect the tokens. The Strong SO power does not allow ABIGAIL to use the private objects in the tokens:

```
PERMIT SO.FTPSRV.* CLASS(CRYPTOZ) ID(ABIGAIL) ACC(CONTROL)
PERMIT SO.WEBSRV.* CLASS(CRYPTOZ) ID(ABIGAIL) ACC(CONTROL)
```

Next, the security administrator gives the users FTPSRV and WEBSRV Weak User power for their respective tokens. This power allows them to use the private objects within the tokens, but not change the set of trusted CA certificates.

```
PERMIT USER.FTPSRV.* CLASS(CRYPTOZ) ID(FTPSRV) ACC(UPDATE)
PERMIT USER.WEBSRV.* CLASS(CRYPTOZ) ID(WEBSRV) ACC(UPDATE)
```

Finally, the security administrator refreshes the in-storage profiles for the CRYPTOZ class, so that the changes he just made take effect:

```
SETROPTS RACLIST(CRYPTOZ) REFRESH
```

Now the set up is complete: ABIGAIL has Strong SO power over the tokens for the FTP server and the Web server, and can create the required tokens. FTPSRV and WEBSRV have User power over their respective tokens, and can use them as key stores after ABIGAIL has created them.

The task now is to create and populate the tokens for the servers with RACF certificates. The following certificates exist:

1. A root CA certificate installed under CERTAUTH with label 'Local Root CA for Servers'.
2. An end-entity certificate and private key installed under user FTPSRV with label 'FTP Key'. This certificate was signed by the first certificate.
3. An end-entity certificate and private key installed under user WEBSRV with label 'Web Key'. This certificate was also signed by the first certificate.

User ABIGAIL issues the following TSO commands to create the tokens, using the company's naming conventions:

```
RACDCERT ADDTOKEN(ftpsrv.ftp.server.pkcs11.token)
RACDCERT ADDTOKEN(websrv.web.server.pkcs11.token)
```

Next, issue the commands that bind the root CA certificate to the two tokens:

```
RACDCERT BIND(CERTAUTH LABEL('Local Root CA for Servers'))
TOKEN(ftpsrv.ftp.server.pkcs11.token)
RACDCERT BIND(CERTAUTH LABEL('Local Root CA for Servers'))
TOKEN(websrv.web.server.pkcs11.token)
```

Now, bind the end-entity certificates to their respective tokens. Each should be the default in the token.

```
RACDCERT BIND(ID(FTPSRV) LABEL("FTP key"))
TOKEN(ftpsrv.ftp.server.pkcs11.token) DEFAULT)
RACDCERT BIND(ID(WEBSRV) LABEL("Web key"))
TOKEN(websrv.web.server.pkcs11.token) DEFAULT)
```

The final step is for the user (ABIGAIL) to configure both servers to use their respective tokens: add directives to the servers' configuration files.

For the web server (IBM HTTP Server), the keyfile directive in the httpd.conf file is set as follows:

```
keyfile *TOKEN*/WEBSRV.WEB.SERVER.PKCS11.TOKEN SAF
```

The SAF keyword indicates to SSL that this is a key ring and is controlled by SAF; it is not a KDB file. The TOKEN keyword indicates that the key ring is a token. The FTP server configuration file also requires a token-qualified key ring name:



```
keyfile *TOKEN*/FTPSRV.FTP.SERVER.PKCS11.TOKEN
```

## Sample scenario for controlling clear key processing

The following examples show how the RACF administrator will use the new CRYPTOZ resource, **CLEARKEY.token-label**, to set policy on the use of clear keys.

In this scenario, company XYZ wishes to use the ABC program. The ABC program will be creating session keys and be using the system level token named 'SYSTOK-SESSION-ONLY' for cryptographic operations. Company XYZ wants to ensure that all keys created by the ABC program are secure keys. The user ID assigned to the ABC program is ABCUSER.

Company XYZ also has other applications that use the system level token for cryptographic operations. These applications should not be restricted to using only secure keys.

User RACFADM, who has the RACF SPECIAL, creates the profiles necessary by issuing the following RACF TSO commands:

1. Activate the CRYPTOZ class with generics and RACLIST it:

```
SETROPTS CLASSACT(CRYPTOZ) GENERIC(CRYPTOZ) RACLIST(CRYPTOZ)
```

2. Create the CLEARKEY profile for the system level token:

```
RDEF CRYPTOZ CLEARKEY.SYSTOK-SESSION-ONLY UACC(NONE)
```

3. Restrict user ID ABCUSER to secure keys only:

```
PERMIT CLEARKEY.SYSTOK-SESSION-ONLY CLASS(CRYPTOZ) ID(ABCUSER) ACC(NONE)
```

4. Allow all other user IDs to create clear keys – normal mode:

```
PERMIT CLEARKEY.SYSTOK-SESSION-ONLY CLASS(CRYPTOZ) ID(*) ACC(UPDATE)
```

5. Refresh the RACLIST to pick up the changes:

```
SETROPTS RACLIST(CRYPTOZ) REFRESH
```

## Auditing PKCS #11 functions

PKCS #11 functions are audited in the following ways:

- The SMF type 82 subtype 1 record that is written during ICSF initialization contains the data set name of the token data set (TKDS).
- The SMF type 82 subtype 21 record that is written when a member joins or leaves a sysplex group contains the cryptographic keys data set (CKDS) data set name if the member joined or left the ICSF CKDS sysplex group, or the TKDS data set name if the member joined or left the ICSF TKDS sysplex group.
- ICSF writes SMF type 82 subtype 23 records whenever a TKDS record for a token or token object is created, modified, or deleted. ICSF does not write SMF records for changes to session objects.

For descriptions of the SMF records that ICSF writes, see [z/OS MVS System Management Facilities \(SMF\)](#).

## Component trace for PKCS #11 functions

The following ICSF component trace entries trace events related to the token data set (TKDS):

- Type 16 (XCFTMSGS) traces the broadcast of an XCF message related to TKDS I/O.
- Type 17 (XCFTMSGR) traces the receipt of an XCF message related to TKDS I/O.
- Type 18 (XCFTENQ) traces the return of control to the TKDS I/O subtask following the request for an exclusive ENQ on the SYSZTKDS.TKDSdsn resource.

These trace entry types are always traced.

When viewed via IPCS, these entries show the ASCB address, the TCB address, the ASID, the general purpose registers, the GPR length, and the CSS address. For more information about IPCS, see [z/OS MVS IPCS User's Guide](#).

## Object types

---

ICSF supports PKCS #11 session objects and token objects. The following classes of objects can be associated with these object types:

- Certificate
- Public key
- Private key
- Secret key
- Data objects
- Domain parameters

## Session objects

A session object exists for the life of a PKCS #11 session. ICSF allocates session object memory areas to hold session objects; they are not maintained on DASD. ICSF associates a session object memory area with the application that requested the creation of a session object. There is only one session object memory area for an application, even if the application spawns multiple PKCS #11 sessions. The same session objects are available to all PKCS #11 sessions within an application.

ICSF creates a session object memory area the first time a session object is created, if there is currently no session object memory area associated with the application. The session object memory area exists as long as the PKCS #11 application's address space and job step TCB exist. ICSF deletes the memory area if either the address space or job step TCB terminates. If ICSF terminates, all session object memory areas are destroyed.

ICSF creates one session-object token, the omnipresent token, to provide PKCS #11 support even if no other token is available to an application. For example, no other token is available when a TKDS is not identified using the TKDSN option in the ICSF installation options data set, or when the SAF CRYPTOZ class has not been activated. This session object token (labeled SYSTOK-SESSION-ONLY) is write protected, cannot be used to store persistent attributes, and cannot be deleted.

On z/OS, an application can be running in either single address space mode, or in cross memory mode. The PKCS #11 standard has no concept of cross memory mode, so there is no predefined expected behavior for a PKCS #11 application running in cross memory mode. If running in cross memory mode, you should be aware of the guidelines pertaining to session objects described in [“Cross memory considerations”](#) on page 20.

## Token objects

Token objects are stored in the token data set, with one record per object. They are visible to all applications that have sufficient permission to the token. They are persistent: they remain associated with the token even after a session is closed.

## Operating in compliance with FIPS 140-2

---

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

z/OS PKCS #11 cryptography is designed to meet FIPS 140-2 Level 1 criteria, and can be configured to operate in compliance with FIPS 140-2 specifications. Applications that need to comply with the FIPS 140-2 standard can therefore 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. There are three modes of FIPS operation:

- The services can be configured so that all z/OS PKCS #11 applications are forced to comply with the FIPS 140-2 standard. This is called *FIPS standard mode*.
- For installations where only certain z/OS PKCS #11 applications need to comply with the FIPS 140-2 standard, the services can be configured so that only the necessary applications are restricted from using the non-approved algorithms and key sizes, while other applications are not. This is called *FIPS compatibility mode*. You can also use FIPS compatibility mode to test individual applications to ensure FIPS compliance before switching to FIPS standard mode.
- In *FIPS no enforcement mode*, ICSF will not impose FIPS algorithm or key size restrictions unless the calling application explicitly requests it.

ICSF installation options are described in the *z/OS Cryptographic Services ICSF System Programmer's Guide*. The installation option FIPSMODE indicates one of the following:

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

Indicates that ICSF is to operate in FIPS standard mode - all applications that call ICSF PKCS #11 services will have a need to run in FIPS 140-2 compliant fashion. Therefore, ICSF is to honor FIPS 140 restrictions pertaining to PKCS #11 algorithms and keys for all applications that call ICSF PKCS #11 services.

ICSF initialization will test that it is running on an IBM Z<sup>®</sup> model type and version/release of z/OS that supports FIPS. If so, ICSF initialization will also perform a series of cryptographic known answer self tests. Should a test fail, the action ICSF initialization takes is dependent on the fail option:

**FIPSMODE(YES, FAIL(YES))**

Indicates ICSF is to terminate abnormally if there is a failure in any of the tests performed.

**FIPSMODE(YES, FAIL(NO))**

Indicates ICSF initialization processing is to continue even if there is a failure in any of the tests performed. However, PKCS #11 support will be limited or nonexistent depending on the test that failed:

- If ICSF is running on an IBM Z model type or with a version/release 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 or those requests that explicitly ask for FIPS processing will result in a failure return code.
- If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable.

**FIPSMODE(COMPAT, FAIL(fail-option))**

Indicates that ICSF is to operate in FIPS compatibility mode - some selected applications that call ICSF PKCS #11 services will have a need to run in FIPS 140-2 compliant fashion while others may not be so restricted.

ICSF initialization will test that it is running on an IBM Z model type and version/release of z/OS that supports FIPS. If so, ICSF initialization will also perform a series of cryptographic known answer self tests. Should a test fail, the action ICSF initialization takes is dependent on the fail option:

**FIPSMODE(COMPAT, FAIL(YES))**

Indicates ICSF is to terminate abnormally if there is a failure in any of the tests performed.

**FIPSMODE(COMPAT, FAIL(NO))**

Indicates ICSF initialization processing is to continue even if there is a failure in any of the tests performed. However, PKCS #11 support will be limited or nonexistent depending on the test that failed:

- If ICSF is running on an IBM Z model type or with a version/release 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 or those requests that explicitly ask for FIPS processing will result in a failure return code.

- If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable.

Assuming all tests are successful, ICSF will honor FIPS 140 restrictions pertaining to PKCS #11 algorithms and keys for selected applications that call ICSF PKCS #11 services. This selection process is enabled by the following function:

- New CRYPTOZ Class resource profiles will allow the customer to control the scope of FIPS 140-2 compliance at the token level. The naming convention for these resources is:

**FIPSEXEMPT.token-name**

The levels of access are as follows:

- Profile not defined or Profile defined, caller has access NONE – User not exempt. Algorithms and key sizes restricted as per FIPS 140-2. For example: usage is treated as if ICSF was started with FIPSMODE(YES, FAIL(*fail-option*)).
- Profile defined, caller has access READ – User's use of the token is exempt from the FIPS 140-2 algorithm restrictions.

A new vendor defined Boolean key attribute is now supported, CKA\_IBM\_FIPS140. Applications may explicitly set this at the time the key is created. The default value is FALSE. If set to TRUE, ICSF will ensure that the key is only used in a FIPS 140-2 compliant fashion, and treated as if FIPSEXEMPT.token-name access NONE was specified.

**Note:** If CKA\_IBM\_FIPS140 is specified as a key generation attribute, this would include the generation of the key as well.

**FIPSMODE(NO,FAIL(*fail-option*))**

Indicates that ICSF should operate in FIPS no enforcement mode, also known as FIPS on-demand mode. Applications may request strict adherence to FIPS 140 restrictions when requesting ICSF services. However, applications not requesting FIPS processing are not required to adhere to FIPS 140 restrictions. FIPSEXEMPT.token-name profiles, if they exist, will not be examined. If ICSF is running on an IBM Z model type that does not support FIPS, requests to generate or use a key with CKA\_IBM\_FIPS140=TRUE or those requests that explicitly ask for FIPS processing will result in a failure return code.

ICSF initialization will test that it is running on an IBM Z model type and version/release of z/OS that supports FIPS. If so, ICSF initialization will also perform a series of cryptographic known answer self tests. Should a test fail, the action ICSF initialization takes is dependent on the fail option:

**FIPSMODE(NO, FAIL(YES))**

Indicates ICSF is to terminate abnormally if there is a failure in any of the tests performed.

**FIPSMODE(NO, FAIL(NO))**

Indicates ICSF initialization processing is to continue even if there is a failure in any of the tests performed. However, PKCS #11 support will be limited or nonexistent depending on the test that failed:

- If ICSF is running on an IBM Z model type or with a version/release 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 or explicitly ask for FIPS processing will result in a failure return code.
- If a known answer test failed, all ICSF PKCS #11 callable services will be unavailable.

If the FIPSMODE option is not specified, FIPSMODE(NO, FAIL(NO)) is the default.

If any z/OS PKCS #11 application intends to use the services in compliance with the FIPS 140-2 standard, then, in accordance with that standard, the integrity of the load module containing the z/OS PKCS #11 services must be checked when ICSF is started. This load module is digitally signed, and, in order for applications using its services to be FIPS 140-2 compliant, the signature must be verified when

ICSF is started. For more information, refer to [“Requiring signature verification for ICSF module CSFINPV2”](#) on page 13.

If any application will use PKCS #11 objects for AES Galois/Counter Mode (GCM) encryption or GMAC generation, and will have ICSF generate the initialization vectors, then you need to set ECVTSPLX or CVTSNAME to a unique value. Refer to [z/OS Cryptographic Services ICSF System Programmer's Guide](#) for more information.

## Requiring signature verification for ICSF module CSFINPV2

If your installation needs to operate z/OS PKCS #11 in compliance with the FIPS 140-2 standard, then the integrity of the cryptographic functions shipped by IBM must be verified at your installation during ICSF startup. The load module that contains the software cryptographic functions is SYS1.SIEALNKE(CSFINPV2), and this load module is digitally signed when it is shipped from IBM. Using RACF, you can verify that the module has remained unchanged from the time it was built and installed on your system. To do this, you create a profile in the PROGRAM class for the CSFINPV2 module, and use this profile to indicate that signature verification is required before the module can be loaded.

To require signature verification for ICSF module CSFINPV2:

1. Make sure that RACF has been prepared to verify signed programs. As described in [z/OS Security Server RACF Security Administrator's Guide](#), a security administrator prepares RACF to verify signed programs by creating a key ring for signature verification, and adding the code-signing CA certificate that is supplied with RACF to the key ring. If RACF has been prepared to verify signed programs, there will be a key ring dedicated to signature verification, the code-signing CA certificate will be attached to the key ring, and the PROGRAM class will be active.

- a. If RACF has been prepared to verify signed programs, the discrete profile IRR.PROGRAM.SIGNATURE.VERIFICATION in the FACILITY class will specify the name of the signature-verification key ring. To determine if a signature key ring is already active, enter the command:

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

If there is no discrete profile with this name, have your security administrator prepare RACF to verify signed programs using the information in [z/OS Security Server RACF Security Administrator's Guide](#).

- b. If the signature verification key ring exists, the RLIST command will display information for the discrete profile IRR.PROGRAM.SIGNATURE.VERIFICATION in the FACILITY class. The name of the signature verification key ring and the name of the key ring owner will be included in the APPLICATION DATA field of the RLIST command output. Using this information, enter the RACDCERT LISTRING command to make sure the code-signing CA certificate is attached to the key ring:

```
RACDCERT ID(key-ring-owner) LISTRING(key-ring-name)
```

The label of the code-signing CA certificate is 'STG Code Signing CA - G2'. If this label is not shown in the RACDCERT LISTRING command output, have your security administrator prepare RACF to verify signed programs using the information in [z/OS Security Server RACF Security Administrator's Guide](#).

- c. Program control must be active in order for RACF to perform signature verification processing. To make sure the PROGRAM class is active, enter the SETROPTS LIST command.

```
SETROPTS LIST
```

The ACTIVE CLASSES field of the command output should include the PROGRAM class. If it does not, have your security administrator prepare RACF to verify signed programs using the information in [z/OS Security Server RACF Security Administrator's Guide](#).

2. Create a profile for the CSFINPV2 program module in the PROGRAM class, indicating that the program must be signed. The following command specifies that the program should fail to load if the signature

cannot be verified for any reason. This command also specifies that all signature verification failures should be logged.

**Note:** Due to space constraints, this command example appears on two lines. However, the RDEFINE command should be entered completely on one line.

```
RDEFINE PROGRAM CSFINPV2 ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))
```

You will need to activate your profile changes in the PROGRAM class.

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

## Requiring FIPS 140-2 compliance from all z/OS PKCS #11 applications

If all z/OS PKCS #11 applications running on your system must comply with the FIPS 140-2 standard, your installation's system programmer should configure ICSF so that z/OS PKCS #11 operates in FIPS standard mode. To do this:

1. Make sure the integrity of the cryptographic functions shipped by IBM in the ICSF module CSFINPV2 will be verified by RACF before the module is loaded. This is done by following the instructions in “Requiring signature verification for ICSF module CSFINPV2” on page 13. If these steps are not followed to verify the digital signature of the module, no application calling the z/OS PKCS #11 services can be considered FIPS 140-2 compliant.
2. To specify FIPS standard mode, have your installation's system programmer include the installation option FIPSMODE(YES, FAIL(*fail-option*)) in the ICSF installation options data set.

When this option is used, ICSF will operate in FIPS standard mode. In this mode, ICSF initialization will test that it is running on an IBM 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 the tests succeed, then all applications calling z/OS PKCS services will be restricted from using the PKCS #11 algorithms and key sizes that are prohibited by the FIPS 140-2 standard (as outlined in [Table 6 on page 29](#)).

If any of the installation tests should fail, the action ICSF initialization takes depends on the *fail-option* specified. The *fail-option* within the FIPSMODE(YES, FAIL(*fail-option*)) installation option can be either:

- YES (which indicates that ICSF should terminate abnormally if there is a failure in any of the tests that are performed).
- NO (which indicates that ICSF initialization processing should continue even if there is a failure in one or more of the tests that are performed). If an initialization test does fail, however, PKCS #11 support will be limited or nonexistent depending on the test that failed.
  - If ICSF is running on an IBM 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.

For more information on this or other ICSF installation options, refer to [z/OS Cryptographic Services ICSF System Programmer's Guide](#).

## Requiring FIPS 140-2 compliance from select z/OS PKCS #11 applications

If only certain z/OS PKCS #11 applications running on your system must comply with the FIPS 140-2 standard, while other z/OS PKCS #11 applications do not, your system programmer should configure ICSF so that z/OS PKCS #11 operates in FIPS compatibility mode. In FIPS compatibility mode, you can use resource profiles in the CRYPTOZ class to specify, at a token level, the applications that are exempt from FIPS 140-2 compliance and, for that reason, should not be subject to FIPS restrictions. To configure the z/OS PKCS #11 services to operate in FIPS compatibility mode:



1. Make sure the integrity of the cryptographic functions shipped by IBM in the module ICSF module CSFINPV2 will be verified by RACF before the module is loaded. This is done by following the instructions in “[Requiring signature verification for ICSF module CSFINPV2](#)” on page 13. If these steps are not followed to verify the digital signature of the module, no application calling the z/OS PKCS #11 services can be considered FIPS 140-2 compliant.
2. To specify FIPS compatibility mode, have your installation's system programmer include the installation option FIPSMODE(COMPAT, FAIL(*fail-option*)) in the ICSF installation options data set.

When this option is used, ICSF will operate in FIPS compatibility mode. In this mode, ICSF initialization will test that it is running on an IBM 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 the tests are successful, then, by default, all applications calling z/OS PKCS services will be restricted from using the PKCS #11 algorithms and key sizes that are prohibited by the FIPS 140-2 standard (as outlined in [Table 6 on page 29](#)). Using profiles in the CRYPTOZ class, however, you can identify applications that are exempt from FIPS 140-2 compliance (as described in the next step).

If any of the installation tests should fail, the action ICSF initialization takes depends on the *fail-option* specified. The *fail-option* within the FIPSMODE(COMPAT, FAIL(*fail-option*)) installation option can be either:

- YES (which indicates that ICSF should terminate abnormally if there is a failure in any of the tests that are performed).
- NO (which indicates that ICSF initialization processing should continue even if there is a failure in one or more of the tests that are performed). If an initialization test does fail, however, PKCS #11 support will be limited or nonexistent depending on the test that failed.
  - If ICSF is running on an IBM 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.

For more information on this and other ICSF installation options, refer to [z/OS Cryptographic Services ICSF System Programmer's Guide](#).

3. To specify which applications must comply with FIPS 140-2 restrictions and which applications do not need to comply, create FIPSEXEMPT.*token-label* resource profiles in the CRYPTOZ class. If no FIPSEXEMPT.*token-label* resource profiles are created, then all z/OS PKCS #11 applications will be subject to FIPS restrictions. By creating a FIPSEXEMPT.*token-label* resource profile for a particular token, however, you can specify whether or not a particular user ID should be considered exempt from FIPS restrictions when using that token.

- If a user ID has access authority NONE to the FIPSEXEMPT.*token-label* resource, ICSF will enforce FIPS 140-2 compliance for that user ID.
- If a user ID has access authority READ to the FIPSEXEMPT.*token-label* resource, that user ID is exempt from FIPS 140-2 restrictions.

To specify which applications must comply with the FIPS 140-2 restrictions, and which do not, the security administrator must:

- a. If it is not already activated, activate the CRYPTOZ class with generics and RACLIST it:

```
SETROPTS CLASSACT(CRYPTOZ) GENERIC(CRYPTOZ) RACLIST(CRYPTOZ)
```

- b. Create the FIPSEXEMPT.*token-label* resource profile for each z/OS PKCS #11 token. The following command creates the profile for the omnipresent session-object token SYSTOK-SESSION-ONLY.

```
RDEF CRYPTOZ FIPSEXEMPT.SYSTOK-SESSION-ONLY UACC(NONE)
```

Although the use of generic profiles in the CRYPTOZ class is permitted, you should begin the profile name with “FIPSEXEMPT”. Failure to do this could result in generic characters unintentionally

matching the `SO.token-label` or `USER.token-label` resources for token access, and so could have unintended consequences.

- c. Using the PERMIT command, specify READ access authority for user IDs that are exempt from FIPS 140-2 restrictions, and NONE access authority for user IDs that must comply with FIPS 140-2. The following command indicates that all user IDs are exempt, except for the daemon user ID BOGD.

```
PERMIT FIPSEXEMPT.SYSTOK-SESSION-ONLY CLASS(CRYPTOZ) ID(*) ACC(READ)
PERMIT FIPSEXEMPT.SYSTOK-SESSION-ONLY CLASS(CRYPTOZ) ID(BOGD) ACC(NONE)
```

- d. Refresh the CRYPTOZ class in common storage:

```
SETRPTS RACLIST(CRYPTOZ) REFRESH
```

### Specifying FIPS 140-2 compliance from within a z/OS PKCS #11 application

When running in FIPS compatibility mode or FIPS no enforcement mode, a PKCS #11 application can, when creating a key, specify that generation and subsequent use of the key must adhere to FIPS 140-2 restrictions. An application specifies this by setting the Boolean attribute `CKA_IBM_FIPS140` to TRUE when creating the key. If an application does this, the FIPS 140-2 restrictions (as outlined in [Table 6 on page 29](#)) will be enforced for the key regardless of any specifications made at the token level using `FIPSEXEMPT.token-label` resource profiles.

If the `FIPSMODE(YES, FAIL(fail-option))` installation option is specified, indicating FIPS 140-2 compliance is required by all applications, setting the Boolean attribute `CKA_IBM_FIPS140` to TRUE is merely redundant and does not result in an error.

## Preparing to use PKCS #11 applications

---

Before an installation can use PKCS #11 applications, some preparation is required on the part of the system programmer, security administrator, auditor, and application programmers. This topic describes the preparation required.

### Tasks for the system programmer

If persistent PKCS #11 tokens and objects are needed, the system programmer allocates a token data set (TKDS) for use by PKCS #11 functions, and specifies the data set name of the TKDS in the `TKDSN` option of the ICSF installation options data set.

The system programmer must decide whether or not sysplex-wide consistency of the TKDS is required, and must specify the `SYSPLEXTKDS` option in the ICSF installation options data set to define the processing of TKDS updates in a sysplex environment.

If any application must comply with the FIPS 140-2 standard, the system programmer must configure ICSF to run PKCS #11 services in compliance with FIPS 140-2. To do this, the system administrator uses the `FIPSMODE` option to specify either FIPS standard mode or FIPS compatibility mode as required. For more information on the `FIPSMODE` option, refer to [“Operating in compliance with FIPS 140-2” on page 10](#), and the [z/OS Cryptographic Services ICSF System Programmer's Guide](#).

The system programmer should run the `testpkcs11` utility program to test the PKCS #11 configuration. For information about running the `testpkcs11` program, see [“Running the pre-compiled version of testpkcs11” on page 73](#).

### Tasks for the security administrator

The security administrator creates a RACF profile to protect the data set that contains the token data set. It is important to protect this data set because keys in the token data set are not encrypted.

The security administrator needs to grant the appropriate access authority to users for accessing tokens and objects, by defining profiles in the CRYPTOZ class. For more information, see [“Controlling token access and key policy” on page 2](#).



The security administrator controls access to the PKCS #11 callable services by defining profiles in the CSFSERV class. For information about defining profiles in the CSFSERV class, see *z/OS Cryptographic Services ICSF Administrator's Guide*. For a list of the resource names for token services, see [Table 2 on page 4](#).

If PKCS #11 services must run in compliance with the FIPS 140-2 standard, the security administrator must ensure that the digital signature of the load module that contains the z/OS PKCS #11 services is verified when ICSF starts. This must be done to satisfy FIPS 140-2 requirements. For more information, see [“Requiring signature verification for ICSF module CSFINPV2” on page 13](#) and *z/OS Security Server RACF Security Administrator's Guide*.

To use Secure Key PKCS #11, an active Enterprise PKCS #11 coprocessor is required. For the steps necessary to activate the Enterprise PKCS #11 coprocessors, see 'Cryptographic Hardware Features supported by z/OS ICSF' in *z/OS Cryptographic Services ICSF Administrator's Guide*.

## Tasks for the auditor

Auditors should become familiar with the data in SMF records that is related to PKCS #11 functions. For more information, see [“Auditing PKCS #11 functions” on page 9](#).

## Tasks for application programmers

Application programmers can write applications using the PKCS #11 API provided by ICSF. They should become familiar with the PKCS #11 specification, and with the information in this book. The PKCS #11 standard can be found at [PKCS#11: Cryptographic Token Interface Standard \(www.cryptosoft.com/pkcs11doc\)](http://www.cryptosoft.com/pkcs11doc).

Application programmers can use the sample program, testpkcs11, to learn about building and running PKCS #11 applications, and to troubleshoot problems. For information about the sample program, see [Chapter 3, “Sample PKCS #11 C programs ,” on page 73](#).

## Optional Crypto Express adapters

---

Optional cryptographic adapters (Crypto Express) can be configured as:

- A CCA cryptographic coprocessor.
- An accelerator.
- A PKCS #11 cryptographic coprocessor.

For details on hardware adapters and their configuration options, see 'Cryptographic Hardware Features supported by z/OS ICSF' in *z/OS Cryptographic Services ICSF Administrator's Guide*.

In some cases, an optional adapter is required. When the optional adapter is not required, ICSF uses the optional adapter if available with some restrictions. Otherwise, the operation is done in software. To determine which services use available hardware, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

- A secure PKCS #11 cryptographic coprocessor is required to generate and use secure PKCS #11 keys. It can also be used, if present, to offload MIPS for some clear key operations such as DSA and DH domain parameter generation.
- The CCA cryptographic coprocessor or accelerator adapters are optional. If present, they can be used to offload MIPS for the following clear key operations:
  - For an Accelerator or CCA coprocessor:
    - RSA Sign/Verify (but not RSA PSS), Encrypt/Decrypt, Key Wrap/Unwrap.
    - DH Key Agreement.
  - For a CCA coprocessor only:
    - RSA and EC Key-pair Generate.
    - EC-DH Key Agreement.

- ECDSA Signature Verify.
- Operations that must meet FIPS 140-2 standards are not directed to the CCA cryptographic coprocessors.
- Operations that must meet FIPS 140-2 standards are only directed to an accelerator when at least one accelerator was online at ICSF startup. If the first accelerator comes online after ICSF startup, you must restart ICSF if you want that accelerator to be used for any PKCS #11 RSA functions that require adherence to the FIPS 140-2 standard. For non-FIPS restricted functions, the accelerator is used regardless of when the first accelerator comes online.

---

## Chapter 2. The C API

ICSF provides a PKCS #11 C language application program interface (API). This topic highlights the differences between the z/OS API and the PKCS #11 V2.20 specification. To use this API, you must be familiar with both the PKCS #11 specification and the information in this topic.

All manifest constants specified in this chapter can be found in the `csnpdefs.h` include file and (with the exceptions noted) in the PKCS #11 specification.

---

### Using the C API

To create or use a z/OS PKCS #11 token, an application needs to do the following:

1. Implicitly or explicitly load the PKCS #11 API DLL (CSNPCAPI for applications running in 31-bit addressing mode not using XPLINK, CSNPCA3X for applications running in 31-bit addressing mode using XPLINK, CSNPCA64 for 64-bit addressing mode).
2. Locate the functions within that DLL, using the `C_GetFunctionList` function.
3. Call `C_Initialize`, which enables the application to call other functions in the API.
4. Determine the slots present, using the `C_GetSlotList` function. This function returns a slot number for each existing token to which the application has access.
5. To use an existing token, the application iterates through the slots using `C_GetTokenInfo` to find the token wanted.

To create a new token, the application uses the `C_WaitForSlotEvent` function to add a new slot containing an uninitialized token. The application then uses the `C_InitToken` function to initialize the new token and save it in the TKDS.

### Deleting z/OS PKCS #11 tokens

Because PKCS #11 tokens are typically physical hardware devices, the PKCS #11 specification provides no mechanism to delete tokens. However, for z/OS PKCS #11 tokens, which are virtual, there must be a capability to delete tokens. An application does this by calling the `C_InitToken` function with a special label value `$$DELETE-TOKEN$$` (assuming code page IBM1047), left-justified and padded on the right to 32 characters.

**Tip:** Use the constant `DEL_TOK` defined in `csnpdefs.h`.

You cannot delete the omnipresent token `SYSTOK-SESSION-ONLY` (created by ICSF to provide PKCS #11 support even if no other token is available to an application). If an application attempts to delete the omnipresent token, the `C_InitToken` function will fail with a return value of `CKR_TOKEN_WRITE_PROTECTED`.

### Environment

**Note:** PKCS #11 programs must run in a POSIX-enabled environment such as that created by the z/OS Unix shell. For additional options, see 'Running POSIX-enabled Programs' in *z/OS Language Environment Programming Guide*.

#### Restriction

The calling program must be running as a Language-Environment-enabled (LE-enabled) application in TCB mode only. SRB mode is not supported.

#### Guideline

To use PKCS #11 in SRB mode, you must call the PKCS #11 ICSF callable services directly.

## Cross memory considerations

On z/OS, an application can be running in either single address space mode, or in cross memory mode. The PKCS #11 standard has no concept of cross memory mode, so there is no predefined expected behavior for a PKCS #11 application running in cross memory mode.

When running in cross memory mode, the unit of work is running with PRIMARY set to an address space that differs from HOME. This PRIMARY space may be another address space that is logically part of the overall application (for example, if the application was designed to be cross memory aware) or it may be a daemon or subsystem address space dedicated to some system service that the calling application has invoked using a Program Call (PC). Either way, you should be aware of the following z/OS PKCS #11 application behaviors and associated guidelines.

- The C API invokes Language Environment (LE) services that are not supported in cross memory mode.

**Guideline:** To use PKCS #11 in cross memory mode, you must call the PKCS #11 ICSF callable services directly.

- Tokens, token objects, and session objects belonging to installation-defined PKCS #11 tokens (but not to the omnipresent token) are protected by RACF access control. Additionally, the ICSF callable services themselves may also be protected by RACF access control. In both cases, the user ID that is used for the access check is always associated with the unit of work. This is either the user ID assigned to the HOME ASCB or the user ID assigned to the Task Control Block (TCB) or System Request Block (SRB).

**Guideline:** If the PRIMARY address space function invoked by a PC uses PKCS #11 services, the user ID associated with the caller's unit of work must be appropriately permitted to the CRYPTOZ or CSFSERV resource being checked. If this is a system service, then all such callers must have access.

- FIPS140 compatibility mode behavior is also controlled by resources in the CRYPTOZ Class.

**Guideline:** If the system is configured for FIPS140 compatibility mode and the PRIMARY address space function invoked by a PC is expected to adhere to FIPS 140-2 restrictions, the user ID associated with the caller's unit of work should not be permitted to the CRYPTOZ FIPSEXEMPT resource being checked. If this is a system service, then all such callers should not to be given access.

- By definition, session objects (including those belonging to the omnipresent token) are scoped to a single address space. For session objects belonging to installation defined PKCS #11 tokens, the scoping is to the HOME address space at the time of object creation, even if PRIMARY does not equal HOME. These objects are accessible to all units of work belonging to the HOME address space only, even if the PRIMARY address space function invoked by a PC is intended to be the logical owner of the PKCS #11 object.

In contrast, session objects belonging to the omnipresent token are scoped to the PRIMARY address space at the time of object creation and are addressable by all units of work running with that address space set as PRIMARY. For session objects created by system services invoked by a PC, such session objects would not be addressable by the caller once returning from the service call.

**Guideline:** System services invoked by a PC should use the omnipresent token instead of an installation defined PKCS #11 token when creating session objects that are to be owned by the system service.

- For certain multipart PKCS #11 cryptographic operations, ICSF will save session-state information across calls. This state information is scoped to the PRIMARY address space, similar to the scoping for omnipresent token objects. Such state objects are only addressable to units of work running with that address space set as PRIMARY.

**Guideline:** If you begin a multipart PKCS #11 cryptographic operation, you must remain running in the same PRIMARY address space in order to continue the operation.

## Key types and mechanisms supported

---

ICSF supports the following PKCS #11 key types (CKA\_KEY\_TYPE). All of these key types are supported in software. Whether they are also supported in hardware depends on the limitations of your cryptographic hardware configuration.

- CKK\_AES - Key lengths 128, 192, and 256 bits.
- CKK\_BLOWFISH - Key lengths 8 up to 448 bits (in increments of 8 bits).
- CKK\_DES.
- CKK\_DES2.
- CKK\_DES3.
- CKK\_DH - Key lengths 512 up to 2048 bits (in increments of 64 bits).
- CKK\_DSA - Key lengths 512 up to 2048 bit prime lengths (in increments of 64 bits).
- CKK\_EC (CKK\_ECDSA) - Key lengths 160 up to 521 bits.
- CKK\_GENERIC\_SECRET - Key lengths 8 up to 2048 bits, unless further restricted by the generation mechanism:
  - CKM\_DH\_PKCS\_DERIVE - Key lengths 512 up to 2048 bits.
  - CKM\_SSL3\_MASTER\_KEY\_DERIVE - 384-bit key lengths.
  - CKM\_SSL3\_MASTER\_KEY\_DERIVE\_DH - 384-bit key lengths.
  - CKM\_SSL3\_PRE\_MASTER\_KEY\_GEN - 384-bit key lengths.
  - CKM\_TLS\_MASTER\_KEY\_DERIVE - 384-bit key lengths.
  - CKM\_TLS\_MASTER\_KEY\_DERIVE\_DH - 384-bit key lengths.
  - CKM\_TLS\_PRE\_MASTER\_KEY\_GEN - 384-bit key lengths.
- CKK\_IBM\_CHACHA20.
- CKK\_RC4 - Key lengths 8 up to 2048 bits.
- CKK\_RSA - Key lengths 512 up to 4096 bits.
- CKK\_IBM\_SM2 - 256-bit key lengths.
- CKK\_IBM\_SM4 - A vendor-defined key type for SM4 cryptography.
- CKK\_IBM\_DILITHIUM – A quantum-safe key type.

The following table shows the mechanisms that are supported by different hardware configurations. All the mechanisms are supported in software, and some might be available in hardware. If the mechanism is available in hardware, ICSF uses the hardware mechanism. If the mechanism is not available in hardware, ICSF uses the software mechanism. The following table also shows the flags that are returned by the `C_GetMechanismInfo` function in the `CK_MECHANISM_INFO` structure. Whether the `CKF_HW` flag is returned in the `CK_MECHANISM_INFO` structure indicates whether the mechanism is supported in the hardware.

*Table 4. Mechanism information as returned by `C_GetMechanismInfo` (`CK_MECHANISM_INFO`)*

Type ( <code>CK_MECHANISM_TYPE</code> )	Size factor	Flags
<code>CKM_AES_CBC</code> <sup>3</sup>	Bytes	[ <code>CKF_HW</code> ] <code>CKF_ENCRYPT</code> <code>CKF_DECRYPT</code>
<code>CKM_AES_CBC_PAD</code> <sup>3</sup>	Bytes	[ <code>CKF_HW</code> ] <code>CKF_ENCRYPT</code> <code>CKF_DECRYPT</code> <code>CKF_WRAP</code> <code>CKF_UNWRAP</code>
<code>CKM_AES_CTS</code> <sup>3</sup>	Bytes	[ <code>CKF_HW</code> ] <code>CKF_ENCRYPT</code> <code>CKF_DECRYPT</code>
<code>CKM_AES_ECB</code> <sup>3</sup>	Bytes	[ <code>CKF_HW</code> ] <code>CKF_ENCRYPT</code> <code>CKF_DECRYPT</code>
<code>CKM_AES_GCM</code> <sup>3,7</sup>	Bytes	<code>CKF_ENCRYPT</code> <code>CKF_DECRYPT</code>
<code>CKM_AES_KEY_GEN</code>	Bytes	[ <code>CKF_HW</code> ] <code>CKF_GENERATE</code>
<code>CKM_BLOWFISH_CBC</code> <sup>4,7</sup>	Bytes	<code>CKF_ENCRYPT</code> <code>CKF_DECRYPT</code>
<code>CKM_BLOWFISH_KEY_GEN</code> <sup>7</sup>	Bytes	[ <code>CKF_HW</code> ] <code>CKF_GENERATE</code>
<code>CKM_DES_CBC</code> <sup>7</sup>	Not applicable	[ <code>CKF_HW</code> ] <code>CKF_ENCRYPT</code> <code>CKF_DECRYPT</code>

Table 4. Mechanism information as returned by C\_GetMechanismInfo (CK\_MECHANISM\_INFO) (continued)

Type (CK_MECHANISM_TYPE)	Size factor	Flags
CKM_DES_CBC_PAD <sup>7</sup>	Not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP
CKM_DES_ECB <sup>7</sup>	Not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_DES_KEY_GEN <sup>7</sup>	Not applicable	[CKF_HW] CKF_GENERATE
CKM_DES2_KEY_GEN <sup>7</sup>	Not applicable	[CKF_HW] CKF_GENERATE
CKM_DES3_CBC <sup>3</sup>	Not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_DES3_CBC_PAD <sup>3</sup>	Not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP
CKM_DES3_ECB <sup>3</sup>	Not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_DES3_KEY_GEN	Not applicable	[CKF_HW] CKF_GENERATE
CKM_DH_PKCS_DERIVE	Bits	[CKF_HW] CKF_DERIVE
CKM_DH_PKCS_KEY_PAIR_GEN	Bits	[CKF_HW] CKF_GENERATE_KEY_PAIR
CKM_DH_PKCS_PARAMETER_GEN	Bits	[CKF_HW] CKF_GENERATE
CKM_DSA	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_DSA_KEY_PAIR_GEN	Bits	[CKF_HW] CKF_GENERATE_KEY_PAIR
CKM_DSA_PARAMETER_GEN	Bits	[CKF_HW] CKF_GENERATE
CKM_DSA_SHA1	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_EC_KEY_PAIR_GEN	Bits	[CKF_HW] CKF_GENERATE_KEY_PAIR CKF_EC_F_P <sup>1</sup> CKF_EC_NAMEDCURVE <sup>2</sup> CKF_EC_UNCOMPRESS
CKM_ECDH1_DERIVE	Bits	[CKF_HW] CKF_DERIVE CKF_EC_F_P <sup>1</sup> CKF_EC_NAMEDCURVE <sup>2</sup> CKF_EC_UNCOMPRESS
CKM_ECDSA	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY CKF_EC_F_P <sup>1</sup> CKF_EC_NAMEDCURVE <sup>2</sup> CKF_EC_UNCOMPRESS
CKM_ECDSA_SHA1	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY CKF_EC_F_P <sup>1</sup> CKF_EC_NAMEDCURVE <sup>2</sup> CKF_EC_UNCOMPRESS
CKM_GENERIC_SECRET_KEY_GEN <sup>3</sup>	Bits	[CKF_HW] CKF_GENERATE
CKM_IBM_ATTRIBUTEBOUND_WRAP <sup>8</sup> (vendor specific mechanism - 0x80020004). IBM proprietary wrap/ unwrap mechanism that includes the Boolean usage attributes along with the key data. Only supported for secure keys that have the CKA_IBM_ATTRBOUND attribute set TRUE	Not applicable	[CKF_HW] CKF_WRAP CKF_UNWRAP
CKM_IBM_ISO2_SM4_MAC (0x80058008)	Not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY

Table 4. Mechanism information as returned by C\_GetMechanismInfo (CK\_MECHANISM\_INFO) (continued)

Type (CK_MECHANISM_TYPE)	Size factor	Flags
<b>CKM_IBM_ISO2_SM4_MAC_GENERAL (0x80050008)</b>	Not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
<b>CKM_IBM_SM2 (0x8005000B)</b>	Not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
<b>CKM_IBM_SM2_ENCRYPT (0x8005000D)</b>	Not applicable	[CKF_HW] CKF_WRAP CKF_UNWRAP
<b>CKM_IBM_SM2_KEY_PAIR_GEN (0x8005000A)</b>	Not applicable	[CKF_HW] CKF_GENERATE_KEY_PAIR
<b>CKM_IBM_SM2_SM3 (0x8005000C)</b>	Not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
<b>CKM_IBM_SM3 (0x8005000E)</b>	Not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
<b>CKM_IBM_SM4_CBC (0x80050002)</b>	Not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
<b>CKM_IBM_SM4_ECB (0x80050004)</b>	Not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP
<b>CKM_IBM_SM4_ECB_ENCRYPT_DATA (0x80050009)</b>	Not applicable	[CKF_HW] CKF_DERIVE
<b>CKM_IBM_SM4_KEY_GEN (0x80050001)</b>	Not applicable	[CKF_HW] CKF_GENERATE
<b>CKM_IBM_SM4_MAC (0x80058007)</b>	Not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
<b>CKM_IBM_SM4_MAC_GENERAL (0x80050007)</b>	Not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
<b>CKM_MD2</b>	Not applicable	CKF_DIGEST
<b>CKM_MD2_RSA_PKCS<sup>5,6</sup></b>	Bits	CKF_SIGN CKF_VERIFY
<b>CKM_MD5</b>	Not applicable	CKF_DIGEST
<b>CKM_MD5_HMAC</b>	Not applicable	CKF_SIGN CKF_VERIFY
<b>CKM_MD5_RSA_PKCS<sup>5,6</sup></b>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
<b>CKM_PBE_SHA1_DES3_EDE_CBC</b>	Not applicable	[CKF_HW] CKF_GENERATE
<b>CKM_RC4<sup>4,7</sup></b>	Bits	CKF_ENCRYPT CKF_DECRYPT
<b>CKM_RC4_KEY_GEN<sup>7</sup></b>	Bits	[CKF_HW] CKF_GENERATE
<b>CKM_RIPEMD160</b>	Not applicable	CKF_DIGEST
<b>CKM_RSA_PKCS<sup>5,6</sup></b>	Bits	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP CKF_SIGN CKF_VERIFY CKF_SIGN_RECOVER CKF_VERIFY_RECOVER
<b>CKM_RSA_PKCS_KEY_PAIR_GEN</b>	Bits	[CKF_HW] CKF_GENERATE_KEY_PAIR
<b>CKM_RSA_PKCS_PSS<sup>9</sup></b>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
<b>CKM_RSA_X_509<sup>5,6,7</sup></b>	Bits	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_SIGN CKF_VERIFY CKF_SIGN_RECOVER CKF_VERIFY_RECOVER
<b>CKM_SHA_1</b>	Not applicable	[CKF_HW] CKF_DIGEST

Table 4. Mechanism information as returned by C\_GetMechanismInfo (CK\_MECHANISM\_INFO) (continued)

Type (CK_MECHANISM_TYPE)	Size factor	Flags
CKM_SHA_1_HMAC	Not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA1_RSA_PKCS <sup>5, 6</sup>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA1_RSA_PKCS_PSS <sup>9</sup>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA224	Not applicable	[CKF_HW] CKF_DIGEST
CKM_SHA224_HMAC	Not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA224_RSA_PKCS <sup>5, 6</sup>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA224_RSA_PKCS_PSS <sup>9</sup>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA256	Not applicable	[CKF_HW] CKF_DIGEST
CKM_SHA256_HMAC	Not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA256_RSA_PKCS <sup>5, 6</sup>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA256_RSA_PKCS_PSS <sup>9</sup>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA384	Not applicable	[CKF_HW] CKF_DIGEST
CKM_SHA384_HMAC	Not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA384_RSA_PKCS <sup>5, 6</sup>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA384_RSA_PKCS_PSS <sup>9</sup>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA512	Not applicable	[CKF_HW] CKF_DIGEST
CKM_SHA512_HMAC	Not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA512_RSA_PKCS <sup>5, 6</sup>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA512_RSA_PKCS_PSS <sup>9</sup>	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SSL3_KEY_AND_MAC_DERIVE <sup>7</sup>	Not applicable	CKF_DERIVE
CKM_SSL3_MASTER_KEY_DERIVE <sup>7</sup>	Bytes	CKF_DERIVE
CKM_SSL3_MASTER_KEY_DERIVE_DH <sup>7</sup>	Bytes	CKF_DERIVE
CKM_SSL3_MD5_MAC <sup>7</sup>	Bits	CKF_SIGN CKF_VERIFY
CKM_SSL3_PRE_MASTER_KEY_GEN <sup>7</sup>	Bytes	[CKF_HW] CKF_GENERATE
CKM_SSL3_SHA1_MAC <sup>7</sup>	Bits	CKF_SIGN CKF_VERIFY
CKM_TLS_KEY_AND_MAC_DERIVE <sup>7</sup>	Not applicable	CKF_DERIVE
CKM_TLS_MASTER_KEY_DERIVE <sup>7</sup>	Bytes	CKF_DERIVE
CKM_TLS_MASTER_KEY_DERIVE_DH <sup>7</sup>	Bytes	CKF_DERIVE
CKM_TLS_PRE_MASTER_KEY_GEN <sup>7</sup>	Bytes	[CKF_HW] CKF_GENERATE
CKM_TLS_PRF <sup>7</sup>	Not applicable	CKF_DERIVE
CKM_XOR_BASE_AND_DATA	Not applicable	[CKF_HW] CKF_DERIVE

Footnotes for Table 4 on page 21

<sup>1</sup> The PKCS #11 standard designates two ways of implementing Elliptic Curve Cryptography, which is nicknamed  $F_p$  and  $F_2^m$ . z/OS PKCS #11 supports the  $F_p$  variety only.



<sup>2</sup> ANSI X9.62 has the following ASN.1 definition for Elliptic Curve domain parameters:

```
Parameters ::= CHOICE {
    ecParameters    ECParameters,
    namedCurve      OBJECT IDENTIFIER,
    implicitlyCA    NULL }
```

z/OS PKCS #11 supports the specification of CKA\_EC\_PARAMS attribute by using the namedCurve CHOICE. The following NIST-recommended named curves are supported:

- secp192r1 – { 1 2 840 10045 3 1 1 }
- secp224r1 – { 1 3 132 0 33 }
- secp256r1 – { 1 2 840 10045 3 1 7 }
- secp384r1 – { 1 3 132 0 34 }
- secp521r1 – { 1 3 132 0 35 }

The following Brainpool-defined named curves are supported:

- brainpoolP160r1 – { 1 3 36 3 3 2 8 1 1 1 }
- brainpoolP192r1 – { 1 3 36 3 3 2 8 1 1 3 }
- brainpoolP224r1 – { 1 3 36 3 3 2 8 1 1 5 }
- brainpoolP256r1 – { 1 3 36 3 3 2 8 1 1 7 }
- brainpoolP320r1 – { 1 3 36 3 3 2 8 1 1 9 }
- brainpoolP384r1 – { 1 3 36 3 3 2 8 1 1 11 }
- brainpoolP512r1 – { 1 3 36 3 3 2 8 1 1 13 }

The following Edwards named curves are supported:

- Ed448 – { 1 3 101 113 }
- Ed25519 – { 1 3 101 112 }

The following Montgomery named curves are supported:

- X448 – { 1 3 101 111 }
- X25519 – { 1 3 101 110 }

In addition, z/OS PKCS #11 has limited support for the ecParameters CHOICE. When specified, the DER encoding must contain the optional cofactor field and must not contain the optional Curve.seed field. Also, calls to C\_GetAttributeValue to retrieve the CKA\_EC\_PARAMS attribute always returns the value in the namedCurve form regardless of how the attribute was specified when the object was created. Due to these limitations, the CKF\_EC\_ECPARAMETERS flag is not turned on for the applicable mechanisms.

Also, z/OS PKCS#11 supports the specification of the CKA\_IBM\_DILITHIUM\_MODE attribute for the CKK\_IBM\_DILITHIUM key type. The following mode is supported:

- CRYSTALS-Dilithium (6,5) - { 1 3 6 1 4 1 2 267 1 6 5 }

<sup>3</sup> Mechanism not present on a system that is export controlled.

<sup>4</sup> Mechanism limited to 56-bit on a system that is export controlled.

<sup>5</sup> In general, z/OS PKCS #11 expects RSA private keys to be in Chinese Remainder Theorem (CRT) format. However, for Decrypt, Sign, or UnwrapKey (z890, z990 or higher only) where one of the following is true, the shorter Modulus Exponent (ME) is permitted:

- There is an accelerator present and the key is less than or equal to 2048 bits in length.
- There is a coprocessor present and the key is less than or equal to 1024 bits in length and FIPS restrictions do not apply.

<sup>6</sup> RSA public or private keys that have a public exponent greater than 8 bytes in length can only be used when a coprocessor or accelerator is present.

<sup>7</sup> Mechanism supported for clear keys only.

<sup>8</sup> Mechanism supported for secure keys only.

<sup>9</sup> PARAM field restrictions for PSS algorithms:

```
typedef struct CK_RSA_PKCS_PSS_PARAMS {
    CK_MECHANISM_TYPE hashAlg;
    CK_RSA_PKCS_MGF_TYPE mgf;
    CK_ULONG slen;
} CK_RSA_PKCS_PSS_PARAMS;
```

- For mechanisms other than CKM\_RSA\_PKCS\_PSS, the hashAlg must match the mechanism specified. For mechanism CKM\_RSA\_PKCS\_PSS, the hashAlg must be a supported SHA algorithm.
- mgf must match the algorithm specified by hashAlg
- slen must be either 0 or the size of the output of the hashAlg specified.

The following table lists the mechanisms supported by specific cryptographic hardware. When a particular mechanism is not available in hardware, ICSF uses the software implementation of the mechanism.

<i>Table 5. Mechanisms supported by specific cryptographic hardware</i>		
<b>Machine type and cryptographic hardware</b>	<b>Mechanisms supported</b>	<b>Notes</b>
<b>z9 - CEX2C</b>	CKM_DES_KEY_GEN CKM_DES2_KEY_GEN CKM_DES3_KEY_GEN CKM_RSA_PKCS CKM_RSA_X_509 CKM_MD5_RSA_PKCS CKM_SHA1_RSA_PKCS CKM_DES_CBC CKM_DES_CBC_PAD CKM_DES3_CBC CKM_DES3_CBC_PAD CKM_SHA_1 CKM_BLOWFISH_KEY_GEN CKM_RC4_KEY_GEN CKM_AES_KEY_GEN CKM_SSL3_PRE_MASTER_KEY_GEN CKM_TLS_PRE_MASTER_KEY_GEN CKM_GENERIC_SECRET_KEY_GEN CKM_RSA_PKCS_KEY_PAIR_GEN CKM_EC_KEY_PAIR_GEN CKM_DES_ECB CKM_DES3_ECB  PCIXCC set plus: CKM_RSA_PKCS_KEY_PAIR_GEN CKM_DES_ECB CKM_DES3_ECB CKM_SHA224_RSA_PKCS CKM_SHA256_RSA_PKCS CKM_SHA224 CKM_SHA256 CKM_AES_CBC CKM_AES_CBC_PAD CKM_AES_CTS CKM_AES_ECB	This is the base set.  AES key operations limited to 128 bits in length (maximum).  RSA private key operations limited to 1024 bits in length (maximum) and no key pair generation capability.  RSA private key operations limited to 40496 bits in length (maximum).

Table 5. Mechanisms supported by specific cryptographic hardware (continued)

Machine type and cryptographic hardware	Mechanisms supported	Notes
<b>z10 - CEX2C or CEX3C</b>	z9 CEX2C set plus: CKM_SHA384_RSA_PKCS CKM_SHA512_RSA_PKCS CKM_SHA384 CKM_SHA512	AES key operations limited to 256 bits in length (maximum).
<b>IBM zEnterprise 196 - CEX3C</b>	z9 CEX2C set plus: CKM_SHA384_RSA_PKCS CKM_SHA512_RSA_PKCS CKM_SHA384 CKM_SHA512	AES key operations limited to 256 bits in length (maximum).  RSA private key operations limited to 4096 bits in length (maximum).
<b>IBM zEnterprise EC12 or IBM zEnterprise BC12 with an Enterprise PKCS #11 coprocessor</b>	z10 set plus: CKM_IBM_ATTRIBUTEBOUND_WRAP CKM_PBE_SHA1_DES3_EDE_CBC CKM_DSA_PARAMETER_GEN CKM_DH_PKCS_KEY_PAIR_GEN CKM_DH_PKCS_DERIVE CKM_ECDH1_DERIVE CKM_RSA_PKCS_PSS CKM_SHA1_RSA_PKCS_PSS CKM_SHA224_RSA_PKCS_PSS CKM_SHA256_RSA_PKCS_PSS CKM_SHA384_RSA_PKCS_PSS CKM_SHA512_RSA_PKCS_PSS	Requires the Sept. 2013 or later licensed internal code (LIC).  The CKM_ECDH1_DERIVE mechanism may only derive a clear key.
<b>IBM zEnterprise EC12 or later</b>	CKM_IBM_SM4_CBC CKM_IBM_SM4_ECB CKM_IBM_SM4_ECB_ENCRYPT_DATA CKM_IBM_ISO2_SM4_MAC CKM_IBM_ISO2_SM4_MAC_GENERAL CKM_IBM_SM4_KEY_GEN CKM_IBM_SM4_MAC CKM_IBM_SM4_MAC_GENERAL CKM_XOR_BASE_AND_DATA	A regional cryptographic server must be active.
<b>IBM z13 or z13s</b>	zEC12 and zBC12 set	
<b>IBM z14 or z14 ZR1</b>	z13 and z13s set	

Table 5. Mechanisms supported by specific cryptographic hardware (continued)

Machine type and cryptographic hardware	Mechanisms supported	Notes
<b>IBM z15</b>	z14 and z14 ZR1 set	With the November 2019 or later licensed internal code (LIC): <ul style="list-style-type: none"> <li>• CKM_ECDH1_DERIVE mechanism may derive a secure key.</li> <li>• CKK_IBM_DILITHIUM key type may be used to create or generate Dilithium secure keys.</li> <li>• CKA_IBM_PROTKEY_EXTRACTABLE may be specified for generation, creation, and derivation of supported key types (AES, DES3, secp256r1, secp384r1, secp521r1, Ed448, and Ed25519).</li> </ul>
<b>z890, z990, or later with Regional Cryptographic Server Gen 1</b>	CKM_IBM_SM4_CBC CKM_IBM_SM4_ECB CKM_IBM_SM4_ECB_ENCRYPT_DATA CKM_IBM_SM4_KEY_GEN CKM_IBM_SM4_MAC CKM_IBM_SM4_MAC_GENERAL CKM_IBM_ISO2_SM4_MAC CKM_IBM_ISO2_SM4_MAC_GENERAL CKM_XOR_BASE_AND_DATA	
<b>z890, z990, or later with Regional Cryptographic Server Gen 2</b>	CKM_IBM_SM2 CKM_IBM_SM2_ENCRYPT CKM_IBM_SM2_KEY_PAIR_GEN CKM_IBM_SM2_SM3 CKM_IBM_SM3	

Table 5. Mechanisms supported by specific cryptographic hardware (continued)

Machine type and cryptographic hardware	Mechanisms supported	Notes
<b>z890, z990, or later with Regional Cryptographic Server Gen 3</b>	CKM_AES_CBC CKM_AES_CBC_PAD CKM_AES_ECB CKM_AES_KEY_GEN CKM_DES3_CBC CKM_DES3_CBC_PAD CKM_DES3_ECB CKM_DES3_KEY_GEN CKM_ECDSA CKM_EC_KEY_PAIR_GEN CKM_RSA_PKCS CKM_RSA_PKCS_KEY_PAIR_GEN CKM_RSA_PKCS_PSS CKM_RSA_X_509 CKM_SHA1_RSA_PKCS CKM_SHA1_RSA_PKCS_PSS CKM_SHA224_RSA_PKCS CKM_SHA224_RSA_PKCS_PSS CKM_SHA256_RSA_PKCS CKM_SHA256_RSA_PKCS_PSS CKM_SHA384_RSA_PKCS CKM_SHA384_RSA_PKCS_PSS CKM_SHA512_RSA_PKCS CKM_SHA512_RSA_PKCS_PSS	

The following table lists the algorithms and uses (by mechanism) that are not allowed when operating in compliance with FIPS 140-2. For more information about how the z/OS PKCS #11 services can be configured to operate in compliance with the FIPS 140-2 standard, see [“Operating in compliance with FIPS 140-2”](#) on page 10.

Table 6. Restricted algorithms and uses when running in compliance with FIPS 140-2

Algorithm	Mechanisms	Usage disallowed
<b>AES GCM</b>	CKM_AES_GCM	GCM encryption or GMAC generation with externally generated initialization vectors. Initialization vector lengths other than 12 bytes. Tag byte sizes 4 and 8.
<b>Blowfish</b>	CKM_BLOWFISH_KEY_GEN, CKM_BLOWFISH_CBC	All
<b>ChaCha20-Poly1305</b>	CKM_IBM_CHACHA20_POLY1305	All
<b>Diffie Hellman</b>	CKM_DH_PKCS_DERIVE	Prime size less than 1024 bits.
	CKM_DH_PKCS_PARAMETER_GEN	Prime sizes other than 1024 or 2048 bits.
<b>Dilithium</b>	N/A	All
<b>DSA</b>	CKM_DSA_SHA1, CKM_DSA	Prime sizes less than 1024 bits.

Table 6. Restricted algorithms and uses when running in compliance with FIPS 140-2 (continued)

Algorithm	Mechanisms	Usage disallowed
<b>DSA</b>	CKM_DSA_PARAMETER_GEN, CKM_DSA_KEY_PAIR_GEN or Sign	Combinations other than the following: <ul style="list-style-type: none"> <li>• Prime size = 1024 bits, subprime size = 160 bits.</li> <li>• Prime size = 2048 bits, subprime size = 224 bits, or 256 bits.</li> </ul>
<b>HMAC</b>	CKM_SHA_1, CKM_SHA224, CKM_SHA256, CKM_SHA384, CKM_SHA512	Base key sizes less than one half the output size.
<b>MD2</b>	CKM_MD2, CKM_MD2_RSA_PKCS	All
<b>MD5</b>	CKM_MD5, CKM_MD5_RSA_PKCS, CKM_MD5_HMAC	All
<b>RC4</b>	CKM_RC4	All
<b>RIPEMD</b>	CKM_RIPEMD160	All
<b>RSA</b>	CKM_RSA_X_509	All
<b>RSA</b>	CKM_RSA_PKCS	Key sizes less than 1024 bits.
<b>RSA</b>	CKM_RSA_PKCS_KEY_PAIR_GEN or Sign without an active accelerator	Key sizes that are less than 1024 bits or not a multiple of 256 bits or public key exponents less than 0x010001.
<b>Single DES</b>	CKM_DES_ECB, CKM_DES_CBC, CKM_DES_CBC_PAD	All
<b>SSL3</b>	CKM_SSL3_MD5_MAC, CKM_SSL3_SHA1_MAC, CKM_SSL3_MASTER_KEY_DERIVE, CKM_SSL3_MASTER_KEY_DERIVE_DH, CKM_SSL3_KEY_AND_MAC_DERIVE	All
<b>TLS</b>	CKM_TLS_MASTER_KEY_DERIVE, CKM_TLS_MASTER_KEY_DERIVE_DH, CKM_TLS_KEY_AND_MAC_DERIVE	Base key sizes less than 10 bytes.
<b>Triple DES</b>	CKM_DES3_ECB, CKM_DES3_CBC, CKM_DES3_CBC_PAD	Two key Triple DES.  Three key Triple DES encryption or key wrap where the individual DES key parts are not unique.

## Additional manifest constants for Dilithium quantum-safe algorithm support

```
#define CKK_IBM_DILITHIUM           0x 80010023
#define CKA_IBM_DILITHIUM_MODE     0x 80000010
```

## Objects and attributes supported

ICSF supports the following PKCS #11 object types (CK\_OBJECT\_CLASS):

- CKO\_DATA.
- CKO\_CERTIFICATE - CKC\_X\_509.
- CKO\_DOMAIN\_PARAMETERS - CKK\_DSA and CKK\_DH.
- CKO\_PUBLIC\_KEY - CKK\_RSA, CKK\_EC (CKK\_ECDSA), CKK\_DSA, CKK\_DH, and CKK\_IBM\_SM2.
- CKO\_PRIVATE\_KEY - CKK\_RSA, CKK\_EC (CKK\_ECDSA), CKK\_DSA, CKK\_DH, and CKK\_IBM\_SM2.
- CKO\_SECRET\_KEY - CKK\_DES, CKK\_DES2, CKK\_DES3, CKK\_AES, CKK\_BLOWFISH, CKK\_RC4, CKK\_GENERIC\_SECRET, CKK\_IBM\_CHACHA20, and CKK\_IBM\_SM4.

The footnotes described in Table 7 on page 31 are taken from the PKCS #11 specification and apply to the attribute tables that follow.

Footnote number	Footnote meaning
1	Must be specified when object is created with <b>C_CreateObject</b> .
2	Must <i>not</i> be specified when object is created with <b>C_CreateObject</b> .
3	Must be specified when object is generated with <b>C_GenerateKey</b> or <b>C_GenerateKeyPair</b> .
4	Must <i>not</i> be specified when object is generated with <b>C_GenerateKey</b> or <b>C_GenerateKeyPair</b> .
5	Must be specified when object is unwrapped with <b>C_UnwrapKey</b> .
6	Must <i>not</i> be specified when object is unwrapped with <b>C_UnwrapKey</b> .
7	Cannot be revealed if object has its <b>CKA_SENSITIVE</b> attribute set to TRUE or its <b>CKA_EXTRACTABLE</b> attribute set to FALSE.
8	May be modified after object is created with a <b>C_SetAttributeValue</b> call, or in the process of copying object with a <b>C_CopyObject</b> call. However, it is possible that a particular token may not permit modification of the attribute, or may not permit modification of the attribute during the course of a <b>C_CopyObject</b> call.
9	Default value is token-specific, and may depend on the values of other attributes.
10	Can only be set to TRUE by the SO user.
11	May be changed during a <b>C_CopyObject</b> call but not on a <b>C_SetAttributeValue</b> call
12	Attribute cannot be changed once set to CK_TRUE. It becomes a read only attribute.
13	May be changed to <b>CK_TRUE</b> during the course of a copy operation but only if the source object is not already a secure key. When the source object is a secure key, the attribute is read only.
14	If <b>CKA_PRIVATE=TRUE</b> , can only be set TRUE by a user who is a Strong SO or a Weak USER who is also an SO.
15	Only modifiable with an Enterprise PKCS #11 coprocessor configured with August 2013 or later licensed internal code (LIC).

Attribute	Data type	Notes
<b>CKA_CLASS</b> <sup>1</sup>	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.

Table 8. Data object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31. (continued)

Attribute	Data type	Notes
<b>CKA_TOKEN<sup>11</sup></b>	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.
<b>CKA_PRIVATE<sup>11</sup></b>	CK_BBOOL	Default value on create is TRUE. An application can specify the value when the object is created (or generated) only.
<b>CKA_MODIFIABLE<sup>11</sup></b>	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
<b>CKA_LABEL</b>	Printable EBCDIC string	Application-specific nickname. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
<b>CKA_ID</b>	Byte array	Key or other identifier. Default is empty. An application can set or change the value at any time.
<b>CKA_VALUE</b>	Byte array	Any value. Default is empty. An application can set or change the value at any time.
<b>CKA_APPLICATION</b>	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
<b>CKA_OBJECT_ID</b>	Byte array	DER-encoded OID. Default is empty. An application can set or change the value at any time.
<b>CKA_IBM_ICSF_HANDLE (vendor specific attribute - 0x80010009)</b>	Printable EBCDIC string	44-character ICSF record locator value. This attribute may be retrieved via C_GetAttributeValue. It is not valid in the attribute list of any other function or callable service.

Table 9. X.509 certificate object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_CLASS<sup>1</sup></b>	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
<b>CKA_TOKEN<sup>11</sup></b>	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.
<b>CKA_PRIVATE<sup>11</sup></b>	CK_BBOOL	Default value on create is FALSE. An application can specify the value when the object is created (or generated) only.



Table 9. X.509 certificate object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31. (continued)

Attribute	Data type	Notes
<b>CKA_MODIFIABLE<sup>11</sup></b>	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
<b>CKA_LABEL</b>	Printable EBCDIC string	Application-specific nickname. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
<b>CKA_CERTIFICATE_TYPE</b>	CK_CERTIFICATE_TYPE	Always CKC_X_509. An application can specify the value when the object is created (or generated) only.
<b>CKA_TRUSTED</b>	CK_BBOOL	Always set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
<b>CKA_SUBJECT</b>	Byte array	DER-encoding as found in certificate. If not specified, ICSF sets it from the certificate. If specified, ICSF enforces that it matches the subject in the certificate. An application can specify the value when the object is created (or generated) only.
<b>CKA_ID</b>	Byte array	Key identifier. Default is empty. An application can set or change the value at any time.
<b>CKA_ISSUER</b>	Byte array	DER-encoding as found in certificate. If not specified, ICSF sets from the certificate. If specified, ICSF enforces that it matches the issuer in the certificate. An application can specify the value when the object is created (or generated) only.
<b>CKA_SERIAL_NUMBER</b>	Byte array	DER-encoding as found in certificate. If not specified, ICSF sets from the certificate. If specified, ICSF enforces that it matches the serial number in the certificate. An application can specify the value when the object is created (or generated) only.
<b>CKA_VALUE</b>	Byte array	This is the DER-encoding of the certificate. (Required.) An application can specify the value when the object is created (or generated) only.

Table 9. X.509 certificate object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31. (continued)

Attribute	Data type	Notes
<b>CKA_CERTIFICATE_CATEGORY</b>	CK_ULONG	<p>Categorization of the certificate:</p> <p><b>1</b> Token user</p> <p><b>2</b> Certificate authority</p> <p><b>3</b> Other entity</p> <p>If not specified, ICSF sets it to 2 if the certificate has the BasicConstraints CA flag on. Otherwise it is not set.</p> <p><b>Note:</b> If specified (or defaulted) to 2, the certificate is considered a CA certificate. The user must have appropriate authority.</p> <p>An application can set or change the value at any time.</p>
<b>CKA_APPLICATION</b>	Printable EBCDIC string	<p>Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page.</p> <p>An application can specify the value when the object is created (or generated) only.</p>
<b>CKA_IBM_DEFAULT</b> (vendor specific attribute - 0x80000002)	CK_BBOOL	<p>Default flag. Default is FALSE.</p> <p>An application can set or change the value at any time.</p>
<b>CKA_IBM_ICSF_HANDLE</b> (vendor specific attribute - 0x80010009)	Printable EBCDIC string	<p>44-character ICSF record locator value. This attribute may be retrieved via C_GetAttributeValue. It is not valid in the attribute list of any other function or callable service.</p>

Table 10. Secret key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_CLASS<sup>1</sup></b>	CKO_OBJECT_CLASS	<p>Object class (type).</p> <p>An application can specify the value when the object is created (or generated) only.</p>
<b>CKA_TOKEN<sup>11</sup></b>	CK_BBOOL	<p>Default value on create is FALSE. Object hardened to the TKDS if TRUE.</p> <p>An application can specify the value when the object is created (or generated) only.</p>
<b>CKA_PRIVATE<sup>11</sup></b>	CK_BBOOL	<p>Default value on create is TRUE.</p> <p>An application can specify the value when the object is created (or generated) only.</p>
<b>CKA_MODIFIABLE<sup>11</sup></b>	CK_BBOOL	<p>Default value is TRUE.</p> <p>An application can specify the value when the object is created (or generated) only.</p>

Table 10. Secret key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.  
(continued)

Attribute	Data type	Notes
<b>CKA_LABEL</b>	Printable EBCDIC string	Application-specific nickname. Default is empty. The string is assumed to come from the IBM1047 code page.  An application can set or change the value at any time.
<b>CKA_ID</b>	Byte array	Default is empty.  An application can set or change the value at any time.
<b>CKA_KEY_TYPE<sup>1, 5</sup></b>	CK_KEY_TYPE	Type of key: CKK_AES, CKK_BLOWFISH, CKK_DES, CKK_DES2, CKK_DES3, CKK_GENERIC_SECRET, CKK_IBM_CHACHA20, CKK_IBM_SM4, or CKK_RC4.  An application can specify the value when the object is created (or generated) only.
<b>CKA_START_DATE<sup>8</sup></b>	CK_DATE	Start date for the key. Default is empty.  An application can set or change the value at any time.
<b>CKA_END_DATE<sup>8</sup></b>	CK_DATE	End date for the key. Default is empty.  An application can set or change the value at any time.
<b>CKA_DERIVE<sup>8</sup></b>	CK_BBOOL	TRUE if key supports key derivation (other keys can be derived from this one). Default is TRUE.  An application can set or change the value at any time.
<b>CKA_LOCAL<sup>2, 4, 6</sup></b>	CK_BBOOL	TRUE only if key was generated locally.  Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
<b>CKA_GEN_MECHANISM<sup>2, 4, 6</sup></b>	CK_MECHANISM_TYPE	Identifier of the mechanism used to generate the key. Always CK_UNAVAILABLE_INFORMATION.  Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
<b>CKA_ENCRYPT<sup>8</sup></b>	CK_BBOOL	TRUE if key supports encryption <sup>9</sup> . Default is TRUE.  An application can set or change the value at any time.
<b>CKA_VERIFY<sup>8</sup></b>	CK_BBOOL	TRUE if key supports verification where the signature is an appendix to the data. Default is TRUE.  An application can set or change the value at any time.
<b>CKA_WRAP<sup>8</sup></b>	CK_BBOOL	TRUE if key supports wrapping (can be used to wrap other keys). <sup>9</sup> Default is TRUE.  An application can set or change the value at any time.

Table 10. Secret key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.  
(continued)

Attribute	Data type	Notes
<b>CKA_DECRYPT<sup>8</sup></b>	CK_BBOOL	TRUE if key supports decryption. <sup>9</sup> Default is TRUE. An application can set or change the value at any time.
<b>CKA_SIGN<sup>8</sup></b>	CK_BBOOL	TRUE if key supports signatures where the signature is an appendix to the data. <sup>9</sup> Default is TRUE. An application can set or change the value at any time.
<b>CKA_UNWRAP<sup>8</sup></b>	CK_BBOOL	TRUE if key supports unwrapping (can be used to unwrap other keys) <sup>9</sup> . Default is TRUE. An application can set or change the value at any time.
<b>CKA_EXTRACTABLE<sup>8</sup></b>	CK_BBOOL	TRUE if key is extractable. Caller can change from TRUE to FALSE only. Default is TRUE. An application can set or change the value, as per PKCS #11 restrictions.
<b>CKA_SENSITIVE<sup>8</sup></b>	CK_BBOOL	TRUE if key is sensitive. Caller can change from FALSE to TRUE only. Default is FALSE. An application can set or change the value, as per PKCS #11 restrictions. <b>Note:</b> When CKA_IBM_SECURE is TRUE, CKA_SENSITIVE is set TRUE.
<b>CKA_ALWAYS_SENSITIVE<sup>2, 4, 6</sup></b>	CK_BBOOL	TRUE if key has always had the CKA_SENSITIVE attribute set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
<b>CKA_NEVER_EXTRACTABLE<sup>2, 4, 6</sup></b>	CK_BBOOL	TRUE if key has never had the CKA_EXTRACTABLE attribute set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
<b>CKA_VALUE<sup>1, 4, 6, 7</sup></b>	Byte array	The key. Sensitive key part. An application can specify the value when the object is created (or generated) only.
<b>CKA_VALUE_LEN<sup>2, 3</sup></b>	CK_ULONG	Length of the key in bytes (AES, Blowfish, RC4, and Generic secret keys only). An application can specify the value when the object is generated only.
<b>CKA_APPLICATION</b>	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can specify the value when the object is created (or generated) only.

Table 10. Secret key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.  
(continued)

Attribute	Data type	Notes
<b>CKA_IBM_FIPS140</b> (vendor specific attribute 0x80000005)	CK_BBOOL	TRUE if the key must only be used in a FIPS 140-2 compliant fashion. The default value is FALSE.  An application can specify the value when the object is created (or generated) only.
<b>CKA_TRUSTED<sup>10, 14</sup></b>	CK_BBOOL	The wrapping key can be used to wrap keys with CKA_WRAP_WITH_TRUSTED set to CK_TRUE. Always set CK_FALSE when the key is created or generated. May be set to CK_TRUE via C_SetAttributeValue, with restrictions
<b>CKA_WRAP_WITH_TRUSTED<sup>12</sup></b>	CK_BBOOL	CK_TRUE if the key can only be wrapped with a wrapping key that has CKA_TRUSTED set to CK_TRUE. Default is CK_FALSE
<b>CKA_CHECK_VALUE</b>	Byte array	3-byte key checksum  The attribute has no value (0 length) for clear keys or for secure keys created prior to ICSF FMID HCR77B1. Otherwise, normal PKCS #11 processing rules apply.
<b>CKA_IBM_REGIONAL<sup>6, 12</sup></b> (vendor specific attribute - 0x80050000)	CK_BBOOL	If TRUE, key is for a regional cryptographic server.  The default value will be TRUE for key types supported by the regional cryptographic servers and FALSE for all others.  When this is set TRUE by the caller, the key type must be one that is supported by the regional cryptographic servers.
<b>CKA_IBM_SECURE<sup>6, 11, 12, 13</sup></b> (vendor specific attribute - 0x80000006)	CK_BBOOL	CK_TRUE if the key is an Enterprise PKCS #11 coprocessor or a regional cryptographic server secure key.  A secure key may be requested by setting this attribute CK_TRUE during key creation or generation. For key generation, if set CK_FALSE or not specified at all, ICSF will determine the security.  When this is set TRUE by the caller or ICSF, the key is treated as sensitive (CKA_SENSITIVE is set TRUE).
<b>CKA_IBM_ALWAYS_SECURE<sup>2, 4, 6</sup></b> (vendor specific attribute - 0x80000008)	CK_BBOOL	CK_TRUE if key has always had the CKA_IBM_SECURE attribute set to CK_TRUE. Only applicable to secure keys. This attribute will not have a value for clear keys.

Table 10. Secret key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.  
(continued)

Attribute	Data type	Notes
<b>CKA_IBM_CARD_COMPLIANCE<sup>2, 4, 6, 15</sup></b> (vendor specific attribute - 0x80000007)	CK_ULONG	A bit mask field indicating the Enterprise PKCS #11 coprocessor compliance mode of the secure key:  n/a 0 (Regional cryptographic server key) FIPS2009 1 BSI2009 2 FIPS2011 4 BSI2011 8 BSICC2017 64  Only applicable to secure keys. This attribute will not have a value for clear keys. When changing the value, the new value must specify the current mode or modes and any additional new modes. Changing the value to 0 results in the compliance mode being reset to the current compliance mode of the coprocessor.
<b>CKA_IBM_ATTRBOUND<sup>2, 6</sup></b> (vendor specific attribute - 0x80010004)	CK_BBOOL	CK_TRUE has the following meaning: <ul style="list-style-type: none"> <li>• The key must only be exported with its boolean usage attributes.</li> <li>• The key may be used as a signing or verification key for attribute bound wrap/unwrap.</li> <li>• The key may be used as a wrapping or unwrapping key for attribute bound wrap/unwrap.</li> <li>• The key may not be used as a wrapping key for non-attribute bound wrap.</li> </ul> May only be set CK_TRUE during key generation. The default value is CK_FALSE.  When this is set TRUE by the caller, the key will be a secure key (CKA_IBM_SECURE is set TRUE).  Only applicable to Enterprise PKCS #11 secure keys. This attribute will not have a value for clear keys.
<b>CKA_IBM_ICSF_HANDLE</b> (vendor specific attribute - 0x80010009)	Printable EBCDIC string	44-character ICSF record locator value. This attribute may be retrieved via C_GetAttributeValue. It is not valid in the attribute list of any other function or callable service.
<b>CKA_IBM_PROTKEY_EXTRACTABLE<sup>11</sup></b> (vendor specific attribute - 0x8001000C)	CK_BBOOL	CK_TRUE if the key is an Enterprise PKCS #11 coprocessor secure key capable of being used as a protected key. This attribute may only be CK_TRUE for AES and DES3 keys.  A protected-key-capable secure key may be requested by setting this attribute to CK_TRUE during key creation, generation, or derivation. If explicitly set to CK_FALSE, a secure key will be generated that may never be used for protected key operations. If not specified, either a clear or secure key that may not be used as a protected key will be generated, created, or derived, depending on the value of CKA_IBM_SECURE.  The value may only be changed from CK_TRUE to CK_FALSE.

Table 10. Secret key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31. (continued)

Attribute	Data type	Notes
<b>CKA_IBM_PROTKEY_NEVER_EXTRACTABLE</b> 2, 4, 6 (vendor specific attribute - 0x8001000D)	CK_BBOOL	CK_TRUE if key has never had the CKA_IBM_PROTKEY_EXTRACTABLE attribute set to CK_TRUE.

Table 11. Public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_CLASS</b> <sup>1</sup>	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
<b>CKA_TOKEN</b> <sup>11</sup>	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.
<b>CKA_PRIVATE</b> <sup>11</sup>	CK_BBOOL	Default value on create is FALSE. An application can specify the value when the object is created (or generated) only.
<b>CKA_MODIFIABLE</b> <sup>11</sup>	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
<b>CKA_LABEL</b>	Printable EBCDIC string	Application-specific nickname. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
<b>CKA_TRUSTED</b> <sup>10, 14</sup>	CK_BBOOL	The wrapping key can be used to wrap keys with CKA_WRAP_WITH_TRUSTED set to CK_TRUE. Always set CK_FALSE when the key is created or generated. May be set to CK_TRUE via C_SetAttributeValue, with restrictions
<b>CKA_SUBJECT</b>	Byte array	DER-encoding. Default empty. An application can set or change the value at any time.
<b>CKA_ID</b>	Byte array	Key identifier. Default empty. An application can set or change the value at any time.
<b>CKA_KEY_TYPE</b> <sup>1, 5</sup>	CK_KEY_TYPE	Type of key. CKK_DH, CKK_DSA, CKK_EC, CKK_RSA, CKK_IBM_SM2, and CKK_IBM_DILITHIUM only. An application can specify the value when the object is created (or generated) only.
<b>CKA_START_DATE</b> <sup>8</sup>	CK_DATE	Start date for the key. Default empty. An application can set or change the value at any time.
<b>CKA_END_DATE</b> <sup>8</sup>	CK_DATE	End date for the key. Default empty. An application can set or change the value at any time.

Table 11. Public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.  
(continued)

Attribute	Data type	Notes
<b>CKA_DERIVE</b> <sup>8,9</sup>	CK_BBOOL	TRUE if key supports key derivation (if other keys can be derived from this one). Default is TRUE. An application can set or change the value at any time.
<b>CKA_LOCAL</b> <sup>2, 4, 6</sup>	CK_BBOOL	TRUE only if key was generated locally. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
<b>CKA_KEY_GEN_MECHANISM</b> <sup>2, 4, 6</sup>	CK_MECHANISM_TYPE	Identifier of the mechanism used to generate the key. Always CK_UNAVAILABLE_INFORMATION. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
<b>CKA_ENCRYPT</b> <sup>8,9</sup>	CK_BBOOL	TRUE if key supports encryption. Default is TRUE. An application can set or change the value at any time.
<b>CKA_VERIFY</b> <sup>8,9</sup>	CK_BBOOL	TRUE if key supports verification where the signature is an appendix to the data. Default is TRUE. An application can set or change the value at any time.
<b>CKA_VERIFY_RECOVER</b> <sup>8,9</sup>	CK_BBOOL	TRUE if key supports verification where the data is recovered from the signature. Default is TRUE. An application can set or change the value at any time.
<b>CKA_WRAP</b> <sup>8,9</sup>	CK_BBOOL	TRUE if key supports wrapping (can be used to wrap other keys). Default is TRUE. An application can set or change the value at any time.
<b>CKA_APPLICATION</b>	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can specify the value when the object is created (or generated) only.
<b>CKA_IBM_FIPS140</b> (vendor specific attribute - 0x80000005)	CK_BBOOL	TRUE if the key must only be used in a FIPS 140-2 compliant fashion. The default value is FALSE. An application can specify the value when the object is created (or generated) only.
<b>CKA_IBM_REGIONAL</b> <sup>6, 12</sup> (vendor specific attribute - 0x80050000)	CK_BBOOL	If TRUE, key is for a regional cryptographic server. The default value will be TRUE for key types supported by the regional cryptographic servers and FALSE for all others. When this is set TRUE by the caller, the key type must be one that is supported by the regional cryptographic servers.



Table 11. Public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.  
(continued)

Attribute	Data type	Notes
<b>CKA_IBM_SECURE<sup>11, 12, 13</sup></b> (vendor specific attribute - <b>0x80000006</b> )	CK_BBOOL	CK_TRUE if the key is an Enterprise PKCS #11 coprocessor or a regional cryptographic server secure key.  A secure key may be requested by setting this attribute CK_TRUE during key creation or generation.  For key-pair generation, if set CK_FALSE or not specified at all, ICSF will determine the security. If set CK_TRUE by the caller or ICSF, the matching private key will also be a secure key.
<b>CKA_IBM_CARD COMPLIANCE<sup>2, 4, 6, 15</sup></b> (vendor specific attribute - <b>0x80000007</b> )	CK_ULONG	A bit mask field indicating the Enterprise PKCS #11 coprocessor compliance mode of the secure key:  n/a 0 (Regional cryptographic server key) FIPS2009 1 BSI2009 2 FIPS2011 4 BSI2011 8 BSICC2017 64  Only applicable to secure keys. This attribute will not have a value for clear keys. When changing the value, the new value must specify the current mode or modes and any additional new modes. Changing the value to 0 results in the compliance mode being reset to the current compliance mode of the coprocessor.
<b>CKA_IBM_ATTRBOUND<sup>11, 12, 13</sup></b> (vendor specific attribute - <b>0x80010004</b> )	CK_BBOOL	CK_TRUE if the key may be used as a wrapping key to export attribute bound secret keys. (Such public keys may not be used for non-attribute bound wrap.) CK_TRUE also means that the key may be used as an attribute bound unwrap verification key. The default value is CK_FALSE.  An attribute bound key may be requested by setting this attribute CK_TRUE during key creation or generation.  When this is set TRUE by the caller, the key will be a secure key (CKA_IBM_SECURE is set TRUE).  Only applicable to Enterprise PKCS #11 secure keys. This attribute will not have a value for clear keys.
<b>CKA_IBM_ICSF_HANDLE</b> (vendor specific attribute - <b>0x80010009</b> )	Printable EBCDIC string	44-character ICSF record locator value. This attribute may be retrieved via C_GetAttributeValue. It is not valid in the attribute list of any other function or callable service.

Table 12. RSA public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_MODULUS<sup>1, 4</sup></b>	Big integer	Modulus $n$  An application can specify the value when the object is created (or generated) only.

Table 12. RSA public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31. (continued)

Attribute	Data type	Notes
<b>CKA_MODULUS_BITS<sup>2,3</sup></b>	CK_ULONG	Length in bits of modulus $n$ An application can specify the value when the object is created (or generated) only.
<b>CKA_PUBLIC_EXPONENT<sup>1</sup></b>	Big integer	Public exponent $e$ An application can specify the value when the object is created (or generated) only.

Table 13. DSA public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_PRIME<sup>1,3</sup></b>	Big integer	Prime $p$ (512 to 2048 bits in steps of 64 bits)
<b>CKA_SUBPRIME<sup>1,3</sup></b>	Big integer	Subprime $q$ (160 bits for $p \leq 1024$ bits, 224 bits or 256 bits for $p > 1024$ bits)
<b>CKA_BASE<sup>1,3</sup></b>	Big integer	Base $g$
<b>CKA_VALUE<sup>1,4</sup></b>	Big integer	Public value $y$

Table 14. Diffie-Hellman public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_PRIME<sup>1,3</sup></b>	Big integer	Prime $p$ (512 to 2048 bits in steps of 64 bits)
<b>CKA_BASE<sup>1,3</sup></b>	Big integer	Base $g$
<b>CKA_VALUE<sup>1,4</sup></b>	Big integer	Public value $y$

Table 15. Elliptic Curve public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_EC_PARAMS<sup>1,3</sup></b> (CKA_ECDSA_PARAMS)	Byte array	DER-encoding of an ANSI X9.62 Parameters value
<b>CKA_EC_POINT<sup>1,4</sup></b>	Byte array	DER-encoding of an ANSI X9.62 ECPoint value $Q$

Table 16. SM2 public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_EC_PARAMS<sup>2,4</sup></b>	Byte array	DER-encoding of an ANSI X9.62 Parameters value
<b>CKA_EC_POINT<sup>1,4</sup></b>	Byte array	DER-encoding of an ANSI X9.62 ECPoint value $Q$

Table 17. Private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_CLASS<sup>1</sup></b>	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
<b>CKA_TOKEN<sup>11</sup></b>	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE. An application can specify the value when the object is created (or generated) only.
<b>CKA_PRIVATE<sup>11</sup></b>	CK_BBOOL	Default value on create is TRUE. An application can specify the value when the object is created (or generated) only.
<b>CKA_MODIFIABLE<sup>11</sup></b>	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
<b>CKA_LABEL</b>	Printable EBCDIC string	Application-specific nickname. Default is empty. The string is assumed to come from the IBM1047 code page. An application can set or change the value at any time.
<b>CKA_SUBJECT</b>	Byte array	DER-encoding. An application can set or change the value at any time.
<b>CKA_ID</b>	Byte array	Default is empty. An application can set or change the value at any time.
<b>CKA_KEY_TYPE<sup>1, 5</sup></b>	CK_KEY_TYPE	Type of key. CKK_EC, CKK_DH, CKK_DSA, CKK_RSA, CKK_IBM_SM2, and CKK_IBM_DILITHIUM only. An application can specify the value when the object is created (or generated) only.
<b>CKA_START_DATE<sup>8</sup></b>	CK_DATE	Start date for the key. Default empty. An application can set or change the value at any time.
<b>CKA_END_DATE<sup>8</sup></b>	CK_DATE	End date for the key. Default empty. An application can set or change the value at any time.
<b>CKA_DERIVE<sup>8,9</sup></b>	CK_BBOOL	TRUE if key supports key derivation (if other keys can be derived from this one). Default is TRUE. An application can set or change the value at any time.
<b>CKA_LOCAL<sup>2, 4, 6</sup></b>	CK_BBOOL	TRUE only if key was generated locally. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
<b>CKA_KEY_GEN_MECHANISM<sup>2, 4, 6</sup></b>	CK_MECHANISM_TYPE	Identifier of the mechanism used to generate the key material. Always CK_UNAVAILABLE_INFORMATION. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
<b>CKA_DECRYPT<sup>8,9</sup></b>	CK_BBOOL	TRUE if key supports decryption. Default is TRUE. An application can set or change the value at any time.

Table 17. Private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31. (continued)

Attribute	Data type	Notes
<b>CKA_SIGN<sup>8,9</sup></b>	CK_BBOOL	TRUE if key supports signatures where the signature is an appendix to the data. Default is TRUE. An application can set or change the value at any time.
<b>CKA_SIGN_RECOVER<sup>8,9</sup></b>	CK_BBOOL	TRUE if key supports signatures where the data can be recovered from the signature. Default is TRUE. An application can set or change the value at any time.
<b>CKA_UNWRAP<sup>8,9</sup></b>	CK_BBOOL	TRUE if key supports unwrapping (can be used to unwrap other keys). Default is TRUE. An application can set or change the value at any time.
<b>CKA_EXTRACTABLE<sup>8</sup></b>	CK_BBOOL	TRUE if key is extractable. Default is TRUE. An application can set or change the value, as per PKCS #11 restrictions. Caller can change from TRUE to FALSE only.
<b>CKA_SENSITIVE<sup>8</sup></b>	CK_BBOOL	TRUE if key is sensitive. Default is FALSE. An application can set or change the value, as per PKCS #11 restrictions. Caller can change from FALSE to TRUE only. <b>Note:</b> When CKA_IBM_SECURE is TRUE, CKA_SENSITIVE is set TRUE.
<b>CKA_ALWAYS_SENSITIVE<sup>2,4,6</sup></b>	CK_BBOOL	TRUE if key has always had the CKA_SENSITIVE attribute set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
<b>CKA_NEVER_EXTRACTABLE<sup>2,4,6</sup></b>	CK_BBOOL	TRUE if key has never had the CKA_EXTRACTABLE attribute set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
<b>CKA_APPLICATION</b>	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page. An application can specify the value when the object is created (or generated) only.
<b>CKA_IBM_FIPS140 (vendor specific attribute 0x80000005)</b>	CK_BBOOL	TRUE if the key must only be used in a FIPS 140-2 compliant fashion. The default value is FALSE. An application can specify the value when the object is created (or generated) only.
<b>CKA_WRAP_WITH_TRUSTED<sup>12</sup></b>	CK_BBOOL	CK_TRUE if the key can only be wrapped with a wrapping key that has CKA_TRUSTED set to CK_TRUE. Default is CK_FALSE

Table 17. Private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31. (continued)

Attribute	Data type	Notes
<b>CKA_IBM_REGIONAL<sup>2, 6, 12</sup></b> (vendor specific attribute - 0x80050000)	CK_BBOOL	<p>If TRUE, key is for a regional cryptographic server.</p> <p>The default value will be TRUE for key types supported by the regional cryptographic server and FALSE for all others.</p> <p>When this is set TRUE by the caller, the key type must be one that is supported by the regional cryptographic servers.</p>
<b>CKA_IBM_SECURE<sup>6, 11, 12, 13</sup></b> (vendor specific attribute - 0x80000006)	CK_BBOOL	<p>CK_TRUE if the key is an Enterprise PKCS #11 coprocessor or a regional cryptographic server secure key.</p> <p>A secure key may be requested by setting this attribute CK_TRUE during key creation or generation.</p> <p>For key-pair generation, if set CK_TRUE by the caller or ICSF, the matching public key will also be a secure key. If set CK_FALSE or not specified at all, ICSF will determine the security.</p> <p>When this is set TRUE by the caller or ICSF, the key is treated as sensitive (CKA_SENSITIVE is set TRUE).</p> <p>May be changed from FALSE to TRUE during C_CopyObject.</p>
<b>CKA_IBM_ALWAYS_SECURE<sup>2, 4, 6</sup></b> (vendor specific attribute - 0x80000008)	CK_BBOOL	<p>CK_TRUE if key has always had the CKA_IBM_SECURE attribute set to CK_TRUE. Only applicable to secure keys. This attribute will not have a value for clear keys.</p>
<b>CKA_IBM_CARD_COMPLIANCE<sup>2, 4, 6, 15</sup></b> (vendor specific attribute - 0x80000007)	CK_ULONG	<p>A bit mask field indicating the compliance mode of the Enterprise PKCS #11 coprocessor at the time the secure key was created:</p> <p>n/a 0 (Regional cryptographic server key)            FIPS2009 1            BSI2009 2            FIPS2011 4            BSI2011 8            BSICC2017 64</p> <p>Only applicable to secure keys. This attribute will not have a value for clear keys. When changing the value, the new value must specify the current mode or modes and any additional new modes. Changing the value to 0 results in the compliance mode being reset to the current compliance mode of the coprocessor.</p>

Table 17. Private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31. (continued)

Attribute	Data type	Notes
<b>CKA_IBM_ATTRBOUND<sup>2,6</sup></b> (vendor specific attribute - 0x80010004)	CK_BBOOL	<p>CK_TRUE has the following meaning:</p> <ul style="list-style-type: none"> <li>The key must only be exported with its boolean usage attributes.</li> <li>The key may be used as a signing key for attribute bound wrap.</li> <li>The key may be used as an unwrapping key for attribute bound unwrap.</li> </ul> <p>May only be set CK_TRUE during key generation. The default value is CK_FALSE.</p> <p>When this is set TRUE by the caller, the key will be a secure key (CKA_IBM_SECURE is set TRUE).</p> <p>Only applicable to Enterprise PKCS #11 secure keys. This attribute will not have a value for clear keys.</p>
<b>CKA_IBM_ICSF_HANDLE</b> (vendor specific attribute - 0x80010009)	Printable EBCDIC string	44-character ICSF record locator value. This attribute may be retrieved via C_GetAttributeValue. It is not valid in the attribute list of any other function or callable service.

Table 18. RSA private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_MODULUS<sup>1,4,6</sup></b>	Big integer	<p>Modulus <math>n</math></p> <p>An application can specify the value when the object is created (or generated) only.</p>
<b>CKA_PUBLIC_EXPONENT<sup>4,6</sup></b>	Big integer	<p>Public exponent <math>e</math></p> <p>An application can specify the value when the object is created (or generated) only.</p>
<b>CKA_PRIVATE_EXPONENT<sup>1,4,6,7</sup></b>	Big integer	<p>Private exponent <math>d</math></p> <p>Sensitive key part.</p> <p>An application can specify the value when the object is created (or generated) only.</p>
<b>CKA_PRIME_1<sup>4,6,7</sup></b>	Big integer	<p>Prime <math>p</math></p> <p>Sensitive key part.</p> <p>An application can specify the value when the object is created (or generated) only.</p>
<b>CKA_PRIME_2<sup>4,6,7</sup></b>	Big integer	<p>Prime <math>q</math></p> <p>Sensitive key part.</p> <p>An application can specify the value when the object is created (or generated) only.</p>

Table 18. RSA private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31. (continued)

Attribute	Data type	Notes
<b>CKA_EXPONENT_1</b> <sup>4, 6, 7</sup>	Big integer	Private exponent $d$ modulo $p-1$ Sensitive key part. An application can specify the value when the object is created (or generated) only.
<b>CKA_EXPONENT_2</b> <sup>4, 6, 7</sup>	Big integer	Private exponent $d$ modulo $q-1$ Sensitive key part. An application can specify the value when the object is created (or generated) only.
<b>CKA_COEFFICIENT</b> <sup>4, 6, 7</sup>	Big integer	CRT coefficient $q^{-1} \bmod p$ Sensitive key part. An application can specify the value when the object is created (or generated) only.

Table 19. DSA private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_PRIME</b> <sup>1,4,6</sup>	Big integer	Prime $p$ (512 to 2048 bits in steps of 64 bits)
<b>CKA_SUBPRIME</b> <sup>1,4,6</sup>	Big integer	Subprime $q$ (160 bits for $p \leq 1024$ bits, 224 bits or 256 bits for $p > 1024$ bits)
<b>CKA_BASE</b> <sup>1,4,6</sup>	Big integer	Base $g$
<b>CKA_VALUE</b> <sup>1,4,6,7</sup>	Big integer	Private value $x$

Table 20. Diffie-Hellman private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_PRIME</b> <sup>1,4,6</sup>	Big integer	Prime $p$ (512 to 2048 bits in steps of 64 bits)
<b>CKA_BASE</b> <sup>1,4,6</sup>	Big integer	Base $g$
<b>CKA_VALUE</b> <sup>1,4,6,7</sup>	Big integer	Private value $x$
<b>CKA_VALUE_BITS</b> <sup>2,6</sup>	CK_ULONG	Length in bits of private value $x$ . For non-FIPS or when prime bit size = 1024, the default is 160. For FIPS prime bit size = 2048, the default is 256

Table 21. Elliptic Curve private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_EC_PARAMS</b> <sup>1,4,6</sup> ( <b>CKA_ECDSA_PARAMS</b> )	Byte array	DER-encoding of an ANSI X9.62 Parameters value
<b>CKA_VALUE</b> <sup>1,4,6,7</sup>	Big integer	ANSI X9.62 private value $d$

Table 21. Elliptic Curve private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31. (continued)

Attribute	Data type	Notes
<b>CKA_IBM_PROTKEY_EXTRACTABLE 11</b> (vendor specific attribute - 0x8001000C)	CK_BBOOL	CK_TRUE if the key is an Enterprise PKCS #11 coprocessor secure key capable of being used as a protected key. This attribute may only be CK_TRUE for secp256r1, secp384r1, secp521r1, Ed448, and Ed25519 private keys.  A protected-key-capable secure key may be requested by setting this attribute to CK_TRUE during key creation, generation, or derivation. If explicitly set to CK_FALSE, a secure key will be generated that may never be used for protected key operations. If not specified, either a clear or secure key that may not be used as a protected key will be generated, created, or derived, depending on the value of CKA_IBM_SECURE.  The value may only be changed from CK_TRUE to CK_FALSE.
<b>CKA_IBM_PROTKEY_NEVER_EXTRACTABLE 2, 4, 6</b> (vendor specific attribute - 0x8001000D)	CK_BBOOL	CK_TRUE if key has never had the CKA_IBM_PROTKEY_EXTRACTABLE attribute set to CK_TRUE.

Table 22. SM2 private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_EC_PARAMS<sup>2,4,6</sup></b>	Byte array	DER-encoding of an ANSI X9.62 Parameters value
<b>CKA_VALUE<sup>2,4,6,7</sup></b>	Byte array	ANSI X9.62 private value <i>d</i>

Table 23. Domain parameter object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_CLASS<sup>1</sup></b>	CKO_OBJECT_CLASS	Object class (type).  An application can specify the value when the object is created (or generated) only.
<b>CKA_TOKEN<sup>11</sup></b>	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE.
<b>CKA_PRIVATE<sup>11</sup></b>	CK_BBOOL	Default value on create is FALSE
<b>CKA_MODIFIABLE<sup>11</sup></b>	CK_BBOOL	Default value is TRUE
<b>CKA_LABEL</b>	Printable EBCDIC string	Application-specific nickname. Default is empty. The string is assumed to come from the IBM1047 code page.  An application can set or change the value at any time.



Table 23. Domain parameter object attributes that ICSF supports. For the meanings of the footnotes, see [Table 7](#) on page 31. (continued)

Attribute	Data type	Notes
<b>CKA_KEY_TYPE<sup>1</sup></b>	CK_KEY_TYPE	Type of key the domain parameters can be used to generate. CKK_DSA and CKK_DH only in this release
<b>CKA_LOCAL<sup>2,4</sup></b>	CK_BBOOL	TRUE only if the parameters were generated locally
<b>CKA_APPLICATION</b>	Printable EBCDIC string	Description of the application that created the object. Default is empty. The string is assumed to come from the IBM1047 code page.
<b>CKA_IBM_ICSF_HANDLE (vendor specific attribute - 0x80010009)</b>	Printable EBCDIC string	44-character ICSF record locator value. This attribute may be retrieved via C_GetAttributeValue. It is not valid in the attribute list of any other function or callable service.

Table 24. DSA domain parameter object attributes that ICSF supports. For the meanings of the footnotes, see [Table 7](#) on page 31.

Attribute	Data type	Notes
<b>CKA_PRIME<sup>1,4</sup></b>	Big integer	Prime $p$ (512 to 2048 bits in steps of 64 bits)
<b>CKA_SUBPRIME<sup>1,4</sup></b>	Big integer	Subprime $q$ (160 bits for $p \leq 1024$ bits, 224 bits or 256 bits for $p > 1024$ bits)
<b>CKA_BASE<sup>1,4</sup></b>	Big integer	Base $g$
<b>CKA_PRIME_BITS<sup>2,3</sup></b>	CK_ULONG	Length of the prime value

Table 25. Diffie-Hellman domain parameter object attributes that ICSF supports. For the meanings of the footnotes, see [Table 7](#) on page 31.

Attribute	Data type	Notes
<b>CKA_PRIME<sup>1,4</sup></b>	Big integer	Prime $p$ (512 to 2048 bits in steps of 64 bits)
<b>CKA_BASE<sup>1,4</sup></b>	Big integer	Base $g$
<b>CKA_PRIME_BITS<sup>2,3</sup></b>	CK_ULONG	Length of the prime value

Table 26. Dilithium public key object attributes that ICSF supports. For the meanings of the footnotes, see [Table 7](#) on page 31.

Attribute	Data type	Notes
<b>CKA_IBM_DILITHIUM_MODE<sup>1,3</sup></b>	Byte array	DER-encoding of a Dilithium key mode.  The value is returned left-justified in a 16-byte field padded with zeros.
<b>CKA_VALUE<sup>1,4</sup></b>	Big integer	Public value.

Table 27. Dilithium private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 7 on page 31.

Attribute	Data type	Notes
<b>CKA_IBM_DILITHIUM_MODE</b> 1,4,6	Byte array	DER-encoding of a Dilithium key mode. The value is returned left-justified in a 16-byte field padded with zeros.
<b>CKA_VALUE</b> 1,4,6,7	Big integer	Private value.

## Library, slot, and token information

PKCS #11 maintains information about the library code, slots, and tokens, which can be set and queried by calling the library functions. For z/OS, this information is as follows:

CK\_INFO - Returned by C\_GetInfo. not modifiable by applications.

- cryptokiVersion - 2.20
- manufacturerID - "IBM Corp. "
- libraryDescription - "z/OS PKCS11 library "
- libraryVersion - 7.70

CK\_SLOT\_INFO - Returned by C\_GetSlotInfo. not modifiable by applications

- slotDescription - "z/OS PKCS11 - virtual smart card "
- manufacturerID - "IBM Corp. "
- flags - for any slot returned by C\_GetSlotList the following flags are set:
  - CKF\_TOKEN\_PRESENT=ON
  - CKF\_REMOVABLE\_DEVICE=ON
  - CKF\_HW\_SLOT=OFF
- hardwareVersion - 7.70
- firmwareVersion - 7.70

CK\_TOKEN\_INFO - Set by C\_InitToken, Returned by C\_GetTokenInfo

- label - As specified
- manufacturerID - "z/OS PKCS11 API " (Might be set to other values if the token was initialized outside of the C API.)
- model - "HCR7770 " (coincides with the release that the token was created) (Might be set to other values if the token was initialized outside of the C API.)
- serialNumber - "0 " (Might be set to other values if the token was initialized outside of the C API.)
- flags - the following flags are set ON for any initialized token. All others are OFF:
  - CKF\_RNG
  - CKF\_PROTECTED\_AUTHENTICATION\_PATH
  - CKF\_TOKEN\_INITIALIZED
  - CKF\_USER\_PIN\_INITIALIZED
- ulMaxSessionCount - CK\_UNAVAILABLE\_INFORMATION
- ulSessionCount - CK\_UNAVAILABLE\_INFORMATION
- ulMaxRwSessionCount - CK\_UNAVAILABLE\_INFORMATION
- ulRwSessionCount - CK\_UNAVAILABLE\_INFORMATION
- ulMaxPinLen - CK\_UNAVAILABLE\_INFORMATION

- ulMinPinLen - 0
- ulTotalPublicMemory - CK\_UNAVAILABLE\_INFORMATION
- ulFreePublicMemory - CK\_UNAVAILABLE\_INFORMATION
- ulTotalPrivateMemory - CK\_UNAVAILABLE\_INFORMATION
- ulFreePrivateMemory - CK\_UNAVAILABLE\_INFORMATION
- hardwareVersion - 7.70
- firmwareVersion - 7.70
- utcTime - GMT date and time that the token was last updated

CK\_SESSION\_INFO - Returned by C\_GetSessionInfo

- slotId - The slot in question
- state - CK\_UNAVAILABLE\_INFORMATION
- flags - As defined by the PKCS #11 specification
- ulDeviceError - A mapping of the last failing ICSF return and reason code values related to this session. For more information see [“Function return codes”](#) on page 70.

## Functions supported

ICSF supports a subset of the standard PKCS #11 functions, and several non-standard functions.

### Standard functions supported

Table 28 on page 51 lists the standard PKCS #11 functions that ICSF supports. Any function not listed is not supported and returns the CKR\_FUNCTION\_NOT\_SUPPORTED return code.

<i>Table 28. Standard PKCS #11 functions that ICSF supports</i>	
Function	Usage notes
<b>General purpose functions:</b>	
<b>C_Initialize()</b>	<ul style="list-style-type: none"> <li>• The library always uses OS locking for thread serialization. Therefore, if C_Initialize is called with the CreateMutex, DestroyMutex, LockMutex, and UnlockMutex function pointer arguments set and the CKF_OS_LOCKING_OK flag is not set, C_Initialize fails and returns the value CKR_CANT_LOCK.</li> <li>• When C_Initialize is called, the application-specific set of (virtual) slot IDs is allocated, one for each preexisting token that the application is authorized to use. (See the descriptions of C_GetSlotList and C_WaitForSlotEvent for information on how this set can increase in size.) The one exception to this occurs when C_Initialize is called by a child process after fork. If the PKCS #11 environment is inherited by the child process, the slot list and token state is not refreshed.</li> <li>• A call to C_Initialize() from an application that is not running POSIX-enabled results in error CKR_FUNCTION_FAILED being returned.</li> </ul>
<b>C_Finalize()</b>	dlclose() cannot be used as an implicit C_Finalize(). If an application uses dlclos() without calling C_Finalize(), and reinitializes PKCS #11, a subsequent call to C_Initialize() will result in error CKR_FUNCTION_FAILED being returned.
<b>C_GetInfo()</b>	
<b>C_GetFunctionList()</b>	

Table 28. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
<b>Slot and token management functions:</b>	
<b>C_GetSlotList()</b>	<ul style="list-style-type: none"> <li>• If the pSlotList argument is NULL, this function returns only the number of allocated slots. In the process of returning this number C_GetSlotList searches for new tokens to which the application has access. If new tokens are found, slot IDs are allocated for them. This search is only performed if at least 5 seconds has passed since the last search was made.</li> <li>• If the pSlotList argument is non-NULL, this function returns the current list of virtual slot IDs. No attempt is made to discover new tokens created by other applications.</li> <li>• The tokenPresent argument flag is meaningless as all allocated slots have a token present.</li> </ul>
<b>C_GetSlotInfo()</b>	
<b>C_GetTokenInfo()</b>	
<b>C_WaitForSlotEvent()</b>	<ul style="list-style-type: none"> <li>• This function is used to dynamically allocate an additional slot in order to create a new token. There are no other slot events. The newly allocated slot ID is returned as the pSlot argument.</li> <li>• The CKF_DONT_BLOCK argument flag is meaningless because this function never blocks. The dynamic slot allocation occurs synchronously.</li> </ul>
<b>C_GetMechanismList()</b>	<p>The list of functions returned reflects the capabilities of the current cryptographic hardware configuration.</p> <p><b>Note:</b> The loss or addition of hardware on the fly is not detected or reflected. (For example, on a z9-109, if the only CEX2C present is deactivated, this function still returns the mechanisms that require an active CEX2C to function.)</p>
<b>C_GetMechanismInfo()</b>	<p>The output of this function reflects the capabilities of the current cryptographic hardware configuration.</p>
<b>C_InitToken()</b>	<p>Tokens are protected by the security manager through profiles in the CRYPTOZ class. PINs are not used. The pPin and ulPinLen arguments are ignored.</p>
<b>C_InitPIN()</b>	<p>Tokens are protected by the security manager through profiles in the CRYPTOZ class. PINs are not used. This function performs no operation and always returns CKR_OK.</p>
<b>C_SetPIN()</b>	<p>Tokens are protected by the security manager through profiles in the CRYPTOZ class. PINs are not used. This function performs no operation and always returns CKR_OK.</p>
<b>Session management functions:</b>	
<b>C_OpenSession()</b>	<p>The Notify and pApplication arguments are ignored.</p>
<b>C_CloseSession()</b>	
<b>C_CloseAllSessions()</b>	

Table 28. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
<b>C_GetSessionInfo()</b>	The state field returned is meaningless. It is always set to CK_UNAVAILABLE_INFORMATION.
<b>C_GetOperationState()</b>	Returns CKR_STATE_UNSAVEABLE if a find is active or more than one cryptographic operation is active.
<b>C_SetOperationState()</b>	
<b>C_Login()</b>	Tokens are protected by the security manager through profiles in the CRYPTOZ class. Applications are always logged in to the security manager. PINs are not used. This function has no effect on the session state and always returns CKR_OK.
<b>C_Logout()</b>	Tokens are protected by the security manager through profiles in the CRYPTOZ class. Applications are always logged in to the security manager. PINs are not used. This function has no effect on the session state and always returns CKR_OK.
<b>Object management functions:</b>	
<b>C_CreateObject()</b>	
<b>C_CopyObject()</b>	
<b>C_DestroyObject()</b>	
<b>C_GetObjectSize()</b>	
<b>C_GetAttributeValue()</b>	
<b>C_SetAttributeValue()</b>	
<b>C_FindObjectsInit()</b>	ulCount may have the high order bit on to specify to use the RTL rule array keyword when invoking the ICSF CSFPTRL callable service.
<b>C_FindObjects()</b>	Sensitive attributes cannot be used as search criteria when the object is marked sensitive or not exportable. Doing so results in no match found.
<b>C_FindObjectsFinal()</b>	
<b>Encryption functions:</b>	

Table 28. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
<b>C_EncryptInit()</b>	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> <li>• CKM_DES_ECB</li> <li>• CKM_DES_CBC</li> <li>• CKM_DES_CBC_PAD</li> <li>• CKM_DES3_ECB</li> <li>• CKM_DES3_CBC</li> <li>• CKM_DES3_CBC_PAD</li> <li>• CKM_RSA_PKCS</li> <li>• CKM_RSA_X_509</li> <li>• CKM_AES_CBC</li> <li>• CKM_AES_ECB</li> <li>• CKM_AES_CBC_PAD</li> <li>• CKM_AES_CTS</li> <li>• CKM_AES_GCM (Limited to single part encryption only and for no more than 1048576 bytes of clear text.)</li> <li>• CKM_BLOWFISH_CBC</li> <li>• CKM_RC4</li> <li>• CKM_IBM_SM4_ECB</li> <li>• CKM_IBM_SM4_CBC</li> </ul> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>• A secure key may not be used for mechanisms CKM_AES_CTS, CKM_AES_GCM, or GCMIVGEN.</li> <li>• All SM4 mechanisms require an active regional cryptographic server.</li> </ul>
<b>C_Encrypt()</b>	
<b>C_EncryptUpdate()</b>	Multiple-part encryption is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
<b>C_EncryptFinal()</b>	Multiple-part encryption is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
<b>Decryption functions:</b>	

Table 28. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
<b>C_DecryptInit()</b>	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> <li>• CKM_DES_ECB</li> <li>• CKM_DES_CBC</li> <li>• CKM_DES_CBC_PAD</li> <li>• CKM_DES3_ECB</li> <li>• CKM_DES3_CBC</li> <li>• CKM_DES3_CBC_PAD</li> <li>• CKM_RSA_PKCS</li> <li>• CKM_RSA_X_509</li> <li>• CKM_AES_CBC</li> <li>• CKM_AES_ECB</li> <li>• CKM_AES_CBC_PAD</li> <li>• CKM_AES_CTS</li> <li>• CKM_AES_GCM (Limited to single part decryption only and for no more than 1048576 bytes of clear text.)</li> <li>• CKM_BLOWFISH_CBC</li> <li>• CKM_RC4</li> <li>• CKM_IBM_SM4_ECB</li> <li>• CKM_IBM_SM4_CBC</li> </ul> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>• A secure key may not be used for mechanisms CKM_AES_CTS, CKM_AES_GCM, or GCMIVGEN.</li> <li>• All SM4 mechanisms require an active regional cryptographic server.</li> </ul>
<b>C_Decrypt()</b>	
<b>C_DecryptUpdate()</b>	Multiple-part decryption is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
<b>C_DecryptFinal()</b>	Multiple-part decryption is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
<b>Message digesting functions:</b>	
<b>C_DigestInit()</b>	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> <li>• CKM_MD2</li> <li>• CKM_MD5</li> <li>• CKM_SHA_1</li> <li>• CKM_SHA224</li> <li>• CKM_SHA256</li> <li>• CKM_SHA384</li> <li>• CKM_SHA512</li> <li>• CKM_RIPEMD160</li> <li>• CKM_IBM_SM3</li> </ul>

Table 28. Standard PKCS #11 functions that ICSF supports (continued)

<b>Function</b>	<b>Usage notes</b>
<b>C_Digest()</b>	
<b>C_DigestUpdate()</b>	
<b>C_DigestFinal()</b>	
<b>Signing and message authentication coding (MACing) functions:</b>	



Table 28. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
<b>C_SignInit()</b>	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> <li>• CKM_RSA_X_509</li> <li>• CKM_RSA_PKCS</li> <li>• CKM_MD5_RSA_PKCS</li> <li>• CKM_SHA1_RSA_PKCS</li> <li>• CKM_SHA224_RSA_PKCS</li> <li>• CKM_SHA256_RSA_PKCS</li> <li>• CKM_SHA384_RSA_PKCS</li> <li>• CKM_SHA512_RSA_PKCS</li> <li>• CKM_DSA</li> <li>• CKM_DSA_SHA1</li> <li>• CKM_MD5_HMAC</li> <li>• CKM_SHA_1_HMAC</li> <li>• CKM_SHA224_HMAC</li> <li>• CKM_SHA256_HMAC</li> <li>• CKM_SHA384_HMAC</li> <li>• CKM_SHA512_HMAC</li> <li>• CKM_SSL3_MD5_MAC</li> <li>• CKM_SSL3_SHA1_MAC</li> <li>• CKM_MD2_RSA_PKCS</li> <li>• CKM_ECDSA</li> <li>• CKM_ECDSA_SHA1</li> <li>• CKM_RSA_PKCS_PSS</li> <li>• CKM_SHA1_RSA_PKCS_PSS</li> <li>• CKM_SHA224_RSA_PKCS_PSS</li> <li>• CKM_SHA256_RSA_PKCS_PSS</li> <li>• CKM_SHA384_RSA_PKCS_PSS</li> <li>• CKM_SHA512_RSA_PKCS_PSS</li> <li>• CKM_IBM_SM4_MAC_GENERAL</li> <li>• CKM_IBM_SM4_MAC</li> <li>• CKM_IBM_ISO2_SM4_MAC_GENERAL</li> <li>• CKM_IBM_ISO2_SM4_MAC</li> <li>• CKM_IBM_SM2</li> <li>• CKM_IBM_SM2_SM3</li> </ul> <p><b>Note:</b> All SMx mechanisms require an active regional cryptographic server.</p>
<b>C_Sign()</b>	
<b>C_SignUpdate()</b>	Multiple-part signature is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.

Table 28. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
<b>C_SignFinal()</b>	Multiple-part signature is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
<b>Functions for verifying signatures and message authentication codes (MACs):</b>	
<b>C_VerifyInit()</b>	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> <li>• CKM_RSA_X_509</li> <li>• CKM_RSA_PKCS</li> <li>• CKM_MD5_RSA_PKCS</li> <li>• CKM_SHA1_RSA_PKCS</li> <li>• CKM_SHA224_RSA_PKCS</li> <li>• CKM_SHA256_RSA_PKCS</li> <li>• CKM_SHA384_RSA_PKCS</li> <li>• CKM_SHA512_RSA_PKCS</li> <li>• CKM_DSA</li> <li>• CKM_DSA_SHA1</li> <li>• CKM_MD5_HMAC</li> <li>• CKM_SHA_1_HMAC</li> <li>• CKM_SHA224_HMAC</li> <li>• CKM_SHA256_HMAC</li> <li>• CKM_SHA384_HMAC</li> <li>• CKM_SHA512_HMAC</li> <li>• CKM_SSL3_MD5_MAC</li> <li>• CKM_SSL3_SHA1_MAC</li> <li>• CKM_MD2_RSA_PKCS</li> <li>• CKM_ECDSA</li> <li>• CKM_ECDSA_SHA1</li> <li>• CKM_RSA_PKCS_PSS</li> <li>• CKM_SHA1_RSA_PKCS_PSS</li> <li>• CKM_SHA224_RSA_PKCS_PSS</li> <li>• CKM_SHA256_RSA_PKCS_PSS</li> <li>• CKM_SHA384_RSA_PKCS_PSS</li> <li>• CKM_SHA512_RSA_PKCS_PSS</li> <li>• CKM_IBM_SM4_MAC_GENERAL</li> <li>• CKM_IBM_SM4_MAC</li> <li>• CKM_IBM_ISO2_SM4_MAC_GENERAL</li> <li>• CKM_IBM_ISO2_SM4_MAC</li> <li>• CKM_IBM_SM2</li> <li>• CKM_IBM_SM2_SM3</li> </ul> <p><b>Note:</b> All SMx mechanisms require an active regional cryptographic server.</p>

Table 28. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
C_Verify()	
C_VerifyUpdate()	Multiple-part verify is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
C_VerifyFinal()	Multiple-part verify is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
<b>Key management functions:</b>	
C_DeriveKey()	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> <li>• CKM_DH_PKCS_DERIVE</li> <li>• CKM_SSL3_MASTER_KEY_DERIVE</li> <li>• CKM_SSL3_MASTER_KEY_DERIVE_DH</li> <li>• CKM_SSL3_KEY_AND_MAC_DERIVE</li> <li>• CKM_TLS_MASTER_KEY_DERIVE</li> <li>• CKM_TLS_MASTER_KEY_DERIVE_DH</li> <li>• CKM_TLS_KEY_AND_MAC_DERIVE</li> <li>• CKM_TLS_PRF (It is the caller's responsibility to supply an ASCII value for the seed)</li> <li>• CKM_ECDH1_DERIVE</li> <li>• CKM_IBM_SM4_ECB_ENCRYPT_DATA</li> <li>• CKM_XOR_BASE_AND_DATA</li> </ul> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>• A secure or clear key may be specified as the base key for derivation mechanisms CKM_ECDH1_DERIVE and CKM_DH_PKCS.</li> <li>• Key derivation mechanism CKM_DH_PKCS derive clear keys only.</li> <li>• Key derivation mechanism CKM_ECDH1_DERIVE may derive clear or secure keys (see <a href="#">Table 5 on page 26</a> for hardware requirements to derive secure keys).</li> <li>• Key derivation mechanisms CKM_ECDH1_DERIVE and CKM_DH_PKCS derive clear keys only.</li> <li>• For key derivation mechanisms CKM_XOR_BASE_AND_DATA and CKM_IBM_SM4_ECB_ENCRYPT_DATA, a regional cryptographic server SM4 key must be specified as the base key.</li> <li>• Key derivation mechanisms CKM_XOR_BASE_AND_DATA and CKM_IBM_SM4_ECB_ENCRYPT_DATA derive secure SM4 keys only.</li> <li>• The derivation of SM4 keys requires key derivation mechanism CKM_XOR_BASE_AND_DATA or CKM_IBM_SM4_ECB_ENCRYPT_DATA.</li> <li>• All SM4 mechanisms require an active regional cryptographic server.</li> </ul>

Table 28. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
<b>C_GenerateKey()</b>	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> <li>• CKM_DES_KEY_GEN</li> <li>• CKM_DES2_KEY_GEN</li> <li>• CKM_DES3_KEY_GEN</li> <li>• CKM_PBE_SHA1_DES3_EDE_CBC</li> <li>• CKM_AES_KEY_GEN</li> <li>• CKM_DSA_PARAMETER_GEN</li> <li>• CKM_DH_PKCS_PARAMETER_GEN</li> <li>• CKM_BLOWFISH_KEY_GEN</li> <li>• CKM_RC4_KEY_GEN</li> <li>• CKM_GENERIC_SECRET_KEY_GEN</li> <li>• CKM_SSL3_PRE_MASTER_KEY_GEN</li> <li>• CKM_TLS_PRE_MASTER_KEY_GEN</li> <li>• CKM_IBM_SM4_KEY_GEN</li> </ul> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>• All SM4 mechanisms require an active regional cryptographic server.</li> <li>• CKM_IBM_SM4_KEY_GEN generates secure keys only.</li> </ul>
<b>C_GenerateKeyPair()</b>	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> <li>• CKM_RSA_PKCS_KEY_PAIR_GEN</li> <li>• CKM_DSA_KEY_PAIR_GEN</li> <li>• CKM_DH_PKCS_KEY_PAIR_GEN</li> <li>• CKM_EC_KEY_PAIR_GEN</li> <li>• CKM_IBM_SM2_KEY_PAIR_GEN</li> </ul> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>• All SMx mechanisms require an active regional cryptographic server.</li> <li>• SMx mechanisms generate secure keys only.</li> </ul>
<b>C_CreateObject</b>	<p><b>Note:</b> The creation of SMx keys requires an active regional cryptographic server.</p>
<b>C_CopyObject</b>	<p><b>Note:</b> The creation of SMx keys requires an active regional cryptographic server.</p>
<b>C_SetAttributeValue</b>	

Table 28. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
<b>C_WrapKey()</b>	<p>The following mechanisms are supported for wrapping secret keys:</p> <ul style="list-style-type: none"> <li>• CKM_RSA_PKCS</li> <li>• CKM_DES_CBC_PAD</li> <li>• CKM_DES3_CBC_PAD</li> <li>• CKM_AES_CBC_PAD</li> <li>• CKM_IBM_ATTRIBUTEBOUND_WRAP</li> <li>• CKM_IBM_SM4_ECB</li> <li>• CKM_IBM_SM2_ENCRYPT</li> </ul> <p>The following mechanisms are supported for wrapping private keys:</p> <ul style="list-style-type: none"> <li>• CKM_DES_CBC_PAD</li> <li>• CKM_DES3_CBC_PAD</li> <li>• CKM_AES_CBC_PAD</li> <li>• CKM_IBM_ATTRIBUTEBOUND_WRAP</li> </ul> <p>Clear keys may not be used to wrap secure keys and secure keys may not be used to wrap clear keys. One exception: Clear RSA public keys may be used to perform a non-attribute bound wrap of secure secret keys.</p> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>• All SMx mechanisms require an active regional cryptographic server.</li> <li>• Regional cryptographic server keys may only be wrapped by other regional cryptographic server keys.</li> </ul>

Table 28. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
<b>C_UnwrapKey()</b>	<p>The following mechanisms are supported for unwrapping secret keys:</p> <ul style="list-style-type: none"> <li>• CKM_RSA_PKCS</li> <li>• CKM_DES_CBC_PAD</li> <li>• CKM_DES3_CBC_PAD</li> <li>• CKM_AES_CBC_PAD</li> <li>• CKM_IBM_ATTRIBUTEBOUND_WRAP</li> <li>• CKM_IBM_SM4_ECB</li> <li>• CKM_IBM_SM2_ENCRYPT</li> </ul> <p>The following mechanisms are supported for unwrapping private keys:</p> <ul style="list-style-type: none"> <li>• CKM_DES_CBC_PAD</li> <li>• CKM_DES3_CBC_PAD</li> <li>• CKM_AES_CBC_PAD</li> <li>• CKM_IBM_ATTRIBUTEBOUND_WRAP</li> </ul> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>• All SMx mechanisms require an active regional cryptographic server.</li> <li>• The key security of the unwrapping key determines the key security of the unwrapped key.</li> </ul>
<b>Random number generation functions:</b>	
<b>C_SeedRandom()</b>	This function always returns the value CKR_RANDOM_SEED_NOT_SUPPORTED because the z/OS hardware random number generator is self-seeding.
<b>C_GenerateRandom()</b>	

## Non-standard functions supported

The following non-standard functions are also supported:

- CSN\_FindALLObjects()

Because they are non-standard, they do not appear in the PKCS #11 CK\_FUNCTION\_LIST structure returned by C\_GetFunctionList(). To invoke these functions, the caller must either locate the desired function in the main DLL using dlsym(), or link the application program with the main DLL's sidedeck.

### CSN\_FindALLObjects()

CSN\_FindALLObjects() is identical to C\_FindObjects(), except that it uses the ALL rule array keyword when invoking the ICSF CSFPTL callable service. This can result in CSN\_FindALLObjects() returning handles to private objects even if the caller has insufficient SAF authority to view such objects. CSN\_FindALLObjects() returns a private key handle (and C\_FindObjects does not) when the following conditions are all met:

1. The private object matches the search criteria.
2. No sensitive attributes were specified in the search criteria. The sensitive values for this service are:

- For a secret key object: CKA\_VALUE
  - For Diffie Hellman, DSA, and Elliptic Curve private key objects: CKA\_VALUE
  - For an RSA private key object: CKA\_PRIVATE\_EXPONENT, CKA\_PRIME\_1, CKA\_PRIME\_2, CKA\_EXPONENT\_1, CKA\_EXPONENT\_2, CKA\_COEFFICIENT
3. The caller has only Weak SO or SO R/W permission to the token.

Syntax of the CK\_RV CSN\_FindAllObjects() function:

```
CSN_FindAllObjects (
    CK_SESSION_HANDLE      hSession,
    CK_OBJECT_HANDLE_PTR  phObject,
    CK_ULONG               ulMaxObjectCount,
    CK_ULONG_PTR           pulObjectCount
);
```

For more information about CSFPTL processing with respect to the ALL rule array keyword, see [z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

## Non-standard mechanisms supported

### CKM\_IBM\_ATTRIBUTEBOUND\_WRAP

The CKM\_IBM\_ATTRIBUTEBOUND\_WRAP function is for wrapping and unwrapping private and secret keys in an IBM proprietary format, where the key's boolean usage attributes are included with the key material in the cryptogram. In addition to the format, the package is also signed, which means that the unwrapping party must have the matching verifying key.

CKM\_IBM\_ATTRIBUTEBOUND\_WRAP has the following restrictions:

- Only works with secure keys (CKA\_IBM\_SECURE=TRUE)
- All keys involved (target, wrapping/unwrapping, signature/verification) must be attribute bound keys (CKA\_IBM\_ATTRBOUND=TRUE), otherwise
  - For the target key on C\_WrapKey, CKR\_KEY\_NOT\_WRAPABLE is returned.
  - For the wrapping/unwrapping, signature/verification keys, CKR\_KEY\_FUNCTION\_NOT\_PERMITTED is returned.
- An attribute template, if specified, may not contain key usage attributes. If such a template is specified, CKR\_TEMPLATE\_INCONSISTENT is returned.
- On C\_WrapKey, the signing private key must be capable of signing (CKA\_SIGN=TRUE), otherwise CKR\_KEY\_FUNCTION\_NOT\_PERMITTED is returned.
- On C\_UnwrapKey, the verification public key must be capable of verifying (CKA\_VERIFY=TRUE), otherwise CKR\_KEY\_FUNCTION\_NOT\_PERMITTED is returned.

The CKM\_IBM\_ATTRIBUTEBOUND\_WRAP function takes a parameter, used to specify the signature or verification key handle.

Syntax of the CKM\_IBM\_ATTRIBUTEBOUND\_WRAP function:

```
typedef struct CKM_IBM_ATTRIBUTEBOUND_WRAP {
    CK_OBJECT_HANDLE hSignVerifyKey;
} CK_IBM_ATTRBOUND_WRAP_PARAMS;
```

**Note:** If attribute bound wrapping is used to import a key, the resulting key object may have certain usage attribute flags set FALSE even though they were set TRUE on the source key. This happens for the following key types:

- CKK\_GENERIC\_SECRET secret keys - CKA\_ENCRYPT, CKA\_DECRYPT, CKA\_WRAP, and CKA\_UNWRAP will be set FALSE
- CKK\_DSA, CKK\_EC, or CKK\_DH private keys - CKA\_SIGN\_RECOVER, CKA\_DECRYPT, and CKA\_UNWRAP will be set FALSE

This behavior, though inconsistent, does not cause a loss of function as these key types are not physically capable of performing the negated operations.

### CKM\_IBM\_SMx mechanisms

ICSF provides PKCS #11 extensions or mechanisms to be used with regional cryptographic servers. For additional information, see [“Regional cryptographic server key types and mechanisms supported”](#) on page 77.

## Enterprise PKCS #11 coprocessors

### Key algorithms/usages that are unsupported or disallowed by the Enterprise PKCS #11 coprocessors

The following table lists the key algorithms/usages that are not supported by the Enterprise PKCS #11 coprocessors or disallowed due to FIPS restrictions that are always enforced. The results of requesting an unsupported algorithm depend on what is being requested. All these results assume the system is properly configured to use secure PKCS #11. Improper configuration would result in different errors:

1. Key generation or creation – Explicitly requesting the generation or creation of an unsupported/disallowed secure key type results in CKR\_TEMPLATE\_INCONSISTENT being returned.
2. Key derivation – Explicitly requesting the derivation of a secure key using a clear base key results CKR\_TEMPLATE\_INCONSISTENT being returned. Attempting key derivation using a secure base key results in CKR\_IBM\_CLEAR\_KEY\_REQ being returned.
3. Standard unwrap key – The target key always has the security of the unwrapping key. Specifying the CKA\_IBM\_SECURE attribute in the unwrap template results in CKR\_ATTRIBUTE\_READ\_ONLY being returned. Requesting the unwrapping of an unsupported/disallowed key type using a secure unwrapping key results in CKR\_IBM\_CLEAR\_KEY\_REQ being returned.
4. Otherwise, requesting an unsupported/disallowed algorithm using a secure key results in CKR\_IBM\_CLEAR\_KEY\_REQ being returned.

Algorithm	PKCS #11 Mechanisms or key types	Comments
<b>MD2</b>	CKM_MD2_RSA_PKCS	Secure private key use for signing disallowed
<b>MD5</b>	CKM_MD5_RSA_PKCS, CKM_MD5_HMAC	Secure private or secret key use for signing disallowed
<b>SSL3</b>	CKM_SSL3_MD5_MAC, CKM_SSL3_SHA1_MAC, CKM_SSL3_MASTER_KEY_DERIVE, CKM_SSL3_MASTER_KEY_DERIVE_DH, CKM_SSL3_KEY_AND_MAC_DERIVE	
<b>TLS</b>	CKM_TLS_MASTER_KEY_DERIVE, CKM_TLS_MASTER_KEY_DERIVE_DH, CKM_TLS_KEY_AND_MAC_DERIVE	
<b>Diffie Hellman</b>	CKK_DH keys	Prime size less than 1024 bits
<b>DSA</b>	CKK_DSA keys	Combinations other than the following are not supported: <ul style="list-style-type: none"> <li>• Prime size = 1024 bits, subprime size = 160 bits</li> <li>• Prime size = 2048 bits, subprime size = 224 bits or 256 bits</li> </ul>



Algorithm	PKCS #11 Mechanisms or key types	Comments
<b>Single DES</b>	CKK_DES keys	
<b>Triple DES</b>	CKK_DES2 keys	
<b>Blowfish</b>	CKK_BLOWFISH keys	
<b>RC4</b>	CKK_RC4 keys	
<b>RSA</b>	CKK_RSA CKM_RSA_PKCS_KEY_PAIR_GEN  CKM_RSA_X_509	Key sizes less than 1024 bits  Key sizes that are less than 1024 bits or not a multiple of 256 bits or public key exponents less than 0x010001  Secure private key use for signing/decryption disallowed
<b>HMAC</b>	CKK_GENERIC_SECRET  CKM_SHA_1_HMAC, CKM_SHA224_HMAC, CKM_SHA256_HMAC, CKM_SHA384_HMAC, CKM_SHA512_HMAC	Key sizes less than 10 bytes  Base key sizes less than ½ the output size are not supported.
<b>AES GCM</b>	CKM_AES_GCM	

## PKCS #11 Coprocessor Access Control Points

The following table lists the Access Control Points that are available on the Enterprise PKCS #11 coprocessors and the PKCS #11 mechanisms or functions that would be disabled for secure keys if the control point is deactivated. A new or a zeroized Enterprise PKCS #11 coprocessor (or domain) comes with an initial set of Access Control Points (ACPs) that are enabled by default. All other ACPs, representing potential future support, are left disabled. When a firmware upgrade is applied to an existing Enterprise PKCS #11 coprocessor, the upgrade might introduce new ACPs. The firmware upgrade does not retroactively enable these ACPs, so they are disabled by default. These ACPs must be enabled with the TKE (or subsequent zeroize) to use the new support they govern.

See the Enabling Access Control Points for PKCS #11 coprocessor firmware section in the Migration topic of the *z/OS Cryptographic Services ICSF System Programmer's Guide* for the list of default ACPs and those ACPs that need to be enabled with the TKE for PKCS #11 coprocessor firmware upgrades.

The following table lists the Access Control Points that are available on the Enterprise PKCS #11 coprocessors and the PKCS #11 mechanisms or functions that would be disabled for secure keys if the control point is deactivated.

Access Control Point name or group	Mechanism/Function requiring enablement	Number
<b>Control Point Management</b>		
<b>Allow addition (activation) of Control Points</b>	Not applicable	0
<b>Allow removal (deactivation) of Control Points</b>	Not applicable	1
<b>Cryptographic Operations</b>		
<b>Sign with private keys</b>	Sign using CKK_RSA, CKK_DSA, of CKK_ECDSA keys.	2

Table 30. PKCS #11 Access Control Points (continued)

Access Control Point name or group	Mechanism/Function requiring enablement	Number
<b>Sign with HMAC or CMAC</b>	Sign using CKM_SHA_1_HMAC, CKM_SHA224_HMAC, CKM_SHA256_HMAC, CKM_SHA384_HMAC, or CKM_SHA512_HMAC.	3
<b>Verify with HMAC or CMAC</b>	Verify using CKM_SHA_1_HMAC, CKM_SHA224_HMAC, CKM_SHA256_HMAC, CKM_SHA384_HMAC, or CKM_SHA512_HMAC.	4
<b>Encrypt with symmetric keys</b>	Encrypt with CKK_DES3 or CKK_AES keys.  Create Object or Copy Object where source is a clear key.	5
<b>Decrypt with private keys</b>	Decrypt with CKK_RSA keys.	6
<b>Decrypt with symmetric keys</b>	Decrypt with CKK_DES3 or CKK_AES keys.	7
<b>Key export with public keys</b>	Wrap Key using a CKK_RSA wrapping key.	8
<b>Key export with symmetric keys</b>	Wrap Key using a CKK_DES3 or CKK_AES wrapping key.	9
<b>Key import with private keys</b>	Unwrap Key using a CKK_RSA unwrapping key.	10
<b>Key import with symmetric keys</b>	Unwrap Key using a CKK_DES3 or CKK_AES unwrapping key.  Create Object or Copy Object where source is a clear key.	11
<b>Generate asymmetric key pairs</b>	Generate Key Pair for CKK_RSA, CKK_DSA, or CKK_ECDSA keys.	12
<b>Generate symmetric keys</b>	Generate key for CKK_DES2 or CKK_AES keys.	13
<b>Allow key derivation</b>	Derive key using a CKK_DH key.	47
<b>Allow protected keys</b>	Encrypt/Decrypt using CKK_AES or CKK_DES keys. Sign using CKK_EC private keys.	64
<b>Cryptographic Algorithms</b>		
<b>RSA private-key use</b>	Generate Key Pair for CKK_RSA Sign or Decrypt using a CKK_RSA key	30
<b>DSA private-key use</b>	Generate Key Pair for CKK_DSA Sign using a CKK_DSA key	31
<b>EC private-key use</b>	Generate Key Pair for CKK_EC Sign or Derive Key using a CKK_EC key	32

Table 30. PKCS #11 Access Control Points (continued)

Access Control Point name or group	Mechanism/Function requiring enablement	Number
DH private-key use	Generate Key Pair for CKK_DH	46
Brainpool (E.U.) EC curves	Sign or Verify using the Brainpool curves	33
NIST/SECG EC curves	Sign or Verify using the NIST EC curves	34
Allow non-BSI algorithms (as of 2009)	Not applicable	21
Allow non-FIPS-approved algorithms (as of 2011)	Not applicable	35
Allow non-BSI algorithms (as of 2011)	Not applicable	36
Allow non-BSI algorithms (as of 2017)	Not applicable	61
Allow using 25519, c41417, and c448 curves	Sign or verify using 25519 and c448 curves.	55
Dilithium key use	Generate Dilithium key pair (CSFPGKP).	65
<b>Key Size</b>		
Allow 80 to 111-bit algorithms	Any use of CKK_GENERIC_SECRET keys smaller than 112 bits, or 160 or 192 bit CKK_ECDSA keys <b>If in a BSI mode:</b> Any use of CKK_DSA keys, or CKK_RSA keys smaller than 2432 bits <b>If not in a BSI mode:</b> Any use of CKK_DSA, or CKK_RSA keys smaller than 2048 bits	24
Allow 112 to 127-bit algorithms	Any use of 2048 bit CKK_DSA keys, CKK_GENERIC_SECRET keys larger than 111 bits but less than 128 bits, 224 bit CKK_ECDSA keys, or CKK_DES3 keys <b>If in a BSI mode:</b> Any use of CKK_RSA keys larger that 2431 bits but less than 3248 bits <b>If not in a BSI mode:</b> Any use of CKK_RSA keys larger that 2047 bits but less than 3072 bits	25
Allow 128 to 191-bit algorithms	Any use of CKK_GENERIC_SECRET keys larger than 127 bits but less than 192 bits, 128 bit CKK_AES keys, or 256 bit CKK_ECDSA keys <b>If in a BSI mode:</b> Any use of CKK_RSA keys larger that 3247 bits <b>If not in a BSI mode:</b> Any use of CKK_RSA keys larger that 3071 bits	26
Allow 192 to 255-bit algorithms	Any use of CKK_GENERIC_SECRET keys larger than 191 bits, 192 bit CKK_AES keys or 384 bit CKK_ECDSA keys.	27
Allow 256-bit algorithms	Any coprocessor use other than random number generation.	28

Table 30. PKCS #11 Access Control Points (continued)

<b>Access Control Point name or group</b>	<b>Mechanism/Function requiring enablement</b>	<b>Number</b>
<b>Allow RSA public exponents below 0x10001</b>	Generate Key or Generate Key Pair for CKK_RSA where the exponent is 3.	29
<i>Miscellaneous</i>		
<b>Allow backend to save semi-retained keys</b>	Not applicable	14
<b>Allow keywrap without attribute-binding</b>	Wrap Key or Unwrap Key using CKM_RSA_PKCS, CKM_AES_CBC_PAD, or CKM_DES3_CBC_PAD Create Object or Copy Object where source is a clear key.	16
<b>Allow changes to key objects (usage flags only)</b>	Set Attribute Value or Copy Object where the key usage flags are modified	17
<b>Allow mixing external seed to RNG</b>	Not applicable	18
<b>Allow non-administrators to mark key objects TRUSTED</b>	Set Attribute Value where CKA_TRUSTED is set TRUE	37
<b>Do not double-check sign/decrypt operations</b>	Not applicable	38
<b>Allow dual-function keys - key wrapping and data encryption</b>	Generate Key or Generate Key Pair where CKA_WRAP / CKA_UNWRAP and CKA_ENCRYPT / CKA_DECRYPT combinations are requested (or defaulted)  Wrap Key, Unwrap Key, Encrypt or Decrypt with a previously created key containing the previous combination.  Create Object or Copy Object where source is a clear key.	39
<b>Allow dual-function keys - digital signature and data encryption</b>	Create Object, Generate Key or Generate Key Pair where CKA_SIGN / CKA_VERIFY and CKA_ENCRYPT / CKA_DECRYPT combinations are requested (or defaulted)  Sign, Verify, Encrypt or Decrypt with a previously created key containing the previous combination	40
<b>Allow dual-function keys - key wrapping and digital signature</b>	Create Object, Generate Key or Generate Key Pair where CKA_SIGN / CKA_VERIFY and CKA_WRAP / CKA_UNWRAP combinations are requested (or defaulted)  Sign, Verify, Wrap Key or Unwrap Key with a previously created key containing the previous combination	41
<b>Allow non-administrators to mark public key objects ATTRBOUND</b>	Create Object where CKA_IBM_ATTRBOUND is set TRUE	42
<b>Allow clear passphrases for password-based-encryption</b>	Generate Key using CKM_PBE_SHA1_DES3_EDE_CBC	43
<b>Allow wrapping of stronger keys by weaker keys</b>	Wrap Key where the to-be-wrapped key is stronger than the wrapping key.	44

Table 30. PKCS #11 Access Control Points (continued)

Access Control Point name or group	Mechanism/Function requiring enablement	Number
<b>Allow clear public keys as non-attribute bound wrapping keys</b>	Wrap Key where the wrapping key is an CKK_RSA clear public key and the to-be-wrapped key is a secure CKK_DES3, CKK_AES, or CKK_GENERIC_SECRET key.	45

## Standard compliance modes

Enterprise PKCS #11 coprocessors are designed to always operate in a FIPS compliant fashion. The optional compliance modes that a given domain can be in correspond to FIPS modes based on FIPS 140-2 requirements as of 2009 and 2011, and BSI modes based on the BSI HSM protection profile and German Bundesnetzagentur algorithms as of 2009, 2011, and 2017.

These compliance modes are not mutually exclusive and are set ON by disabling certain access control points:

1. **FIPS 2009** – Card adheres to FIPS restrictions that went into effect in 2009 – equivalent to ICSF’s FIPS140 mode. This is the default mode. A domain cannot be set to be less restrictive than this mode. This mode has the following policy/restrictions:
  - a. Algorithms and keys below 80-bit of strength are not permitted.
  - b. RSA private-keys may not be use without padding.
  - c. Newly generated asymmetric keys always undergo selftests.
  - d. The minimum keysize on HMAC is 1/2 the algorithm's output size.
  - e. Only FIPS-approved algorithms (as of 2009) are present.
2. **FIPS 2011** – Card adheres to FIPS restrictions that went into effect in 2011. (More restrictive than FIPS 2009.) The following access control points would need to be disabled:
  - a. Allow 80 to 111-bit algorithms.
  - b. Allow non-FIPS-approved algorithms (as of 2011).
  - c. Allow RSA public exponents below 0x10001.
3. **BSI 2009** – Card adheres to BSI restrictions that went into effect in 2009. The following access control points would need to be disabled:
  - a. Allow keywrap without attribute-binding.
  - b. Allow non-BSI-approved algorithms (as of 2009).
4. **BSI 2011** – Card adheres to BSI restrictions that went into effect in 2011. (More restrictive than BSI 2009.) The following access control points would need to be disabled:
  - a. Allow keywrap without attribute-binding.
  - b. Allow 80 to 111-bit algorithms.
  - c. Allow non-BSI-approved algorithms (as of 2011).
5. **BSI/CC 2017** – Card adheres to the CP settings evaluated for Common Criteria Certification by German Bundesnetzagentur. The following access control points would need to be disabled:
  - a. Allow key wrap without attribute-bindings.
  - b. Allow mixing external seed to RNG.
  - c. Allow RSA public exponents below 0x10001.
  - d. Allow non-administrators to mark key objects TRUSTED.
  - e. Allow dual-function keys - key wrapping and data encryption.
  - f. Allow dual-function keys - digital signature and data encryption.
  - g. Allow dual-function keys - key wrapping and digital signature.

- h. Allow non-administrators to mark public key objects ATTRBOUND.
- i. Allow clear passphrases for password-based-encryption.
- j. Allow wrapping of stronger keys by weaker keys using BSI equivalent key lengths documented in BSI TR-02102-1.
- k. Allow clear public keys as non-attribute bound wrapping keys.

Whenever a secure key is created, the current compliance mode of the Enterprise PKCS #11 coprocessor is recorded inside the secure key. If the compliance mode of the coprocessor is subsequently changed, all previously created secure keys become unusable until their compliance modes are updated. See [“Steps for running the pre-compiled version of testpkcs11” on page 73](#) for information on how to modify the compliance mode of a secure key using a sample program distributed by IBM. The compliance mode of a key may also be updated by using the PKCS #11 Token Browser ISPF panels. For more information on these panels, see [z/OS Cryptographic Services ICSF Administrator's Guide](#).

## Function return codes

In general, the PKCS #11 function return codes are defined in the PKCS #11 specification. However, the following function return codes have a meaning specific to z/OS:

### **CKR\_TOKEN\_NOT\_PRESENT**

ICSF is not running or the TKDS is not properly configured. Note that this return code has no relationship to the slot flag CKF\_TOKEN\_PRESENT.

### **CKR\_TOKEN\_NOT\_RECOGNIZED**

The caller is not authorized to perform the action requested.

### **CKR\_MECHANISM\_INVALID**

The specified mechanism is either unknown or not supported by the current cryptographic hardware configuration.

### **CKR\_DEVICE\_REMOVED**

The token no longer exists. When this error is detected, the token flags are cleared indicating that the token is no longer initialized. It can be re-initialized as a new token, if desired.

Other ICSF-related errors are returned as vendor-defined error codes (CKR\_VENDOR\_DEFINED). The ICSF return and reason codes are combined into the single return code as follows:

```
#define CKR_IBM_ICSF_ERROR 0xC0000000 /* High order byte mask indicating ICSF error */
#define CKR_IBM_ICSF_ERROR_RET 0x00FF0000 /* Second byte is the return code */
#define CKR_IBM_ICSF_ERROR_RSN 0x0000FFFF /* low order half word is reason code */
```

This mapping is also used to store the ICSF return and reason code values in the CK\_SESSION\_INFO ulDeviceError field.

The following constants are defined for select ICSF return reason codes:

```
/* ICSF not configured for FIPS mode OR system does not support FIPS mode */
#define CKR_IBM_ICSF_NOT_FIPS_MODE 0xC0080BFD

/* Algorithm or key size is not valid in FIPS mode */
#define CKR_IBM_ICSF_NOT_VALID_FIPS 0xC0080BFE

/* FIPS known answer tests failed */
#define CKR_IBM_ICSF_FIPS_KAT_FAILED 0xC00C8D3C

/* Service or algorithm not available on current system */
#define CKR_IBM_ICSF_SERV_NOTAVAIL 0xC00C0008
```

```
/* A clear key is required for this operation, but a secure key was supplied */
#define CKR_IBM_CLEAR_KEY_REQ 0xC0080C81

/* Clear key creation denied by policy */
#define CKR_IBM_DENIED_BY_POLICY 0xC0083E88
```

```

/* Key object in more restrictive compliance mode than
current setting of the Enterprise PKCS #11 coprocessors */
#define CKR_IBM_KEY_MODE_ERROR 0xC00C3200

```

## Troubleshooting PKCS #11 applications

**Note:** The information and techniques described in this topic are for use primarily by IBM service personnel in determining the cause of a problem with the ICSF PKCS #11 C API.

You can capture trace data using environment variables. To do this, the trace environment variables CSN\_PKCS11\_TRACE and CSN\_PKCS11\_TRACE\_FILE must be exported prior to the application's first call to any of the PKCS #11 functions.

Table 31. Environment variables for capturing trace data

Environment variable	Usage	Valid values
<b>CSN_PKCS11_TRACE</b>	Specifies the level of tracing to be performed.	<p>An integer value, 1-7. The higher the value, the greater the number of conditions traced. Each level includes the conditions of the levels below it:</p> <p><b>7</b> Debug information</p> <p><b>6</b> Informational conditions</p> <p><b>5</b> Normal but significant conditions</p> <p><b>4</b> Warning conditions</p> <p><b>3</b> Error conditions</p> <p><b>2</b> Critical conditions</p> <p><b>1</b> Immediate action required</p> <p>Any other value causes tracing to be inactive (the default).</p>
<b>CSN_PKCS11_TRACE_FILE</b>	<p>Specifies the name of the trace file. Defaults to /tmp/csnpkcs11.%.trc.</p> <p>The current process identifier is included as part of the trace file name when the name contains a percent sign (%). For example, if CSN_PKCS11_TRACE_FILE is set to:</p> <pre style="background-color: #f0f0f0; padding: 2px;">/tmp/csnpkcs11.%.trc</pre> <p>and the current process identifier is 247, the trace file name is</p> <pre style="background-color: #f0f0f0; padding: 2px;">/tmp/csnpkcs11.247.trc</pre>	<p>Must be set to the full path name of an HFS file in a directory for which the executing application has write permission. The maximum length for the path name is 255 bytes. Values longer than 255 bytes are truncated.</p>

You can also use the utility program, testpkcs11, for troubleshooting. For information about running testpkcs11, see [“Running the pre-compiled version of testpkcs11” on page 73](#).



---

## Chapter 3. Sample PKCS #11 C programs

IBM provides sample PKCS #11 C programs. The source code for the sample programs is provided in `/usr/lpp/pkcs11/samples/`. See [“Building sample PKCS #11 applications from source code”](#) on page 74 for instructions on how to build and run a sample program.

- **Sample testpkcs11:** This program is passed the name of a PKCS #11 token, and performs the following tasks:

1. Creates a token that has the name passed
2. Generates an RSA key-pair
3. Encrypts some test data using the public part of the key-pair
4. Decrypts the data using the private part of the key-pair
5. Deletes the key-pair and the token

You can use this program in several ways:

- As a utility program to test the system configuration for PKCS #11 and troubleshoot problems.
- As a sample application to learn how to build and run a PKCS #11 application.

IBM provides a pre-compiled version of this program installed in `/usr/lpp/pkcs11/bin`. For the source code for this program, see [Appendix B, “Source code for the testpkcs11 sample program,”](#) on page 93.

- **Sample updatecomp:** This sample program uses the C API for updating the FIPS/BSI compliance mode of all keys in a token. With this program, users can update all the keys in a token to the current compliance mode of the Enterprise PKCS # 11 coprocessor or to some other mode. Users will find this program a useful aid when changing the FIPS/BSI compliance mode of an Enterprise PKCS #11 coprocessor where increasing the coprocessor compliance mode makes all existing Secure PKCS #11 keys unusable. This program creates a backup copy of each key (under the same token) before updating its compliance mode. The label of the copy will indicate the sequence number of the original key. The copied keys act as a backup, allowing one to revert to the previous compliance mode, if necessary. Users can examine the backup and updated keys via the PKCS #11 Token Browser ISPF panels. See [“Standard compliance modes”](#) on page 69 for more information.

- Usage: **updatecomp** { `-t token-name [-c comp-mode]` | `-h` }
  - `-t token-name` = The name of the token to be updated.
  - `-c comp-mode` = The numeric value (0-15) to use as the new compliance mode. Optional, ICSF will reset to the current mode of the coprocessor if not specified.
  - `-h` = Displays this help.
- IBM recommends that you use **updatecomp** only to reset your keys to match the compliance mode of the coprocessor. This can be accomplished by specifying `-c 0` or letting the `-c` switch default to 0. Use caution when specifying a non-zero compliance mode via the `-c` switch. Specifying a value that does not match the coprocessor could disable use of the keys permanently.

---

### Running the pre-compiled version of testpkcs11

If you are testing the system configuration for PKCS #11, or troubleshooting problems with the configuration, you can run the pre-compiled version of testpkcs11.

#### Steps for running the pre-compiled version of testpkcs11

##### About this task

**Before you begin:** You need to know how to use z/OS UNIX shells.

Perform the following steps to run the pre-compiled version of testpkcs11.

### Procedure

1. Change to the PKCS #11 bin directory by entering the following command:

```
cd /usr/lpp/pkcs11/bin
```

2. Choose a temporary token name to use. If you need to review the rules for token names, see [“Tokens” on page 1](#).

3. Run testpkcs11, passing it your token name on the -t option. For example, to use a temporary token name of my.temp.token, enter the following command:

```
./testpkcs11 -t my.temp.token
```

### Results

If z/OS PKCS #11 has been set up properly and the you have sufficient authority to the token label specified, you should see the following output:

```
Getting the PKCS11 function list...
Initializing the PKCS11 environment...
Creating the temporary token...
Opening a session...
Generating keys. This may take a while...
Enciphering data...
Deciphering data...
Destroying keys...
Closing the session...
Deleting the temporary token...
Test completed successfully!
```

If you see different messages, there is an error in either your PKCS #11 set up or in the token label that you specified.

The most common user error is specifying token label that is unacceptable to ICSF or already in use. In that case the following is displayed:

```
Getting the PKCS11 function list...
Initializing the PKCS11 environment...
Creating the temporary token...
C_InitToken #1 returned 7 (0x07) CKR_ARGUMENTS_BAD
Make sure your the token name you specified meets ICSF rules:
Contains only alphanumeric characters, nationals (@#)$, and periods.
The first character cannot be a numeric or a period.
```

If you see other error messages, there is probably an error in the setup for the PKCS #11 environment. Determine the error represented by the PKCS #11 error code returned. For information about error codes, see [“Function return codes” on page 70](#).

To display the help text for the testpkcs11 program, run the program with the -h option:

```
cd /usr/lpp/pkcs11/bin
./testpkcs11 -h
```

## Building sample PKCS #11 applications from source code

If you are learning how to build and run a PKCS #11 application, you can use the source code for testpkcs11 to build and run a sample application.

**Before you begin:** You need to know in which directory the PKCS #11 header file is located. By default it is located in the standard include subdirectory under /usr. If your standard include subdirectory it in a

different location, you will need to modify the Makefile in step 2. You also need to know how to use z/OS UNIX shells.

**Makefiles for sample programs:** Makefiles for 31, 64, and 31 XPLINK addressing for the sample programs are provided in `/usr/lpp/pkcs11/samples`. The naming convention of the makefiles are:

```
Makefile.samplepgmname,  
Makefile64.samplepgmname, and  
Makefile3X.samplepgmname
```

where *samplepgmname* is the name of the sample C program.

**Example:** Perform the following steps to build the sample application, `testpkcs11`. (Other samples would be built in a similar fashion.) Issue the commands from the z/OS UNIX command shell.

1. Copy the `testpkcs11.c` program and appropriate Makefile to the current directory. For example, for 64-bit addressing enter the following commands:

```
cp /usr/lpp/pkcs11/samples/testpkcs11.c testpkcs11.c  
cp /usr/lpp/pkcs11/samples/Makefile64.testpkcs11 Makefile
```

- 
2. If the standard include subdirectory is not located under `/usr`, edit the Makefile copied in step 1 and change the `PKCS11_INSTALL_DIR` variable as required.

- 
3. Enter the following command to compile and link and produce the executable, `testpkcs11`:

```
make
```

- 
4. Update your C/C++ environment variable `_CEE_RUNOPTS` to include `XPLINK(ON)` if it does not already include it. For example, execute the following command from a UNIX shell:

```
export _CEE_RUNOPTS=$_CEE_RUNOPTS' XPLINK(ON) '
```

---

When you are done, you have built the `testpkcs1164` application and can run it in your directory. For example, to run `testpkcs1164` in your directory and display its help text, enter the following command:

```
./testpkcs1164 -h
```

To run a test, choose a temporary token name and enter it with the `-t` option. If you need to review the rules for token names, see [“Tokens” on page 1](#). For example, to use a temporary token name of `my.temp.token`, enter the following command:

```
./testpkcs1164 -t my.temp.token
```

For the output that should appear, see [“Steps for running the pre-compiled version of testpkcs11” on page 73](#).



---

## Chapter 4. Regional cryptographic servers

ICSF provides PKCS #11 extensions or mechanisms to be used with regional cryptographic servers.

### Regional cryptographic server key types and mechanisms supported

---

For generation 3 regional cryptographic servers and later, the following international standard mechanisms and key types are supported:

- AES
- RSA
- TDES
- ECC

#### For all the key types (CKK\_\*)

To use the key object with a regional cryptographic server, the attribute CKA\_IBM\_REGIONAL (CK\_BBOOL) is required, must be TRUE, and is bound to the key, just as it is required for CKK\_SM2 and CKK\_SM4 key objects.

#### For CKK\_AES

The following mechanisms are supported:

- CKM\_AES\_KEY\_GEN
- CKM\_AES\_ECB
- CKM\_AES\_CBC
- CKM\_AES\_CBC\_PAD

#### For CKK\_RSA

The following mechanisms are supported:

- CKM\_RSA\_PKCS\_KEY\_PAIR\_GEN
- CKM\_RSA\_PKCS
- CKM\_RSA\_X\_509
- CKM\_SHA1\_RSA\_PKCS
- CKM\_SHA224\_RSA\_PKCS
- CKM\_SHA256\_RSA\_PKCS
- CKM\_SHA384\_RSA\_PKCS
- CKM\_SHA512\_RSA\_PKCS
- CKM\_RSA\_PKCS\_PSS
- CKM\_SHA1\_RSA\_PKCS\_PSS
- CKM\_SHA224\_RSA\_PKCS\_PSS
- CKM\_SHA256\_RSA\_PKCS\_PSS
- CKM\_SHA384\_RSA\_PKCS\_PSS
- CKM\_SHA512\_RSA\_PKCS\_PSS

#### For CKK\_DES3

The following mechanisms are supported:

- CKM\_DES3\_KEY\_GEN
- CKM\_DES3\_ECB
- CKM\_DES3\_CBC
- CKM\_DES3\_CBC\_PAD

### For CKK\_EC

The following mechanisms are supported:

- CKM\_EC\_KEY\_PAIR\_GEN
- CKM\_ECDSA

The CKK\_IBM\_SM2 and CKK\_IBM\_SM4 key types are defined for the type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key objects. All SMx mechanisms require an active regional cryptographic server. The SMx specific vendor-defined mechanisms are:

- [“CKM\\_IBM\\_SM2” on page 79](#)
- [“CKM\\_IBM\\_SM2\\_ENCRYPT” on page 79](#)
- [“CKM\\_IBM\\_SM2\\_KEY\\_PAIR\\_GEN” on page 80](#)
- [“CKM\\_IBM\\_SM2\\_SM3” on page 80](#)
- [“CKM\\_IBM\\_SM3” on page 81](#)
- [“CKM\\_IBM\\_SM4\\_CBC” on page 81](#)
- [“CKM\\_IBM\\_SM4\\_ECB” on page 81](#)
- [“CKM\\_IBM\\_SM4\\_ECB\\_ENCRYPT\\_DATA” on page 82](#)
- [“CKM\\_IBM\\_SM4\\_ISO2\\_MAC” on page 82](#)
- [“CKM\\_IBM\\_SM4\\_ISO2\\_MAC\\_GENERAL” on page 83](#)
- [“CKM\\_IBM\\_SM4\\_KEY\\_GEN” on page 83](#)
- [“CKM\\_IBM\\_SM4\\_MAC” on page 83](#)
- [“CKM\\_IBM\\_SM4\\_MAC\\_GENERAL” on page 84](#)

The standard mechanism CKM\_XOR\_BASE\_AND\_DATA is also supported for SM4 key derivation. For additional details, see [“CKM\\_XOR\\_BASE\\_AND\\_DATA” on page 84](#).

Table 32 on page 78 shows the regional cryptographic server mechanisms that are supported and their functions:

Mechanism	Encrypt and decrypt	Sign and verify	SR and VR	Digest	Generate key or key pair	Wrap and unwrap	Derive
<b>CKM_IBM_SM2_KEY_PAIR_GEN</b>	N/A	N/A	N/A	N/A	X	N/A	N/A
<b>CKM_IBM_SM2</b>	N/A	X	N/A	N/A	N/A	N/A	N/A
<b>CKM_IBM_SM2_SM3</b>	N/A	X	N/A	X	N/A	N/A	N/A
<b>CKM_IBM_SM2_ENCRYPT</b>	N/A	N/A	N/A	N/A	N/A	X	N/A
<b>CKM_IBM_SM3</b>	N/A	N/A	N/A	X	N/A	N/A	N/A
<b>CKM_IBM_SM4_KEY_GEN</b>	N/A	N/A	N/A	N/A	X	N/A	N/A
<b>CKM_IBM_SM4_ECB</b>	X	N/A	N/A	N/A	N/A	X	N/A
<b>CKM_IBM_SM4_CBC</b>	X	N/A	N/A	N/A	N/A	N/A	N/A
<b>CKM_IBM_SM4_MAC_GENERAL</b>	N/A	X	N/A	N/A	N/A	N/A	N/A
<b>CKM_IBM_SM4_MAC</b>	N/A	X	N/A	N/A	N/A	N/A	N/A

Table 32. Regional cryptographic server mechanisms and functions (continued)

Mechanism	Encrypt and decrypt	Sign and verify	SR and VR	Digest	Generate key or key pair	Wrap and unwrap	Derive
CKM_IBM_ISO2_SM4_MAC_GENERAL	N/A	X	N/A	N/A	N/A	N/A	N/A
CKM_IBM_ISO2_SM4_MAC	N/A	X	N/A	N/A	N/A	N/A	N/A
CKM_IBM_SM4_ECB_ENCRYPT_DATA	N/A	N/A	N/A	N/A	N/A	N/A	X
CKM_XOR_BASE_AND_DATA	N/A	N/A	N/A	N/A	N/A	N/A	X

## CKM\_IBM\_SM2

CKM\_IBM\_SM2 is a mechanism for single-part signatures and verification for SM2. CKM\_IBM\_SM2 does not have a parameter, but generates 64-byte signatures.

CKM\_IBM\_SM2 corresponds only to the part of SM2 that processes the final hash value, which is the 256-bit SM3 hash of  $Z_A \parallel \text{Message}$ ; it does not compute the hash value.

The application must produce the 256-bit SM3 hash to be used as the input data for this mechanism. To do so, complete the following the steps:

- Set  $ID_A$  equal to the user ID of the signer.
- Set  $ENTL_A$  equal to the two-byte binary bit length of  $ID_A$ .
- Set  $Z_A = \text{HASH}_{SM3}(\text{ENTL}_A \parallel ID_A \parallel a \parallel b \parallel X_G \parallel Y_G \parallel X_A \parallel Y_A)$ , where  $a$ ,  $b$ ,  $X_G$ , and  $Y_G$  are the curve parameters and  $X_A$  and  $Y_A$  are the signer's public key point coordinates.

Calculate  $\text{HASH}_{SM3}(Z_A \parallel \text{Message})$ . This is the input data for CKM\_IBM\_SM2.

Constraints on key types and the length of data are summarized in Table 33 on page 79:

Function	Key type	Data length	Signature length
C_Sign	SM2 private key	32 bytes	64 bytes
C_Verify	SM2 public key	32 bytes, 64 bytes	N/A

For CKM\_IBM\_SM2, the *ulMinKeySize* and *ulMaxKeySize* fields of the CK\_MECHANISM\_INFO structure are not used.

## CKM\_IBM\_SM2\_ENCRYPT

CKM\_IBM\_SM2\_ENCRYPT is a public key encryption mechanism that utilizes SM2 for key wrapping and key unwrapping. CKM\_IBM\_SM2\_ENCRYPT does not have a parameter, but can wrap and unwrap SM4 keys.

For key wrapping, CKM\_IBM\_SM2\_ENCRYPT encrypts the value of the CKA\_VALUE attribute of the key that is to be wrapped. It does not wrap the key type, key length, or any other information about the key. Your application must supply this information separately.

Let  $K$  equal the value of the SM4 key to be wrapped. The mechanism generates an ephemeral SM2 key-pair  $(x_1, y_1)$ . This key-pair is then used to perform an EC-DH key agreement with the recipient's public

key. The resulting shared secret EC point  $(x_2, y_2)$  is then used as input to the following key derivation function (KDF):

$$t = \text{HASH}_{\text{SM3}}(x_2 || y_2 || 0x00000001)$$

The left-most 16 bytes of  $t$  are XORed with  $K$  to produce the enciphered key. Finally, an authentication tag is produced as follows:

$$\text{tag} = \text{HASH}_{\text{SM3}}(x_2 || K || y_2)$$

The output ciphertext is the following concatenation:

$$\text{Ciphertext} = 0x04 || x_1 || y_1 || \text{enciphered key} || \text{tag}$$

For key unwrapping, CKM\_IBM\_SM2\_ENCRYPT decrypts the wrapped key. Attributes required by the key type must be specified in the template.

The ephemeral key is extracted from the ciphertext and used to perform the equivalent EC-DH key agreement with the recipient's private key, producing shared secret EC point  $(x_2, y_2)$ . This is input to the same KDF producing  $t$ . The left-most 16 bytes of  $t$  are XORed with the enciphered key to recover  $K$ . Finally, the authentication  $\text{tag}$  is verified by checking to see if it matches  $\text{tag} = \text{HASH}_{\text{SM3}}(x_2 || K || y_2)$ . If the tags do not match, the unwrapping fails and returns CKR\_WRAPPED\_KEY\_INVALID.

Constraints on key types and the length of input and output data are summarized in [Table 34 on page 80](#). In the table,  $k$  is the length in bytes of the to-be-wrapped key's CKA\_VALUE attribute. Because this is limited to SM4 keys only,  $k = 16$ .

Function	Key type	Input length	Output length
<b>C_WrapKey</b>	SM2 private key	$k$	$65 + k + 2$
<b>C_UnwrapKey</b>	SM2 public key	$65 + k + 32$	$k$

For CKM\_IBM\_SM2\_ENCRYPT, the *ulMinKeySize* and *ulMaxKeySize* fields of the CK\_MECHANISM\_INFO structure are not used.

## CKM\_IBM\_SM2\_KEY\_PAIR\_GEN

CKM\_IBM\_SM2\_KEY\_PAIR\_GEN is a key pair generation mechanism for SM2.

CKM\_IBM\_SM2\_KEY\_PAIR\_GEN does not have a parameter, but generates SM2 public and private key pairs with the EC parameters defined for SM2 curve GB-256.

CKM\_IBM\_SM2\_KEY\_PAIR\_GEN adds the CKA\_CLASS, CKA\_KEY\_TYPE, CKA\_EC\_PARAMS, and CKA\_EC\_POINT attributes to the new public key and the CKA\_CLASS, CKA\_KEY\_TYPE, CKA\_EC\_PARAMS, and CKA\_VALUE attributes to the new private key. Other attributes supported by the SM2 key type (specifically, the flags indicating the functions that the key supports) may be specified in the template for the key or are assigned default initial values.

For CKM\_IBM\_SM2\_KEY\_PAIR\_GEN, the *ulMinKeySize* and *ulMaxKeySize* fields of the CK\_MECHANISM\_INFO structure are not used.

## CKM\_IBM\_SM2\_SM3

CKM\_IBM\_SM2\_SM3 is a single-part and multi-part signature generation and verification mechanism for SM2 that includes hashing with mechanism SM3. CKM\_IBM\_SM2\_SM3 has a user ID (IDA) parameter which has a maximum of 256 bytes. The mechanism generates 64-byte signatures.

Constraints on key types and the length of input and signature data are summarized in [Table 35 on page 81](#).



Function	Key type	Input length	Signature length
C_Sign	SM2 private key	Any	64 bytes
C_Verify	SM2 public key	Any, 64 <sup>2</sup>	N/A

<sup>2</sup> Data length, signature length.

For CKM\_IBM\_SM2\_SM3, the *ulMinKeySize* and *ulMaxKeySize* fields of the CK\_MECHANISM\_INFO structure are not used.

## CKM\_IBM\_SM3

CKM\_IBM\_SM3 is a message digest mechanism. CKM\_IBM\_SM3 does not have a parameter, but generates 32-byte message digests.

Constraints on the length of digest data are summarized in [Table 36 on page 81](#).

Function	Data length	Digest length
C_Digest	Any	32 bytes

## CKM\_IBM\_SM4\_CBC

CKM\_IBM\_SM4\_CBC is a mechanism for single-part encryption and decryption based on the Chinese national algorithm SM4 and cipher-block chaining mode. CKM\_IBM\_SM4\_CBC has a 16-byte initialization vector parameter.

Constraints on key types and the length of data are summarized in [Table 37 on page 81](#):

Function	Key type	Input length	Output length	Comments
C_Encrypt	SM4	Multiple of block size.	Same as input length.	No final part.
C_Decrypt	SM4	Multiple of block size.	Same as input length.	No final part.

For CKM\_IBM\_SM4\_CBC, the *ulMinKeySize* and *ulMaxKeySize* fields of the CK\_MECHANISM\_INFO structure are not used.

## CKM\_IBM\_SM4\_ECB

CKM\_IBM\_SM4\_ECB is a mechanism for single-part encryption and decryption, key wrapping, and key unwrapping based on the Chinese national algorithm SM4 and electronic codebook mode. CKM\_IBM\_SM4\_ECB does not have a parameter, but can wrap and unwrap an SM4 key.

For wrapping, CKM\_IBM\_SM4\_ECB encrypts the value of the CKA\_VALUE attribute of the key that is wrapped, padded on the trailing end with up to block size minus one null bytes so that the resulting length is a multiple of the block size. The output data is the same length as the padded input data. It does not wrap the key type, key length, or any other information about the key. Your application must supply this information separately.

For unwrapping, CKM\_IBM\_SM4\_ECB decrypts the wrapped key and truncates the result according to the CKA\_KEY\_TYPE attribute of the template and if it has one and the key type supports it, the CKA\_VALUE\_LEN attribute of the template. CKM\_IBM\_SM4\_ECB adds the result as the CKA\_VALUE attribute of the new key. Other attributes required by the key type must be specified in the template.

Constraints on key types and the length of data are summarized in [Table 38 on page 82](#):

<i>Table 38. CKM_IBM_SM4_ECB: Key and data length</i>				
<b>Function</b>	<b>Key type</b>	<b>Input length</b>	<b>Output length</b>	<b>Comments</b>
<b>C_Encrypt</b>	SM4	Multiple of block size.	Same as input length.	No final part.
<b>C_Decrypt</b>	SM4	Multiple of block size.	Same as input length.	No final part.
<b>C_WrapKey</b>	SM4	Any length.	Input length rounded up to a multiple of the block size.	None.
<b>C_UnwrapKey</b>	SM4	Multiple of block size.	Determined by the type of key being unwrapped or CKA_VALUE_LEN.	None.

For CKM\_IBM\_SM4\_ECB, the *ulMinKeySize* and *ulMaxKeySize* fields of the CK\_MECHANISM\_INFO structure are not used.

## **CKM\_IBM\_SM4\_ECB\_ENCRYPT\_DATA**

Key derivation by SM4 data encryption mechanisms allow derivation of keys using the result of an encryption operation as the key value. You can use them with the C\_DeriveKey function. The only mechanism currently defined is CKM\_IBM\_SM4\_ECB\_ENCRYPT\_DATA.

CKM\_IBM\_SM4\_ECB\_ENCRYPT\_DATA has a parameter, a CK\_KEY\_DERIVATION\_STRING\_DATA structure, that specifies the data to be encrypted to produce the key value.

CKM\_IBM\_SM4\_ECB\_ENCRYPT\_DATA functions by performing the encryption over the data provided using the base key. The resulting cipher text is used to create the key value of the resulting key. If not all the cipher text is used, the part discarded is from the trailing end (least significant bytes) of the cipher text data. The derived key is defined by the attribute template supplied, but constrained by the length of cipher text available for the key value and other normal PKCS11 derivation constraints:

- If neither length or key type is provided in the template, the key produced by CKM\_IBM\_SM4\_ECB\_ENCRYPT\_DATA is a generic secret key.
- If length is provided, but key type is not provided in the template, the key produced by CKM\_IBM\_SM4\_ECB\_ENCRYPT\_DATA is a generic secret key of the specified length.
- If a key type is provided, but length is not provided in the template, that key type must have a well-defined length. If the key type does have a well-defined length, the key produced by CKM\_IBM\_SM4\_ECB\_ENCRYPT\_DATA is of the type specified in the template. If the key type does not have a well-defined length, an error is returned.
- If both a key type and a length are provided in the template, the length must be compatible with that key type. The key produced by CKM\_IBM\_SM4\_ECB\_ENCRYPT\_DATA is of the specified type and length.

If the data is too short to make the requested key, CKM\_IBM\_SM4\_ECB\_ENCRYPT\_DATA returns CKR\_DATA\_LENGTH\_INVALID.

## **CKM\_IBM\_SM4\_ISO2\_MAC**

CKM\_IBM\_ISO2\_SM4\_MAC is a special case of the general-length ISO-padded SM4-MAC mechanism. CKM\_IBM\_ISO2\_SM4\_MAC always produces and verifies MACs that are half the block size in length. CKM\_IBM\_ISO2\_SM4\_MAC does not have a parameter.

Constraints on key types and the length of data are summarized in [Table 39 on page 83](#):

Function	Key type	Data length	Signature length
<b>C_Sign</b>	SM4	Any.	1/2 block size (8 bytes).
<b>C_Verify</b>	SM4	Any.	1/2 block size (8 bytes).

For CKM\_IBM\_ISO2\_SM4\_MAC, the *ulMinKeySize* and *ulMaxKeySize* fields of the CK\_MECHANISM\_INFO structure are not used.

## CKM\_IBM\_SM4\_ISO2\_MAC\_GENERAL

CKM\_IBM\_SM4\_ISO2\_MAC\_GENERAL is a mechanism for single-part signatures and verification based on the Chinese national algorithm SM4 with data authentication as defined by as defined in FIPS PUB 113. The data being MACed is padded according to ISO/IEC 9797-1 padding method 2.

CKM\_IBM\_SM4\_ISO2\_MAC\_GENERAL has a parameter, a CK\_MAC\_GENERAL\_PARAMS structure, that specifies the output length desired from the mechanism.

The output bytes from CKM\_IBM\_SM4\_ISO2\_MAC\_GENERAL are taken from the start of the final SM4 cipher block produced in the MACing process.

Constraints on key types and the length of data are summarized in [Table 40 on page 83](#):

Function	Key type	Data length	Signature length
<b>C_Sign</b>	SM4	Any.	4-8 bytes, as specified in parameters.
<b>C_Verify</b>	SM4	Any.	4-8 bytes, as specified in parameters.

For CKM\_IBM\_SM4\_ISO2\_MAC\_GENERAL, the *ulMinKeySize* and *ulMaxKeySize* fields of the CK\_MECHANISM\_INFO structure are not used.

## CKM\_IBM\_SM4\_KEY\_GEN

CKM\_IBM\_SM4\_KEY\_GEN is a key generation mechanism for SM4. CKM\_IBM\_SM4\_KEY\_GEN does not have a parameter, but generates SM4 keys with a length of 16 bytes.

CKM\_IBM\_SM4\_KEY\_GEN adds the CKA\_CLASS, CKA\_KEY\_TYPE, and CKA\_VALUE attributes to the new key. Other attributes supported by the SM4 key type (specifically, the flags indicating the functions that the key supports) may be specified in the template for the key or are assigned default initial values.

For CKM\_IBM\_SM4\_KEY\_GEN, the *ulMinKeySize* and *ulMaxKeySize* fields of the CK\_MECHANISM\_INFO structure are not used.

## CKM\_IBM\_SM4\_MAC

CKM\_IBM\_SM4\_MAC is a special case of the general-length SM4-MAC mechanism. CKM\_IBM\_SM4\_MAC always produces and verifies MACs that are half the block size in length. CKM\_IBM\_SM4\_MAC does not have a parameter.

Constraints on key types and the length of data are summarized in [Table 41 on page 83](#):

Function	Key type	Data length	Signature length
<b>C_Sign</b>	SM4	Any.	1/2 block size (8 bytes).
<b>C_Verify</b>	SM4	Any.	1/2 block size (8 bytes).

For CKM\_IBM\_SM4\_MAC, the *ulMinKeySize* and *ulMaxKeySize* fields of the CK\_MECHANISM\_INFO structure are not used.

## CKM\_IBM\_SM4\_MAC\_GENERAL

CKM\_IBM\_SM4\_MAC\_GENERAL is a mechanism for single-part signatures and verification based on the Chinese national algorithm SM4 with data authentication as defined in FIPS PUB 113. The data being MACed is padded according to ISO/IEC 9797-1 padding method 1 (zero-padded if not already a block multiple). CKM\_IBM\_SM4\_MAC\_GENERAL has a parameter, a CK\_MAC\_GENERAL\_PARAMS structure, that specifies the output length desired from the mechanism.

The output bytes from CKM\_IBM\_SM4\_MAC\_GENERAL are taken from the start of the final SM4 cipher block produced in the MACing process.

Constraints on key types and the length of data are summarized in [Table 42 on page 84](#):

Function	Key type	Data length	Signature length
C_Sign	SM4	Any.	4-8 bytes, as specified in parameters.
C_Verify	SM4	Any.	4-8 bytes, as specified in parameters.

For CKM\_IBM\_SM4\_MAC\_GENERAL, the *ulMinKeySize* and *ulMaxKeySize* fields of the CK\_MECHANISM\_INFO structure are not used.

## CKM\_XOR\_BASE\_AND\_DATA

Similar to mechanism CKM\_IBM\_SM4\_ECB\_ENCRYPT\_DATA, CKM\_XOR\_BASE\_AND\_DATA derives an SM4 key from another SM4 key (the base key) by XORing data with the base key.

CKM\_XOR\_BASE\_AND\_DATA has a parameter, a CK\_KEY\_DERIVATION\_STRING\_DATA structure, that specifies the data to be XORed with the base key to produce the key value.

## Additional manifest constants for regional cryptographic servers

```
#define CKK_IBM_SM2                0x80050002
#define CKM_IBM_SM2_KEY_PAIR_GEN   0x8005000A
#define CKM_IBM_SM2                0x8005000B
#define CKM_IBM_SM2_SM3           0x8005000C
#define CKM_IBM_SM2_ENCRYPT        0x8005000D
#define CKM_IBM_SM3               0x8005000E
#define CKK_IBM_SM4                0x80050001
#define CKM_IBM_SM4_KEY_GEN       0x80050001
#define CKM_IBM_SM4_ECB           0x80050004
#define CKM_IBM_SM4_CBC           0x80050002
#define CKM_IBM_SM4_MAC_GENERAL   0x80050007
#define CKM_IBM_SM4_MAC           0x80058007
#define CKM_IBM_ISO2_SM4_MAC_GENERAL 0x80050008
#define CKM_IBM_ISO2_SM4_MAC     0x80058008
#define CKM_IBM_SM4_ECB_ENCRYPT_DATA 0x80050009
#define CKM_XOR_BASE_AND_DATA     0x00000364 (standard mechanism)
```

## API examples for regional cryptographic servers

To generate an issuer MK:

```
CK_MECHANISM mech;
CK_OBJECT_HANDLE h_imkey;
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE key_gen_tmpl[] = {
    {CKA_DERIVE, &true, sizeof(true)} // key can derive other keys
};
```

```

mech.mechanism = CKM_IBM_SM4_KEY_GEN;
mech.ulParameterLen = 0;
mech.pParameter = NULL;

rc = C_GenerateKey(session, &mech, key_gen_tmpl, 1, &h_imkey);

```

To derive an ICC MK:

```

CK_OBJECT_HANDLE h_icckey;
CK_KEY_TYPE keyType = CKK_IBM_SM4;
CK_ATTRIBUTE derive_tmpl[] = {
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_DERIVE, &true, sizeof(true)} // key can derive other keys
};
CK_KEY_DERIVATION_STRING_DATA drv_data;
CK_BYTE pan[8] = { 0x12, 0x34, 0x56, 0x78, 0x12, 0x34, 0x56, 0x78 };
CK_BYTE Y[16];

memcpy(Y, pan, 8);
for (i=0; i<8; i++)
    Y[i+8] = Y[i] ^ 0xFF;

drv_data.pData = Y;
drv_data.ulLen = 16;

mech.mechanism = CKM_IBM_SM4_ECB_ENCRYPT_DATA;
mech.ulParameterLen = sizeof(drv_data);
mech.pParameter = &drv_data;

rc = C_DeriveKey(session, &mech, h_imkey, derive_tmpl, 2, &h_icckey);

```

To derive an AC session key (Note: Other session keys are derived similarly to this sample):

```

CK_OBJECT_HANDLE h_ackey;
CK_ATTRIBUTE derive_tmpl[] = {
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_SIGN, &true, sizeof(true)}, // key can sign and verify MACs
    {CKA_VERIFY, &true, sizeof(true)}
};
CK_KEY_DERIVATION_STRING_DATA drv_data;
CK_BYTE atc[2] = { 0x12, 0x34 }; // transaction counter value
CK_BYTE Z[16];

memset(Z, 0x00, sizeof(Z));
Z[6] = atc[0];
Z[7] = atc[1];
Z[14] = atc[0] ^ 0xFF;
Z[15] = atc[1] ^ 0xFF;

drv_data.pData = Z;
drv_data.ulLen = 16;

mech.mechanism = CKM_IBM_SM4_ECB_ENCRYPT_DATA;
mech.ulParameterLen = sizeof(drv_data);
mech.pParameter = &drv_data;

rc = C_DeriveKey(session, &mech, h_icckey, derive_tmpl, 3, &h_ackey);

```

To verify an ARQC: address of cryptogram data

```

CK_BYTE *pARQCdata = <address of cryptogram data>
CK_ULONG ulARQCdataLen = <length of cryptogram data>
CK_BYTE *pARQC = <address of ARQC>

mech.mechanism = CKM_IBM_ISO2_SM4_MAC; // ISO padding
mech.ulParameterLen = 0;
mech.pParameter = NULL;

rc = C_VerifyInit(session, &mech, h_ackey);
rc = C_Verify(session, pARQCdata, ulARQCdataLen, pARQC, 8);

```

To generate the ARPC :

```

CK_BYTE arc[2] = { 0x30, 0x35 }; // online declined
CK_BYTE ARPCdata[16];
CK_BYTE ARPC[8];
CK_ULONG ulARPCLen = sizeof(ARPC);

```

```

memset(ARPCdata, 0x00, sizeof(ARPCdata));
memcpy(ARPCdata, pARQC, 8);
ARPCdata[0] ^= arc[0]; // XOR leading bytes of ARQC with ARC
ARPCdata[1] ^= arc[1];

mech.mechanism = CKM_IBM_SM4_MAC; // No (or zero) padding
mech.ulParameterLen = 0;
mech.pParameter = NULL;

C_SignInit(session, &mech, h_ackey);
C_Sign(session, ARPCdata, sizeof(ARPCdata), ARPC, &ulARPClen);

```

To encrypt a script in CBC mode:

```

CK_BYTE iv[16];
CK_BYTE *pScriptData = <address of script data>
CK_ULONG ulScriptDataLen = <length of script data>

memset(iv, 0x00, sizeof(iv)); // zero IV
mech.mechanism = CKM_IBM_SM4_CBC;
mech.ulParameterLen = sizeof(iv);
mech.pParameter = iv;

C_EncryptInit(session, &mech, h_enckey);
C_Encrypt(session, pScriptData, ulScriptDataLen,
           pScriptData, &ulScriptDataLen); // encrypt in place

```

To sign (or MAC) the script:

```

CK_BYTE scriptMAC[4]; // 4-byte MAC is desired
CK_ULONG ulScriptMAClen;

ulScriptMAClen = sizeof(scriptMAC);

mech.mechanism = CKM_IBM_ISO2_SM4_MAC_GENERAL;
mech.ulParameterLen = sizeof(ulScriptMAClen);
mech.pParameter = &ulScriptMAClen;

C_SignInit(session, &mech, h_mackey);
C_Sign(session, pScriptData, ulScriptDataLen,
       scriptMAC, &ulScriptMAClen);

```

To derive a transport (or unwrapping) key from 3 key parts:

```

CK_OBJECT_HANDLE h_transkey, h_basekey, h_subkey;
CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;

CK_BYTE part1[16] = {
0x68, 0x1e, 0xdf, 0x34, 0xd2, 0x06, 0x96, 0x5e,
0x86, 0xb3, 0xe9, 0x4f, 0x53, 0x6e, 0x42, 0x46
};
CK_BYTE part2[16] = {
0xf3, 0x24, 0x18, 0x4f, 0x3c, 0x88, 0x92, 0xb7,
0x2b, 0xdc, 0x9d, 0x7c, 0x61, 0x29, 0x19, 0xde
};
CK_BYTE part3[16] = {
0xce, 0x4d, 0xd6, 0xb8, 0x1f, 0x6d, 0xe9, 0x92,
0xa8, 0x30, 0xda, 0xab, 0x1f, 0x00, 0x80, 0x28
};

CK_ATTRIBUTE create_tmpl[] = {
    {CKA_CLASS, &keyClass, sizeof(keyClass)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_DERIVE, &true, sizeof(true)}, // part 1 can derive
    {CKA_VALUE, part1, sizeof(part1)}
};

CK_ATTRIBUTE derive_tmpl[] = {
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_DERIVE, &true, sizeof(true)} // partial keys derive other keys
};

CK_ATTRIBUTE final_derive_tmpl[] = {
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_UNWRAP, &true, sizeof(true)} // final key can unwrap other keys
};

```

```

CK_KEY_DERIVATION_STRING_DATA drv_data;

mech.mechanism = CKM_XOR_BASE_AND_DATA;
mech.ulParameterLen = sizeof(drv_data);
mech.pParameter = &drv_data;

// Create part 1
rc = C_CreateObject(session, &create_tmpl, 4, &h_basekey);

// XOR part 2
drv_data.pData = part2;
drv_data.ulLen = sizeof(part2);
rc = C_DeriveKey(session, &mech, h_basekey, derive_tmpl, 2, &h_subkey);

// Destroy no longer needed partial key
rc = C_DestroyObject(session, h_basekey);
h_basekey = h_subkey;

// XOR part 3
drv_data.pData = part3;
drv_data.ulLen = sizeof(part3);
rc = C_DeriveKey(session, &mech, h_basekey, final_derive_tmpl, 2,
&h_subkey);

// Destroy no longer needed partial key
rc = C_DestroyObject(session, h_basekey);

h_transkey = h_subkey; // final derivation is the transport key

```

To import (or unwrap) an issuer MK generated elsewhere:

```

CK_ATTRIBUTE unwrap_tmpl[] = {
    {CKA_CLASS, &keyClass, sizeof(keyClass)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_DERIVE, &true, sizeof(true)} // im keys can derive
};

CK_BYTE *wrp_data = <address of wrapped key data>
CK_ULONG wrp_data_len = <length of wrapped data>

mech.mechanism = CKM_IBM_SM4_ECB;
mech.ulParameterLen = 0;
mech.pParameter = NULL;

rc = C_UnwrapKey(session, &mech, h_transkey, wrp_data, wrp_data_len,
unwrap_tmpl, 3, &h_imkey);

```





---

## Chapter 5. ICSF PKCS #11 callable services

The PKCS #11 C language API (described in [Chapter 2, “The C API,”](#) on page 19) requires a Language Environment (LE) runtime to operate. Although an LE is normally provided with C application programs, if you are coding your application in some other language (for example, Assembler), acquiring an LE runtime may not be desirable. For these situations, ICSF provides a base set of PKCS #11 callable services that you can use. (In fact, the C API itself uses these services.) These callable services do not require an LE runtime. The ICSF PKCS #11 callable services include:

- Derive key (CSFPDVK)
- Derive multiple keys (CSFPDMK)
- Generate MAC (CSFPHMG)
- Generate key pair (CSFPGKP)
- Generate secret key (CSFPGSK)
- Get attribute value (CSFPGAV)
- One-way hash generate (CSFPOWH)
- Private key sign (CSFPPKS)
- Pseudo-random function (CSFPPRF)
- Public key verify (CSFPPKV)
- Secret key decrypt (CSFPSKD)
- Secret key encrypt (CSFPSKE)
- Set attribute value (CSFPSAV)
- Token record create (CSFPTRC)
- Token record delete (CSFPTRD)
- Token record list (CSFPTRL)
- Unwrap key (CSFPUWK)
- Verify MAC (CSFPHMV)
- Wrap key (CSFPWPK)

Calls to the system authorization facility (SAF) determine access authorization for the callable services. The CSFSERV class controls access to the PKCS #11 callable services.

For details about the PKCS #11 callable services, see [\*z/OS Cryptographic Services ICSF Application Programmer's Guide\*](#).



---

## Appendix A. SMP/E installation data sets, directories, and files

The following dynamic link libraries (DLLs) are linked into SYS1.SIEALNKE:

### **CSNPCAPI**

The main DLL invoked by applications to use PKCS #11 functions. Also shipped as an HFS file at /usr/lpp/pkcs11/lib/csnpcapi.so.

### **CSNPCA64**

64-bit addressing mode version of CSNPCAPI. Also shipped as an HFS file at /usr/lpp/pkcs11/lib/csnpca64.so.

### **CSNPCA3X**

31-bit addressing mode version of CSNPCAPI with XPLINK. Also shipped as an HFS file at /usr/lpp/pkcs11/lib/csnpca3x.so

### **CSNPCINT**

An internal DLL loaded by CSNPCAPI.

### **CSNPCI64**

64-bit addressing mode version of CSNPCINT.

### **CSNPCI3X**

31-bit addressing mode version of CSNPCINT with XPLINK.

### **CSNPCUTL**

An internal DLL implicitly loaded for utilities.

### **CSNPCU64**

64-bit addressing mode version of CSNPCUTL.

### **CSNPCU3X**

31-bit addressing mode version of CSNPCUTL with XPLINK.

### **CSFDLL31**

CSFDLL in 31-bit addressing mode

### **CSFDLL64**

CSFDLL in 64-bit addressing mode.

### **CSFDLL3X**

31-bit addressing mode version of CSFDLL31 with XPLINK.

SMP/E installs the product files into the HFS directory /usr/lpp/pkcs11. This directory contains the following subdirectories and files:

- /usr/lpp/pkcs11/include subdirectory (members are symbolically linked to /usr/include)

#### **csnpdefs.h**

A header file that applications must include to use PKCS #11 functions. Also copied to SYS1.SIEAHDR.H(CSNPDEFS).

#### **csfbext.h**

A header file that applications must include to use the CSFDLLs. Also copied to SYS1.SIEAHDR.H(CSFBEXT).

- /usr/lpp/pkcs11/lib subdirectory (members are symbolically linked to /usr/lib)

#### **CSNPCAPI.x**

Side deck for CSNPCAPI. Also copied to SYS1.SIEASID(CSNPCAPI).

#### **CSNPCA64.x**

Side deck for CSNPCA64. Also copied to SYS1.SIEASID(CSNPCA64).

**CSNPCA3X.x**

Side deck for CSNPCA3X. Also copied to SYS1.SIEASID(CSNPCA3X).

**csnpcapi.so****csnpca64.so****csnpca3x.so****CSFDLL31.x**

Side deck for CSFDLL31. Also copied to SYS1.SIEASID(CSFDLL31).

**CSFDLL64.x**

Side deck for CSFDLL64. Also copied to SYS1.SIEASID(CSFDLL64).

**CSFDLL3X.x**

Side deck for CSFDLL3X. Also copied to SYS1.SIEASID(CSFDLL3X).

- usr/lpp/pkcs11/bin subdirectory

**testpkcs11**

Program to test system configuration for PKCS #11.

- /usr/lpp/pkcs11/samples subdirectory

**testpkcs11.c**

Source code for the testpkcs11 program.

**Makefile.testpkcs11**

Makefile for the testpkcs11 program.

**Makefile3X.testpkcs11**

Makefile for 31-bit addressing mode version of the testpkcs11 program with XPLINK.

**Makefile64.testpkcs11**

Makefile for 64-bit addressing mode version of the testpkcs11 program.

**updatecomp.c**

Source code for the updatecomp.c program.

**Makefile.updatecomp**

Makefile for the updatecomp.c program.

**Makefile3X.updatecomp**

Makefile for 31-bit addressing mode version of updatecomp.c program with XPLINK.

**Makefile64.updatecomp**

Makefile for 64-bit addressing mode version of the updatecomp.c program.

# Appendix B. Source code for the testpkcs11 sample program

For information about building and using the testpkcs11 sample program, see [Chapter 3, “Sample PKCS #11 C programs,”](#) on page 73.

```

/*****
/*
/* COMPONENT_NAME: testpkcs11.c
/*
/* Licensed Materials - Property of IBM
/* 5650-ZOS
/* Copyright IBM Corp. 2007, 2013
/* Status = HCR7770
/*
/*****
/*****
/*
/* This file contains sample code. IBM PROVIDES THIS CODE ON AN
/* 'AS IS' BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR
/* IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
/* OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
/*
/*****
/*
/* Change Activity:
/* $L0=P11C1 ,HCR7740, 060124,PDJS: PKCS11 initial release
/* $D1=MG08269 ,HCR7740, 061114,PDJS: Misc fixes
/* $D2=MG08740 ,HCR7740, 070302,PDGL: XPLINK
/* $P1=MG13406 ,HCR7770, 090812,PDGL: fix XPLINK define
/* $P2=MG13431 ,HCR7770, 090826,PDER: update prolog
/*****

#ifdef IBM
/* Customers may remove this copyright statement */
#pragma comment (copyright,"
Licensed Materials - Property of IBM \
5650-ZOS Copyright IBM Corp. 2007, 2013 \
All Rights Reserved \
US Government Users Restricted Rights - \
Use, duplication or disclosure restricted by \
GSA ADP Schedule Contract with IBM Corp.")
#endif

/* Install verification test for PKCS #11 */

#define _UNIX03_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <dlfcn.h>
#include <sys/timeb.h>
#include <csnpdefs.h>

int skip_token_obj;

CK_FUNCTION_LIST *funcs;
CK_SLOT_ID slotID = CK_UNAVAILABLE_INFORMATION;
CK_BYTE tokenName[32];

void ProcessRetCode( CK_RV rc )
{
    switch (rc) {
        case CKR_OK: printf(" CKR_OK"); break;
        case CKR_CANCEL: printf(" CKR_CANCEL"); break;
        case CKR_HOST_MEMORY: printf(" CKR_HOST_MEMORY"); break;
        case CKR_SLOT_ID_INVALID: printf(" CKR_SLOT_ID_INVALID"); break;
        case CKR_GENERAL_ERROR: printf(" CKR_GENERAL_ERROR"); break;
        case CKR_FUNCTION_FAILED: printf(" CKR_FUNCTION_FAILED"); break;
        case CKR_ARGUMENTS_BAD: printf(" CKR_ARGUMENTS_BAD"); break;
    }
}

```

```

case CKR_NO_EVENT:                printf(" CKR_NO_EVENT");                break;
case CKR_NEED_TO_CREATE_THREADS:  printf(" CKR_NEED_TO_CREATE_THREADS");  break;
case CKR_CANT_LOCK:               printf(" CKR_CANT_LOCK");               break;
case CKR_ATTRIBUTE_READ_ONLY:     printf(" CKR_ATTRIBUTE_READ_ONLY");     break;
case CKR_ATTRIBUTE_SENSITIVE:     printf(" CKR_ATTRIBUTE_SENSITIVE");     break;
case CKR_ATTRIBUTE_TYPE_INVALID:  printf(" CKR_ATTRIBUTE_TYPE_INVALID");  break;
case CKR_ATTRIBUTE_VALUE_INVALID: printf(" CKR_ATTRIBUTE_VALUE_INVALID");  break;
case CKR_DATA_INVALID:            printf(" CKR_DATA_INVALID");            break;
case CKR_DATA_LEN_RANGE:          printf(" CKR_DATA_LEN_RANGE");          break;
case CKR_DEVICE_ERROR:            printf(" CKR_DEVICE_ERROR");            break;
case CKR_DEVICE_MEMORY:           printf(" CKR_DEVICE_MEMORY");           break;
case CKR_DEVICE_REMOVED:          printf(" CKR_DEVICE_REMOVED");          break;
case CKR_ENCRYPTED_DATA_INVALID:   printf(" CKR_ENCRYPTED_DATA_INVALID");   break;
case CKR_ENCRYPTED_DATA_LEN_RANGE: printf(" CKR_ENCRYPTED_DATA_LEN_RANGE");  break;
case CKR_FUNCTION_CANCELED:       printf(" CKR_FUNCTION_CANCELED");       break;
case CKR_FUNCTION_NOT_PARALLEL:   printf(" CKR_FUNCTION_NOT_PARALLEL");   break;
case CKR_FUNCTION_NOT_SUPPORTED:  printf(" CKR_FUNCTION_NOT_SUPPORTED");  break;
case CKR_KEY_HANDLE_INVALID:      printf(" CKR_KEY_HANDLE_INVALID");      break;
case CKR_KEY_SIZE_RANGE:          printf(" CKR_KEY_SIZE_RANGE");          break;
case CKR_KEY_TYPE_INCONSISTENT:   printf(" CKR_KEY_TYPE_INCONSISTENT");   break;
case CKR_KEY_NOT_NEEDED:          printf(" CKR_KEY_NOT_NEEDED");          break;
case CKR_KEY_CHANGED:             printf(" CKR_KEY_CHANGED");             break;
case CKR_KEY_NEEDED:              printf(" CKR_KEY_NEEDED");              break;
case CKR_KEY_INDIGESTIBLE:        printf(" CKR_KEY_INDIGESTIBLE");        break;
case CKR_KEY_FUNCTION_NOT_PERMITTED: printf(" CKR_KEY_FUNCTION_NOT_PERMITTED"); break;
case CKR_KEY_NOT_WRAPPABLE:       printf(" CKR_KEY_NOT_WRAPPABLE");       break;
case CKR_KEY_UNEXTRACTABLE:       printf(" CKR_KEY_UNEXTRACTABLE");       break;
case CKR_MECHANISM_INVALID:       printf(" CKR_MECHANISM_INVALID");       break;
case CKR_MECHANISM_PARAM_INVALID: printf(" CKR_MECHANISM_PARAM_INVALID");  break;
case CKR_OBJECT_HANDLE_INVALID:   printf(" CKR_OBJECT_HANDLE_INVALID");   break;
case CKR_OPERATION_ACTIVE:        printf(" CKR_OPERATION_ACTIVE");        break;
case CKR_OPERATION_NOT_INITIALIZED: printf(" CKR_OPERATION_NOT_INITIALIZED"); break;
case CKR_PIN_INCORRECT:           printf(" CKR_PIN_INCORRECT");           break;
case CKR_PIN_INVALID:             printf(" CKR_PIN_INVALID");             break;
case CKR_PIN_LEN_RANGE:           printf(" CKR_PIN_LEN_RANGE");           break;
case CKR_PIN_EXPIRED:             printf(" CKR_PIN_EXPIRED");             break;
case CKR_PIN_LOCKED:              printf(" CKR_PIN_LOCKED");              break;
case CKR_SESSION_CLOSED:          printf(" CKR_SESSION_CLOSED");          break;
case CKR_SESSION_COUNT:           printf(" CKR_SESSION_COUNT");           break;
case CKR_SESSION_HANDLE_INVALID:  printf(" CKR_SESSION_HANDLE_INVALID");  break;
case CKR_SESSION_PARALLEL_NOT_SUPPORTED: printf(" CKR_SESSION_PARALLEL_NOT_SUPPORTED"); break;
case CKR_SESSION_READ_ONLY:       printf(" CKR_SESSION_READ_ONLY");       break;
case CKR_SESSION_EXISTS:          printf(" CKR_SESSION_EXISTS");          break;
case CKR_SESSION_READ_ONLY_EXISTS: printf(" CKR_SESSION_READ_ONLY_EXISTS");  break;
case CKR_SESSION_READ_WRITE_SO_EXISTS: printf(" CKR_SESSION_READ_WRITE_SO_EXISTS"); break;
case CKR_SIGNATURE_INVALID:       printf(" CKR_SIGNATURE_INVALID");       break;
case CKR_SIGNATURE_LEN_RANGE:     printf(" CKR_SIGNATURE_LEN_RANGE");     break;
case CKR_TEMPLATE_INCOMPLETE:     printf(" CKR_TEMPLATE_INCOMPLETE");     break;
case CKR_TEMPLATE_INCONSISTENT:   printf(" CKR_TEMPLATE_INCONSISTENT");   break;
case CKR_TOKEN_NOT_PRESENT:       printf(" CKR_TOKEN_NOT_PRESENT");       break;
printf(" CKR_TOKEN_NOT_PRESENT - ICSF is not active or not configured for TKDS operations");
break;
case CKR_TOKEN_NOT_RECOGNIZED:    printf(" CKR_TOKEN_NOT_RECOGNIZED - You are not authorized to perform the token operation");
break;
case CKR_TOKEN_WRITE_PROTECTED:   printf(" CKR_TOKEN_WRITE_PROTECTED");   break;
case CKR_UNWRAPPING_KEY_HANDLE_INVALID: printf(" CKR_UNWRAPPING_KEY_HANDLE_INVALID"); break;
case CKR_UNWRAPPING_KEY_SIZE_RANGE: printf(" CKR_UNWRAPPING_KEY_SIZE_RANGE"); break;
case CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT: printf(" CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT"); break;
case CKR_USER_ALREADY_LOGGED_IN:  printf(" CKR_USER_ALREADY_LOGGED_IN");  break;
case CKR_USER_NOT_LOGGED_IN:      printf(" CKR_USER_NOT_LOGGED_IN");      break;
case CKR_USER_PIN_NOT_INITIALIZED: printf(" CKR_USER_PIN_NOT_INITIALIZED");  break;
case CKR_USER_TYPE_INVALID:       printf(" CKR_USER_TYPE_INVALID");       break;
case CKR_USER_ANOTHER_ALREADY_LOGGED_IN: printf(" CKR_USER_ANOTHER_ALREADY_LOGGED_IN"); break;
case CKR_USER_TOO_MANY_TYPES:     printf(" CKR_USER_TOO_MANY_TYPES");     break;
case CKR_WRAPPED_KEY_INVALID:     printf(" CKR_WRAPPED_KEY_INVALID");     break;
case CKR_WRAPPED_KEY_LEN_RANGE:   printf(" CKR_WRAPPED_KEY_LEN_RANGE");   break;
case CKR_WRAPPING_KEY_HANDLE_INVALID: printf(" CKR_WRAPPING_KEY_HANDLE_INVALID"); break;
case CKR_WRAPPING_KEY_SIZE_RANGE: printf(" CKR_WRAPPING_KEY_SIZE_RANGE");  break;
case CKR_WRAPPING_KEY_TYPE_INCONSISTENT: printf(" CKR_WRAPPING_KEY_TYPE_INCONSISTENT"); break;
case CKR_RANDOM_SEED_NOT_SUPPORTED: printf(" CKR_RANDOM_SEED_NOT_SUPPORTED"); break;
case CKR_RANDOM_NO_RNG:           printf(" CKR_RANDOM_NO_RNG");           break;
case CKR_BUFFER_TOO_SMALL:        printf(" CKR_BUFFER_TOO_SMALL");        break;
case CKR_SAVED_STATE_INVALID:     printf(" CKR_SAVED_STATE_INVALID");     break;
case CKR_INFORMATION_SENSITIVE:   printf(" CKR_INFORMATION_SENSITIVE");   break;
case CKR_STATE_UNSAVEABLE:        printf(" CKR_STATE_UNSAVEABLE");        break;
case CKR_CRYPTOKI_NOT_INITIALIZED: printf(" CKR_CRYPTOKI_NOT_INITIALIZED");  break;
case CKR_CRYPTOKI_ALREADY_INITIALIZED: printf(" CKR_CRYPTOKI_ALREADY_INITIALIZED"); break;
case CKR_MUTEX_BAD:               printf(" CKR_MUTEX_BAD");               break;
case CKR_MUTEX_NOT_LOCKED:        printf(" CKR_MUTEX_NOT_LOCKED");        break;
/* Otherwise - Value does not match a known PKCS11 return value */
}

```

```

}

void showError( char *str, CK_RV rc )
{
    printf("%s returned:  %d (0x%0x)", str, rc, rc );
    ProcessRetCode( rc );
    printf("\n");
}

CK_RV createToken( void )
{
    CK_VOID_PTR      p = NULL; // @D1C
    CK_RV            rc;
    CK_FLAGS         flags = 0;

    printf("Creating the temporary token... \n");
    /* wait for slot event. On z/OS this creates a new slot synchronously */
    rc = funcs->C_WaitForSlotEvent(flags, &slotID, p);
    if (rc != CKR_OK) {
        showError(" C_WaitForSlotEvent #1", rc );
        return !CKR_OK;
    }
    /* The slot has been created. Now initialize the token in the slot */
    /* On z/OS no PIN is required, so we will pass NULL. */
    rc = funcs->C_InitToken(slotID, NULL, 0, tokenName);
    if (rc != CKR_OK) {
        showError(" C_InitToken #1", rc );
        if (rc == CKR_ARGUMENTS_BAD) {
            printf(" Make sure your the token name you specified meets ICSF rules:\n");
            printf(" Contains only alphanumeric characters, nationals (@#$$), and periods.\n");
            printf(" The first character cannot be a numeric or a period.\n");
        }
        return !CKR_OK;
    }
}

return CKR_OK;
}

CK_RV deleteToken( void )
{
    CK_VOID_PTR      p;
    CK_RV            rc;
    CK_FLAGS         flags = 0;

    if (slotID != CK_UNAVAILABLE_INFORMATION) {
        printf("Deleting the temporary token... \n");
        /* C_InitToken with the reserved label $$DELETE-TOKEN$$ is the way to delete a token */
        /* on z/OS */
        memset(tokenName, ' ', sizeof(tokenName));
        memcpy(tokenName, DEL_TOK, sizeof(DEL_TOK));
        rc = funcs->C_InitToken(slotID, NULL, 0, tokenName);
        if (rc != CKR_OK) {
            showError(" C_InitToken #2 (for delete)", rc );
            return !CKR_OK;
        }
    }
}

return CKR_OK;
}

CK_RV encryptRSA( void )
{
    CK_BYTE          data1[100];
    CK_BYTE          data2[256];
    CK_BYTE          cipher[256];
    CK_SLOT_ID      slot_id;
    CK_SESSION_HANDLE session;
    CK_MECHANISM     mech;
    CK_OBJECT_HANDLE publ_key, priv_key;
    CK_FLAGS         flags;
    CK_ULONG        i;
    CK_ULONG        len1, len2, cipherlen;
    CK_RV           rc;
    static CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
    static CK_KEY_TYPE    type = CKK_RSA;
    static CK_OBJECT_CLASS privclass = CKO_PRIVATE_KEY;
    static CK_BBOOL       true = TRUE;
    static CK_BBOOL       false = FALSE;
}

```

```

static CK_ULONG bits = 1024;
static CK_BYTE pub_exp[] = { 0x01, 0x00, 0x01 };

/* Attributes for the public key to be generated */
CK_ATTRIBUTE pub_tmpl[] = {
    {CKA_MODULUS_BITS, &bits, sizeof(bits) },
    {CKA_ENCRYPT, &true, sizeof(true) },
    {CKA_VERIFY, &true, sizeof(true) },
    {CKA_PUBLIC_EXPONENT, &pub_exp, sizeof(pub_exp) }
};

/* Attributes for the private key to be generated */
CK_ATTRIBUTE priv_tmpl[] =
{
    {CKA_DECRYPT, &true, sizeof(true) },
    {CKA_SIGN, &true, sizeof(true) }
};

slot_id = slotID;
flags = CKF_SERIAL_SESSION | CKF_RW_SESSION;
printf("Opening a session... \n");
rc = funcs->C_OpenSession( slot_id, flags, (CK_VOID_PTR) NULL, NULL, &session );
if (rc != CKR_OK) {
    showError(" C_OpenSession #1", rc );
    return !CKR_OK;
}

printf("Generating keys. This may take a while... \n");
mech.mechanism = CKM_RSA_PKCS_KEY_PAIR_GEN;
mech.ulParameterLen = 0;
mech.pParameter = NULL;

rc = funcs->C_GenerateKeyPair( session, &mech,
                               pub_tmpl, 4,
                               priv_tmpl, 2,
                               &publ_key, &priv_key );

if (rc != CKR_OK) {
    showError(" C_GenerateKeyPair #1", rc );
    return !CKR_OK;
}

/* now, encrypt some data */
len1 = sizeof(data1);
len2 = sizeof(data2);
cipherlen = sizeof(cipher);

for (i=0; i < len1; i++)
    data1[i] = (i) % 255;

mech.mechanism = CKM_RSA_PKCS;
mech.ulParameterLen = 0;
mech.pParameter = NULL;

printf("Enciphering data... \n");
rc = funcs->C_EncryptInit( session, &mech, publ_key );
if (rc != CKR_OK) {
    showError(" C_EncryptInit #1", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

rc = funcs->C_Encrypt( session, data1, len1, cipher, &cipherlen );
if (rc != CKR_OK) {
    showError(" C_Encrypt #1", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

/* now, decrypt the data */
printf("Deciphering data... \n");
rc = funcs->C_DecryptInit( session, &mech, priv_key );
if (rc != CKR_OK) {
    showError(" C_DecryptInit #1", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

rc = funcs->C_Decrypt( session, cipher, cipherlen, data2, &len2 );
if (rc != CKR_OK) {
    showError(" C_Decrypt #1", rc );
    funcs->C_CloseSession( session );
}

```



```

    return !CKR_OK;
}

/* PKCS - returns clear data as is */
if (len1 != len2) {
    printf("    ERROR: lengths do not match\n");
    printf("    Length of original data = %d, after decryption = %d\n",len1, len2);
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

for (i=0; i <len1; i++) {
    if (data1[i] != data2[i]) {
        printf("    ERROR: mismatch at byte %d\n", i );
        funcs->C_CloseSession( session );
        return !CKR_OK;
    }
}

printf("Destroying keys... \n");
rc = funcs->C_DestroyObject( session, priv_key );
if (rc != CKR_OK) {
    showError("    C_DestroyObject #1", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

rc = funcs->C_DestroyObject( session, publ_key );
if (rc != CKR_OK) {
    showError("    C_DestroyObject #2", rc );
    funcs->C_CloseSession( session );
    return !CKR_OK;
}

printf("Closing the session... \n");
rc = funcs->C_CloseSession( session );
if (rc != CKR_OK) {
    showError("    C_CloseSession #1", rc );
    return !CKR_OK;
}

return CKR_OK;
}

CK_RV getFunctionList( void )
{
    CK_RV      rc;
    CK_RV      (*pFunc)();
    void       *d;
#ifdef _LP64
    char       e[]="CSNPCA64";
#elif __XPLINK__
    char       e[]="CSNPCA3X";          /* @P1C */
#else
    char       e[]="CSNPCAPI";
#endif

    printf("Getting the PKCS11 function list...\n");

    d = dlopen(e,RTLD_NOW);
    if ( d == NULL ) {
        printf("%s not found in linklist or LIBPATH\n",e); // @D1A
        return !CKR_OK;
    }

    pFunc = (CK_RV (*)())dlsym(d,"C_GetFunctionList");
    if (pFunc == NULL) {
        printf("C_GetFunctionList() not found in module %s\n",e); // @D1A
        return !CKR_OK;
    }
    rc = pFunc(&funcs);

    if (rc != CKR_OK) {
        showError("    C_GetFunctionList", rc );
        return !CKR_OK;
    }

    return CKR_OK;
}

```

```

}

void displaySyntax(char *pgm) {
    printf("usage: %s { -t <token-name> | -h }\n\n", pgm );
    printf(" -t <token-name> = The name of a temporary token to create for the test. The\n");
    printf("   name must be less than 33 characters in length and contains only alphanumeric\n");
    printf("   characters, nationals (@#$), and periods. The first character cannot be a\n");
    printf("   numeric or a period. The token will be deleted when the test is complete.\n\n");
    printf(" -h = Displays this help.\n\n");
}

void main( int argc, char **argv )
{
    CK_C_INITIALIZE_ARGS  cinit_args;
    CK_RV                rc, i;

    memset(tokenName, ' ', sizeof(tokenName)); /* Token name is left justified, padded with blanks */

    if (argc == 3) {
        if (strcmp(argv[1], "-t") == 0)
            if (strlen(argv[2]) > 0 && strlen(argv[2]) < 33) {
                memcpy(tokenName, argv[2], strlen(argv[2]));
            }
            else {
                displaySyntax(argv[0]);
                return;
            }
        else {
            displaySyntax(argv[0]);
            return;
        }
    }
    else {
        displaySyntax(argv[0]);
        return;
    }

    rc = getFunctionList();
    if (rc != CKR_OK) {
        printf("getFunctionList failed!\n"); // @D1C
        return;
    }

    memset( &cinit_args, 0x0, sizeof(cinit_args) );
    cinit_args.flags = CKF_OS_LOCKING_OK;

    printf("Initializing the PKCS11 environment...\n");
    rc = funcs->C_Initialize( &cinit_args );
    if (rc != CKR_OK) {
        showError(" C_Initialize", rc );
        return;
    }

    rc = createToken();
    if (rc != CKR_OK) {
        funcs->C_Finalize( NULL );
        return;
    }

    rc = encryptRSA();
    if (rc != CKR_OK) {
        deleteToken();
        funcs->C_Finalize( NULL );
        return;
    }

    rc = deleteToken();
    if (rc != CKR_OK) {
        funcs->C_Finalize( NULL );
        return;
    }

    rc = funcs->C_Finalize( NULL );
    if (rc != CKR_OK) {
        showError(" C_Initialize", rc );
        return;
    }
    printf("Test completed successfully!\n");
}

```





---

## Appendix C. Accessibility

Accessible publications for this product are offered through [IBM Knowledge Center \(www.ibm.com/support/knowledgecenter/SSLTBW/welcome\)](http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the Contact the z/OS team web page ([www.ibm.com/systems/campaignmail/z/zos/contact\\_z](http://www.ibm.com/systems/campaignmail/z/zos/contact_z)) or use the following mailing address.

IBM Corporation  
Attention: MHVRCFS Reader Comments  
Department H6MA, Building 707  
2455 South Road  
Poughkeepsie, NY 12601-5400  
United States

---

### Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

---

### Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

---

### Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Vol I*

---

### Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The \* symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element \*FILE with dotted decimal number 3 is given the format 3 \\* FILE. Format 3\* FILE indicates that syntax element FILE repeats. Format 3\* \\* FILE indicates that syntax element \* FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines 5.1\*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

#### **? indicates an optional syntax element**

The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

#### **! indicates a default syntax element**

The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE (KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

#### **\* indicates an optional syntax element that is repeatable**

The asterisk or glyph (\*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the \* symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1\* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3\* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

#### **Notes:**

1. If a dotted decimal number has an asterisk (\*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.

2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.
3. The \* symbol is equivalent to a loopback line in a railroad syntax diagram.

**+ indicates a syntax element that must be included**

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the \* symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the \* symbol, is equivalent to a loopback line in a railroad syntax diagram.





## Notices

---

This information was developed for products and services that are offered in the USA or elsewhere.

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

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

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive, MD-NC119  
Armonk, NY 10504-1785  
United States of America*

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.  
19-21, Nihonbashi-Hakozakicho, Chuo-ku  
Tokyo 103-8510, 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.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for the Knowledge Centers. Use of hyperlinks in other output formats of this information is at your own risk.

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

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

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

*IBM Corporation  
Site Counsel  
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 document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

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

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

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

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

#### COPYRIGHT LICENSE:

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

## Terms and conditions for product documentation

---

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

### **Applicability**

These terms and conditions are in addition to any terms of use for the IBM website.

### **Personal use**

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

### **Commercial use**

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

## Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

## IBM Online Privacy Statement

---

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at [ibm.com/privacy](http://ibm.com/privacy) and IBM's Online Privacy Statement at [ibm.com/privacy/details](http://ibm.com/privacy/details) in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at [ibm.com/software/info/product-privacy](http://ibm.com/software/info/product-privacy).

## Policy for unsupported hardware

---

Various z/OS elements, such as DFSMSdfp, JES2, JES3, and MVS™, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

## Minimum supported hardware

---

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS \(www.ibm.com/software/support/systemsz/lifecycle\)](http://www.ibm.com/software/support/systemsz/lifecycle)

- For information about currently-supported IBM hardware, contact your IBM representative.

## Trademarks

---

IBM, the IBM logo, and [ibm.com](http://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 \(www.ibm.com/legal/copytrade.shtml\)](http://www.ibm.com/legal/copytrade.shtml).

# Glossary

---

This glossary defines terms and abbreviations used in Integrated Cryptographic Service Facility (ICSF).

This glossary includes terms and definitions from:

- The American National Standard Dictionary for Information Technology, ANSI INCITS 172, by the American National Standards Institute (ANSI). Copies can be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- The Information Technology Vocabulary, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

Definitions specific to the Integrated Cryptographic Services Facility are labeled “In ICSF.”

## **access method services (AMS)**

The facility used to define and reproduce VSAM key-sequenced data sets (KSDS).

## **Advanced Encryption Standard (AES)**

In computer security, the National Institute of Standards and Technology (NIST) Advanced Encryption Standard (AES) algorithm.

## **AES**

Advanced Encryption Standard.

## **American National Standard Code for Information Interchange (ASCII)**

The standard code using a coded character set consisting of 7-bit characters (8 bits including parity check) that is used for information exchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

## **ANSI X9.19**

An ANSI standard that specifies an optional double-MAC procedure which requires a double-length MAC key.

## **application program**

A program written for or by a user that applies to the user's work, such as a program that does inventory control or payroll.

A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities.

## **application program interface (API)**

A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

In ICSF, a callable service.

## **asymmetric cryptography**

Synonym for public key cryptography.

## **authentication pattern**

An 8-byte pattern that ICSF calculates from the master key when initializing the cryptographic key data set. ICSF places the value of the authentication pattern in the header record of the cryptographic key data set.

## **authorized program facility (APF)**

A facility that permits identification of programs authorized to use restricted functions.

**callable service**

A predefined sequence of instructions invoked from an application program, using a CALL instruction. In ICSF, callable services perform cryptographic functions and utilities.

**CBC**

Cipher block chaining.

**CCA**

Common Cryptographic Architecture.

**CCF**

Cryptographic Coprocessor Feature.

**CDMF**

Commercial Data Masking Facility.

**CEDA**

A CICS transaction that defines resources online. Using CEDA, you can update both the CICS system definition data set (CSD) and the running CICS system.

**Central Credit Committee**

The official English name for *Zentraler Kreditausschuss*, also known as ZKA. ZKA was founded in 1932 and was renamed in August 2011 to *Die Deutsche Kreditwirtschaft*, also known as DK. DK is an association of the German banking industry. The hybrid term in English for DK is 'German Banking Industry Committee'.

**CEX2A**

Crypto Express2 Accelerator

**CEX2C**

Crypto Express2 Coprocessor

**CEX3A**

Crypto Express3 Accelerator

**CEX3C**

Crypto Express3 Coprocessor

**CEX4A**

Crypto Express4 Accelerator

**CEX4C**

Crypto Express4 CCA Coprocessor

**CEX4P**

Crypto Express4 PKCS #11 Coprocessor

**CEX5A**

Crypto Express5 Accelerator

**CEX5C**

Crypto Express5 CCA Coprocessor

**CEX5P**

Crypto Express5 PKCS #11 Coprocessor

**CEX6A**

Crypto Express6 Accelerator

**CEX6C**

Crypto Express6 CCA Coprocessor

**CEX6P**

Crypto Express6 PKCS #11 Coprocessor

**CEX7A**

Crypto Express7 Accelerator

**CEX7C**

Crypto Express7 CCA Coprocessor

**CEX7P**

Crypto Express7 PKCS #11 Coprocessor

**checksum**

The sum of a group of data associated with the group and used for checking purposes. (T)

In ICSF, the data used is a key part. The resulting checksum is a two-digit value you enter when you enter a master key part.

**Chinese Remainder Theorem (CRT)**

A mathematical theorem that defines a format for the RSA private key that improves performance.

**CICS**

Customer Information Control System.

**cipher block chaining (CBC)**

A mode of encryption that uses the data encryption algorithm and requires an initial chaining vector. For encipher, it exclusively ORs the initial block of data with the initial control vector and then enciphers it. This process results in the encryption both of the input block and of the initial control vector that it uses on the next input block as the process repeats. A comparable chaining process works for decipher.

**ciphertext**

In computer security, text produced by encryption.

Synonym for enciphered data.

**CKDS**

Cryptographic Key Data Set.

**clear key**

Any type of encryption key not protected by encryption under another key.

**CMOS**

Complementary metal oxide semiconductor.

**coexistence mode**

An ICSF method of operation during which CUSP or PCF can run independently and simultaneously on the same ICSF system. A CUSP or PCF application program can run on ICSF in this mode if the application program has been reassembled.

**Commercial Data Masking Facility (CDMF)**

A data-masking algorithm using a DES-based kernel and a key that is shortened to an effective key length of 40 DES key-bits. Because CDMF is not as strong as DES, it is called a masking algorithm rather than an encryption algorithm. Implementations of CDMF, when used for data confidentiality, are generally exportable from the USA and Canada.

**Common Cryptographic Architecture: Cryptographic Application Programming Interface**

Defines a set of cryptographic functions, external interfaces, and a set of key management rules that provide a consistent, end-to-end cryptographic architecture across different IBM platforms.

**compatibility mode**

An ICSF method of operation during which a CUSP or PCF application program can run on ICSF without recompiling it. In this mode, ICSF cannot run simultaneously with CUSP or PCF.

**complementary keys**

A pair of keys that have the same clear key value, are different but complementary types, and usually exist on different systems.

**console**

A part of a computer used for communication between the operator or maintenance engineer and the computer. (A)

**control-area split**

In systems with VSAM, the movement of the contents of some of the control intervals in a control area to a newly created control area in order to facilitate insertion or lengthening of a data record when there are no remaining free control intervals in the original control area.

**control block**

A storage area used by a computer program to hold control information. (I) Synonymous with control area.

The circuitry that performs the control functions such as decoding microinstructions and generating the internal control signals that perform the operations requested. (A)

**control interval**

A fixed-length area of direct-access storage in which VSAM stores records and creates distributed free space. Also, in a key-sequenced data set or file, the set of records pointed to by an entry in the sequence-set index record. The control interval is the unit of information that VSAM transmits to or from direct access storage. A control interval always comprises an integral number of physical records.

**control interval split**

In systems with VSAM, the movement of some of the stored records in a control interval to a free control interval to facilitate insertion or lengthening of a record that does not fit in the original control interval.

**control statement input data set**

A key generator utility program data set containing control statements that a particular key generator utility program job will process.

**control statement output data set**

A key generator utility program data set containing control statements to create the complements of keys created by the key generator utility program.

**control vector**

In ICSF, a mask that is exclusive ORed with a master key or a transport key before ICSF uses that key to encrypt another key. Control vectors ensure that keys used on the system and keys distributed to other systems are used for only the cryptographic functions for which they were intended.

**CPACF**

CP Assist for Cryptographic Functions

**CP Assist for Cryptographic Functions**

Implemented on all IBM servers to provide AES and DES encryption and SHA-1 secure hashing.

**cross memory mode**

Synchronous communication between programs in different address spaces that permits a program residing in one address space to access the same or other address spaces. This synchronous transfer of control is accomplished by a calling linkage and a return linkage.

**CRT**

Chinese Remainder Theorem.

**Crypto Express2 Coprocessor**

An asynchronous cryptographic coprocessor available on the z9 EC, z9 BC, z10 EC and z10 BC.

**Crypto Express3 Coprocessor**

An asynchronous cryptographic coprocessor available on z10 EC, z10 BC, z114, z196, zEC12, and zBC12.

**Crypto Express4 Coprocessor**

An asynchronous cryptographic coprocessor available on zEC12 and zBC12.

**Crypto Express5 Coprocessor**

An asynchronous cryptographic coprocessor available on z13 and z13s.

**Crypto Express6 Coprocessor**

An asynchronous cryptographic coprocessor available on z14 and z14 ZR1.

**Crypto Express7 Coprocessor**

An asynchronous cryptographic coprocessor available on z15.

**cryptographic adapter (4764, 4765, and 4767)**

An expansion board that provides a comprehensive set of cryptographic functions for the network security processor and the workstation in the TSS family of products.



**cryptographic coprocessor**

A tamper responding, programmable, cryptographic PCI card, containing CPU, encryption hardware, RAM, persistent memory, hardware random number generator, time of day clock, infrastructure firmware, and software.

**cryptographic key data set (CKDS)**

A data set that contains the encrypting keys used by an installation.

In ICSF, a VSAM data set that contains all the cryptographic keys. Besides the encrypted key value, an entry in the cryptographic key data set contains information about the key.

**cryptography**

The transformation of data to conceal its meaning.

In computer security, the principles, means, and methods for encrypting plaintext and decrypting ciphertext.

In ICSF, the use of cryptography is extended to include the generation and verification of MACs, the generation of MDCs and other one-way hashes, the generation and verification of PINs, and the generation and verification of digital signatures.

**CUSP (Cryptographic Unit Support Program)**

The IBM cryptographic offering, program product 5740-XY6, using the channel-attached 3848. CUSP is no longer in service.

**CUSP/PCF conversion program**

A program, for use during migration from CUSP or PCF to ICSF, that converts a CUSP or PCF cryptographic key data set into a ICSF cryptographic key data set.

**Customer Information Control System (CICS)**

An IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user written application programs. It includes facilities for building, using, and maintaining databases.

**CVC**

Card verification code used by MasterCard.

**CVV**

Card verification value used by VISA.

**data encryption algorithm (DEA)**

In computer security, a 64-bit block cipher that uses a 64-bit key, of which 56 bits are used to control the cryptographic process and 8 bits are used for parity checking to ensure that the key is transmitted properly.

**data encryption standard (DES)**

In computer security, the National Institute of Standards and Technology (NIST) Data Encryption Standard, adopted by the U.S. government as Federal Information Processing Standard (FIPS) Publication 46, which allows only hardware implementations of the data encryption algorithm.

**data key or data-encrypting key**

A key used to encipher, decipher, or authenticate data.

In ICSF, a 64-bit encryption key used to protect data privacy using the DES algorithm. AES data keys are now supported by ICSF.

**data set**

The major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access.

**data-translation key**

A 64-bit key that protects data transmitted through intermediate systems when the originator and receiver do not share the same key.

**DEA**

Data encryption algorithm.

**decipher**

To convert enciphered data in order to restore the original data. (T)

In computer security, to convert ciphertext into plaintext by means of a cipher system.

To convert enciphered data into clear data. Contrast with encipher. Synonymous with decrypt.

**decode**

To convert data by reversing the effect of some previous encoding. (I) (A)

In ICSF, to decipher data by use of a clear key.

**decrypt**

See decipher.

**DES**

Data Encryption Standard.

**diagnostics data set**

A key generator utility program data set containing a copy of each input control statement followed by a diagnostic message generated for each control statement.

**digital signature**

In public key cryptography, information created by using a private key and verified by using a public key. A digital signature provides data integrity and source nonrepudiation.

**Digital Signature Algorithm (DSA)**

A public key algorithm for digital signature generation and verification used with the Digital Signature Standard.

**Digital Signature Standard (DSS)**

A standard describing the use of algorithms for digital signature purposes. One of the algorithms specified is DSA (Digital Signature Algorithm).

**Dilithium**

A quantum-safe cryptographic algorithm. Also known as LI2.

**DK**

*Die Deutsche Kreditwirtschaft* (German Banking Industry Committee). Formerly known as ZKA.

**domain**

That part of a network in which the data processing resources are under common control. (T)

In ICSF, an index into a set of master key registers.

**DSA**

Digital Signature Algorithm.

**DSS**

Digital Signature Standard.

**ECB**

Electronic codebook.

**ECC**

Elliptic Curve Cryptography.

**ECI**

Eurocheque International S.C., a financial institution consortium that has defined three PIN block formats.

**EID**

Environment Identification.

**electronic codebook (ECB) operation**

A mode of operation used with block cipher cryptographic algorithms in which plaintext or ciphertext is placed in the input to the algorithm and the result is contained in the output of the algorithm.

A mode of encryption using the data encryption algorithm, in which each block of data is enciphered or deciphered without an initial chaining vector. It is used for key management functions and the encode and decode callable services.

**electronic funds transfer system (EFTS)**

A computerized payment and withdrawal system used to transfer funds from one account to another and to obtain related financial data.

**encipher**

To scramble data or to convert data to a secret code that masks the meaning of the data to any unauthorized recipient. Synonymous with encrypt.

Contrast with decipher.

**enciphered data**

Data whose meaning is concealed from unauthorized users or observers.

**encode**

To convert data by the use of a code in such a manner that reconversion to the original form is possible. (T)

In computer security, to convert plaintext into an unintelligible form by means of a code system.

In ICSF, to encipher data by use of a clear key.

**encrypt**

See encipher.

**exit**

To execute an instruction within a portion of a computer program in order to terminate the execution of that portion. Such portions of computer programs include loops, subroutines, modules, and so on. (T)

In ICSF, a user-written routine that receives control from the system during a certain point in processing—for example, after an operator issues the START command.

**exportable form**

A condition a key is in when enciphered under an exporter key-encrypting key. In this form, a key can be sent outside the system to another system. A key in exportable form cannot be used in a cryptographic function.

**exporter key-encrypting key**

A 128-bit key used to protect keys sent to another system. A type of transport key.

**file**

A named set of records stored or processed as a unit. (T)

**GBP**

German Bank Pool.

**German Bank Pool (GBP)**

A German financial institution consortium that defines specific methods of PIN calculation.

**German Banking Industry Committee**

A hybrid term in English for *Die Deutsche Kreditwirtschaft*, also known as DK, an association of the German banking industry. Prior to August 2011, DK was named ZKA for *Zentraler Kreditausschuss*, or Central Credit Committee. ZKA was founded in 1932.

**hashing**

An operation that uses a one-way (irreversible) function on data, usually to reduce the length of the data and to provide a verifiable authentication value (checksum) for the hashed data.

**header record**

A record containing common, constant, or identifying information for a group of records that follows.

**ICSF**

Integrated Cryptographic Service Facility.

**importable form**

A condition a key is in when it is enciphered under an importer key-encrypting key. A key is received from another system in this form. A key in importable form cannot be used in a cryptographic function.

**importer key-encrypting key**

A 128-bit key used to protect keys received from another system. A type of transport key.

**initial chaining vector (ICV)**

A 64-bit random or pseudo-random value used in the cipher block chaining mode of encryption with the data encryption algorithm.

**initial program load (IPL)**

The initialization procedure that causes an operating system to commence operation.

The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction.

The process of loading system programs and preparing a system to run jobs.

**input PIN-encrypting key**

A 128-bit key used to protect a PIN block sent to another system or to translate a PIN block from one format to another.

**installation exit**

See exit.

**Integrated Cryptographic Service Facility (ICSF)**

A licensed program that runs under MVS/System Product 3.1.3, or higher, or OS/390 Release 1, or higher, or z/OS, and provides access to the hardware cryptographic feature for programming applications. The combination of the hardware cryptographic feature and ICSF provides secure high-speed cryptographic services.

**International Organization for Standardization**

An organization of national standards bodies from many countries, established to promote the development of standards to facilitate the international exchange of goods and services and to develop cooperation in intellectual, scientific, technological, and economic activity. ISO has defined certain standards relating to cryptography and has defined two PIN block formats.

**ISO**

International Organization for Standardization.

**job control language (JCL)**

A control language used to identify a job to an operating system and to describe the job's requirements.

**key-encrypting key (KEK)**

In computer security, a key used for encryption and decryption of other keys.

In ICSF, a master key or transport key.

**key generator utility program (KGUP)**

A program that processes control statements for generating and maintaining keys in the cryptographic key data set.

**key output data set**

A key generator utility program data set containing information about each key that the key generator utility program generates except an importer key for file encryption.

**key part**

A 32-digit hexadecimal value that you enter for ICSF to combine with other values to create a master key or clear key.

**key part register**

A register in a cryptographic coprocessor that accumulates key parts as they are entered via TKE.

**key store policy**

Ensures that only authorized users and jobs can access secure key tokens that are stored in one of the ICSF key stores - the CKDS or the PKDS.

**key store policy controls**

Resources that are defined in the XFACILIT class. A control can verify the caller has authority to use a secure token and identify the action to take when the secure token is not stored in the CKDS or PKDS.

**LI2**

Abbreviation for the Dilithium quantum-safe algorithm.

**linkage**

The coding that passes control and parameters between two routines.

**load module**

All or part of a computer program in a form suitable for loading into main storage for execution. A load module is usually the output of a linkage editor. (T)

**LPAR mode**

The central processor mode that enables the operator to allocate the hardware resources among several logical partitions.

**MAC generation key**

A 64-bit or 128-bit key used by a message originator to generate a message authentication code sent with the message to the message receiver.

**MAC verification key**

A 64-bit or 128-bit key used by a message receiver to verify a message authentication code received with a message.

**magnetic tape**

A tape with a magnetizable layer on which data can be stored. (T)

**master key**

In computer security, the top-level key in a hierarchy of key-encrypting keys.

ICSF uses master keys to encrypt operational keys. Master keys are known only to the cryptographic coprocessors and are maintained in tamper proof cryptographic coprocessors.

**master key concept**

The idea of using a single cryptographic key, the master key, to encrypt all other keys on the system.

**master key register**

A register in the cryptographic coprocessors that stores the master key that is active on the system.

**master key variant**

A key derived from the master key by use of a control vector. It is used to force separation by type of keys on the system.

**MD5**

Message Digest 5. A hash algorithm.

**message authentication code (MAC)**

The cryptographic result of block cipher operations on text or data using the cipher block chain (CBC) mode of operation.

In ICSF, a MAC is used to authenticate the source of the message, and verify that the message was not altered during transmission or storage.

**modification detection code (MDC)**

A 128-bit value that interrelates all bits of a data stream so that the modification of any bit in the data stream results in a new MDC.

In ICSF, an MDC is used to verify that a message or stored data has not been altered.

**multiple encipherment**

The method of encrypting a key under a double-length key-encrypting key.

**new master key register**

A register in a cryptographic coprocessor that stores a master key before you make it active on the system.

**NIST**

U.S. National Institute of Science and Technology.

**NOCV processing**

Process by which the key generator utility program or an application program encrypts a key under a transport key itself rather than a transport key variant.

**noncompatibility mode**

An ICSF method of operation during which CUSP or PCF can run independently and simultaneously on the same z/OS, OS/390, or MVS system. You cannot run a CUSP or PCF application program on ICSF in this mode.

**nonrepudiation**

A method of ensuring that a message was sent by the appropriate individual.

**OAEP**

Optimal asymmetric encryption padding.

**offset**

The process of exclusively ORing a counter to a key.

**old master key register**

A register in a cryptographic coprocessor that stores a master key that you replaced with a new master key.

**operational form**

The condition of a key when it is encrypted under the master key so that it is active on the system.

**output PIN-encrypting key**

A 128-bit key used to protect a PIN block received from another system or to translate a PIN block from one format to another.

**PAN**

Personal Account Number.

**parameter**

Data passed between programs or procedures.

**parmlib**

A system parameter library, either SYS1.PARMLIB or an installation-supplied library.

**partitioned data set (PDS)**

A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

**Personal Account Number (PAN)**

A Personal Account Number identifies an individual and relates that individual to an account at a financial institution. It consists of an issuer identification number, customer account number, and one check digit.

**personal identification number (PIN)**

The 4- to 12-digit number entered at an automatic teller machine to identify and validate the requester of an automatic teller machine service. Personal identification numbers are always enciphered at the device where they are entered, and are manipulated in a secure fashion.

**Personal Security card**

An ISO-standard "smart card" with a microprocessor that enables it to perform a variety of functions such as identifying and verifying users, and determining which functions each user can perform.

**PIN block**

A 64-bit block of data in a certain PIN block format. A PIN block contains both a PIN and other data.

**PIN generation key**

A 128-bit key used to generate PINs or PIN offsets algorithmically.

**PIN key**

A 128-bit key used in cryptographic functions to generate, transform, and verify the personal identification numbers.

**PIN offset**

For 3624, the difference between a customer-selected PIN and an institution-assigned PIN. For German Bank Pool, the difference between an institution PIN (generated with an institution PIN key) and a pool PIN (generated with a pool PIN key).

**PIN verification key**

A 128-bit key used to verify PINs algorithmically.

**PKA**

Public Key Algorithm.

**PKCS**

Public Key Cryptographic Standards (RSA Data Security, Inc.)

**PKDS**

Public key data set (PKA cryptographic key data set).

**plaintext**

Data in normal, readable form.

**primary space allocation**

An area of direct access storage space initially allocated to a particular data set or file when the data set or file is defined. See also secondary space allocation.

**private key**

In computer security, a key that is known only to the owner and used with a public key algorithm to decrypt data or generate digital signatures. The data is encrypted and the digital signature is verified using the related public key.

**processor complex**

A configuration that consists of all the machines required for operation.

**Processor Resource/Systems Manager**

Enables logical partitioning of the processor complex, may provide additional byte-multiplexer channel capability, and supports the VM/XA System Product enhancement for Multiple Preferred Guests.

**Programmed Cryptographic Facility (PCF)**

An IBM licensed program that provides facilities for enciphering and deciphering data and for creating, maintaining, and managing cryptographic keys.

The IBM cryptographic offering, program product 5740-XY5, using software only for encryption and decryption. This product is no longer in service; ICSF is the replacement product.

**PR/SM**

Processor Resource/Systems Manager.

**public key**

In computer security, a key made available to anyone who wants to encrypt information using the public key algorithm or verify a digital signature generated with the related private key. The encrypted data can be decrypted only by use of the related private key.

**public key algorithm (PKA)**

In computer security, an asymmetric cryptographic process in which a public key is used for encryption and digital signature verification and a private key is used for decryption and digital signature generation.

**public key cryptography**

In computer security, cryptography in which a public key is used for encryption and a private key is used for decryption. Synonymous with asymmetric cryptography.

**QSA**

Quantum-safe algorithm. Examples include CRYSTALS-Dilithium Digital Signature Algorithm.

**RACE Integrity Primitives Evaluatiuon Message Digest**

A hash algorithm.

**RCE redirection**

Ability for an existing CCA service to be used in such a way that a request is reformatted and routed to a Regional Cryptographic Server for processing. The RCE redirection function is available when the system has only an RCE device active or the user has READ access to CSF.RCE.FUNCTION.REDIRECT in the XFACILIT SAF resource class.

**RCS**

Regional Cryptographic Server.

**RDO**

Resource definition online.

**record chaining**

When there are multiple cipher requests and the output chaining vector (OCV) from the previous encipher request is used as the input chaining vector (ICV) for the next encipher request.

**Resource Access Control Facility (RACF)**

An IBM licensed program that provides for access control by identifying and verifying the users to the system, authorizing access to protected resources, logging the detected unauthorized attempts to enter the system, and logging the detected accesses to protected resources.

**retained key**

A private key that is generated and retained within the secure boundary of the Crypto Express adapter.

**return code**

A code used to influence the execution of succeeding instructions. (A)

A value returned to a program to indicate the results of an operation requested by that program.

**Rivest-Shamir-Adleman (RSA) algorithm**

A process for public key cryptography that was developed by R. Rivest, A. Shamir, and L. Adleman.

**RMF**

Resource Manager Interface.

**RMI**

Resource Measurement Facility.

**RSA**

Rivest-Shamir-Adleman.

**RSA-PSS**

RSA-Probabilistic Signature Scheme. RSA-PSS is a signature scheme that is based on the RSA cryptosystem and provides increased security assurance. It was added in version 2.1 of PKCS #1.

**SAF**

System Authorization Facility.

**save area**

Area of main storage in which contents of registers are saved. (A)

**secondary space allocation**

In systems with VSAM, area of direct access storage space allocated after primary space originally allocated is exhausted. See also primary space allocation.

**Secure Electronic Transaction**

A standard created by Visa International and MasterCard for safe-guarding payment card purchases made over open networks.

**secure key**

A key that is encrypted under a master key. When ICSF uses a secure key, it is passed to a cryptographic coprocessor where the coprocessor decrypts the key and performs the function. The secure key never appears in the clear outside of the cryptographic coprocessor.

**Secure Sockets Layer**

A security protocol that provides communications privacy over the Internet by allowing client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

**sequential data set**

A data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape.

**SET**

Secure Electronic Transaction.

**SHA (Secure Hash Algorithm, FIPS 180)**

(Secure Hash Algorithm, FIPS 180) The SHA (Secure Hash Algorithm) family is a set of related cryptographic hash functions designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST). The first member of the family, published in 1993, is officially called SHA. However, today, it is often unofficially called SHA-0 to avoid confusion with its successors. Two years later, SHA-1, the first successor to SHA, was published. Four more variants, have since been published with increased output ranges and a slightly different design: SHA-224, SHA-256, SHA-384, and SHA-512 (all are sometimes referred to as SHA-2).



**SHA-1 (Secure Hash Algorithm 1, FIPS 180)**

A hash algorithm required for use with the Digital Signature Standard.

**SHA-2 (Secure Hash Algorithm 2, FIPS 180)**

Four additional variants to the SHA family, with increased output ranges and a slightly different design: SHA-224, SHA-256, SHA-384, and SHA-512 (all are sometimes referred to as SHA-2).

**SHA-3 (Secure Hash Algorithm 3, FIPS 202)**

SHA-3 is a subset of the cryptographic primitive family Keccak and is used to build instances of Permutation-Based Hash and Extendable-Output Functions (see also SHAKE). Because of the successful attacks on MD5, SHA-0, and SHA-1, NIST perceived a need for an alternative, dissimilar cryptographic hash, which became SHA-3.

**SHA-224**

One of the SHA-2 algorithms.

**SHA-256**

One of the SHA-2 algorithms.

**SHA-384**

One of the SHA-2 algorithms.

**SHA-512**

One of the SHA-2 algorithms.

**SHA3-224**

An instance of the SHA-3 algorithm that provides a Permutation-Based Hash.

**SHA3-256**

An instance of the SHA-3 algorithm that provides a Permutation-Based Hash.

**SHA3-384**

An instance of the SHA-3 algorithm that provides a Permutation-Based Hash.

**SHA3-512**

An instance of the SHA-3 algorithm that provides a Permutation-Based Hash.

**SHAKE (combination of Secure Hash Algorithm and Keccak)**

A set of Extendable-Output Functions defined in FIPS PUB 202.

**SHAKE128**

An instance of the SHA-3 algorithm that provides an Extendable-Output Function.

**SHAKE256**

An instance of the SHA-3 algorithm that provides an Extendable-Output Function.

**smart card**

A plastic card that has a microchip capable of storing data or process information.

**special secure mode**

An alternative form of security that allows you to enter clear keys with the key generator utility program or generate clear PINs.

**SSL**

Secure Sockets Layer.

**supervisor state**

A state during which a processing unit can execute input/output and other privileged instructions.

**System Authorization Facility (SAF)**

An interface to a system security system like the Resource Access Control Facility (RACF).

**system key**

A key that ICSF creates and uses for internal processing.

**System Management Facility (SMF)**

A base component of z/OS that provides the means for gathering and recording information that can be used to evaluate system usage.

**TDEA**

Triple Data Encryption Algorithm.

**TKE**

Trusted key entry.

**Transaction Security System**

An IBM product offering including both hardware and supporting software that provides access control and basic cryptographic key-management functions in a network environment. In the workstation environment, this includes the 4755 Cryptographic Adapter, the Personal Security Card, the 4754 Security Interface Unit, the Signature Verification feature, the Workstation Security Services Program, and the AIX Security Services Program/6000. In the host environment, this includes the 4753 Network Security Processor and the 4753 Network Security Processor MVS Support Program.

**transport key**

A key used to protect keys distributed from one system to another. A transport key can be an AES or DES key-encrypting key (importer or exporter).

**transport key variant**

A key derived from a transport key by use of a control vector. It is used to force separation by type for keys sent between systems.

**TRUE**

Task-related User Exit (CICS). The CICS-ICSF Attachment Facility provides a CSFATRUE and CSFATREN routine.

**UAT**

UDX Authority Table.

**UDF**

User-defined function.

**UDK**

User-derived key.

**UDP**

User Developed Program.

**UDX**

User Defined Extension.

**verification pattern**

An 8-byte pattern that ICSF calculates from the key parts you enter when you enter a master key or clear key. You can use the verification pattern to verify that you have entered the key parts correctly and specified a certain type of key.

**Virtual Storage Access Method (VSAM)**

An access method for indexed or sequential processing of fixed and variable-length records on direct-access devices. The records in a VSAM data set or file can be organized in logical sequence by means of a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by means of relative-record number.

**Virtual Telecommunications Access Method (VTAM)**

An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

**VISA**

A financial institution consortium that has defined four PIN block formats and a method for PIN verification.

**VISA PIN Verification Value (VISA PVV)**

An input to the VISA PIN verification process that, in practice, works similarly to a PIN offset.

**3621**

A model of an IBM Automatic Teller Machine that has a defined PIN block format.

**3624**

A model of an IBM Automatic Teller Machine that has a defined PIN block format and methods of PIN calculation.

**4764**

The IBM 4764 PCI-X Cryptographic Coprocessor processor provides a secure programming and hardware environment where AES, DES, and RSA processes are performed.

**4765**

The IBM 4765 PCIe Cryptographic Coprocessor processor provides a secure programming and hardware environment where AES, DES, ECC, and RSA processes are performed.

**4767**

The IBM 4767 PCIe Cryptographic Coprocessor processor provides a secure programming and hardware environment where AES, DES, ECC, and RSA processes are performed.



---

# Index

## A

- accessibility
  - contact IBM [101](#)
  - features [101](#)
- additional manifest constants
  - Dilithium quantum-safe algorithms [30](#)
  - regional cryptographic servers [84](#)
- API examples
  - regional cryptographic servers [84](#)
- assistive technologies [101](#)
- auditing PKCS #11 functions [9](#)

## C

- C application program interface (API), using [19](#)
- CCA (Common Cryptographic Architecture) [1](#)
- CK\_RV CSN\_FindAllObjects() function [62](#)
- CKK\_IBM\_SM4
  - key type [77](#)
- CKM\_IBM\_ISO2\_SM4\_MAC [82](#)
- CKM\_IBM\_SM2 [79](#)
- CKM\_IBM\_SM2\_ENCRYPT [79](#)
- CKM\_IBM\_SM2\_KEY\_PAIR\_GEN [80](#)
- CKM\_IBM\_SM2\_SM3 [80](#)
- CKM\_IBM\_SM3 [81](#)
- CKM\_IBM\_SM4\_CBC [81](#)
- CKM\_IBM\_SM4\_ECB [81](#)
- CKM\_IBM\_SM4\_KEY\_GEN [83](#)
- CKM\_IBM\_SM4\_MAC [83](#)
- CKM\_IBM\_SM4\_MAC\_GENERAL [84](#)
- Common Cryptographic Architecture (CCA) [1](#)
- component trace entries for TKDS events [9](#)
- constants, manifest
  - where defined [19](#)
- contact
  - z/OS [101](#)
- crypto education [x](#)
- Cryptoki [1](#)
- CRYPTOZ class [3](#)
- CSFSERV class
  - resources for token services [4](#)

## D

- data object
  - attributes that ICSF supports [30](#)
- deleting token [19](#)
- Diffie-Hellman domain parameter object
  - attributes that ICSF supports [30](#)
- Diffie-Hellman private key object
  - attributes that ICSF supports [30](#)
- Diffie-Hellman public key object
  - attributes that ICSF supports [30](#)
- Dilithium quantum-safe algorithms
  - additional manifest constants [30](#)
- DLLs provided by ICSF [91](#)

- domain parameter object
  - attributes that ICSF supports [30](#)
- DSA domain parameter object
  - attributes that ICSF supports [30](#)
- DSA private key object
  - attributes that ICSF supports [30](#)
- DSA public key object
  - attributes that ICSF supports [30](#)
- dynamic link libraries (DLLs) provided by ICSF [91](#)

## E

- Elliptic Curve private key object
  - attributes that ICSF supports [30](#)
- Elliptic Curve public key object
  - attributes that ICSF supports [30](#)
- environment variables for tracing [71](#)

## F

- feedback [xi](#)
- FIPS 140-2
  - algorithms restricted when complying with [30](#)
  - operating in compliance with [10](#)
- function
  - non-standard PKCS #11 supported by ICSF [62](#)
  - standard PKCS #11 supported by ICSF [51](#)
- function return code
  - unique to z/OS, list of [70](#)

## H

- header file for C API [91](#)

## K

- key type
  - CKK\_IBM\_SM4 [77](#)
- key types supported [20](#)
- keyboard
  - navigation [101](#)
  - PF keys [101](#)
  - shortcut keys [101](#)

## L

- Language Environment [19](#)
- library
  - information that can be set and queried [50](#)

## M

- manifest constants
  - where defined [19](#)
- mechanism
  - information returned by C\_GetMechanismInfo [21](#)

mechanism (*continued*)  
which cryptographic hardware supports [26](#)

## N

navigation  
keyboard [101](#)

## O

object  
data  
attributes that ICSF supports [30](#)  
Diffie-Hellman domain parameter  
attributes that ICSF supports [30](#)  
Diffie-Hellman private key  
attributes that ICSF supports [30](#)  
Diffie-Hellman public key  
attributes that ICSF supports [30](#)  
domain parameter  
attributes that ICSF supports [30](#)  
DSA domain parameter  
attributes that ICSF supports [30](#)  
DSA private key  
attributes that ICSF supports [30](#)  
DSA public key  
attributes that ICSF supports [30](#)  
Elliptic Curve private key  
attributes that ICSF supports [30](#)  
Elliptic Curve public key  
attributes that ICSF supports [30](#)  
private key  
attributes that ICSF supports [30](#)  
RSA private key  
attributes that ICSF supports [30](#)  
RSA public key  
attributes that ICSF supports [30](#)  
secret key  
attributes that ICSF supports [30](#)  
SM2 private key  
attributes that ICSF supports [30](#)  
SM2 public key  
attributes that ICSF supports [30](#)  
X.509 certificate  
attributes that ICSF supports [30](#)  
object type  
supported by ICSF, list of [30](#)

## P

PIN [3](#)  
private key object  
attributes that ICSF supports [30](#)  
program, sample  
building and using [73](#)  
source code for [93](#)  
Public Key Cryptography Standards (PKCS) [1](#)  
public key object  
attributes that ICSF supports [30](#)

## R

regional cryptographic servers

regional cryptographic servers (*continued*)  
additional manifest constants [84](#)  
API examples [84](#)  
mechanism [77](#)  
XORing of a key and data [84](#)  
RSA private key object  
attributes that ICSF supports [30](#)  
RSA public key object  
attributes that ICSF supports [30](#)

## S

sample C program, PKCS #11  
building and using [73](#)  
sample program, testpkcs11  
source code for [93](#)  
secret key object  
attributes that ICSF supports [30](#)  
sending to IBM  
reader comments [xi](#)  
session object [10](#)  
shortcut keys [101](#)  
slot  
information that can be queried [50](#)  
slot ID [1](#)  
SM2 private key object  
attributes that ICSF supports [30](#)  
SM2 public key object  
attributes that ICSF supports [30](#)  
SM4 data encryption mechanism [82](#)  
SMF records written for PKCS #11 functions [9](#)  
SO R/W  
description [3](#)  
SO role [3](#)  
SRB mode [19](#)  
Strong SO  
description [3](#)

## T

tasks  
running the sample program  
steps [73](#)  
TCB mode [19](#)  
testpkcs11 program  
building and using [73](#)  
source code for [93](#)  
TKDS (token data set)  
description [2](#)  
token  
access levels [3](#)  
deleting [19](#)  
information that can be set and queried [50](#)  
managing, interfaces for [7](#)  
token data set (TKDS)  
description [2](#)  
token object [10](#)  
token, on z/OS  
description [1](#)  
rules for name [1](#)  
trace data [71](#)  
trace entries for TKDS events [9](#)  
trademarks [108](#)

troubleshooting applications [71](#)

## U

updatecomp program  
  building and using [73](#)  
user interface  
  ISPF [101](#)  
  TSO/E [101](#)  
User R/O  
  description [3](#)  
User R/W  
  description [3](#)  
User role [3](#)

## V

V2R2 changed information FMID HCR77C0 [xv](#)  
V2R2 deleted information FMID HCR77C0 [xv](#)  
V2R2 new information FMID HCR77C0 [xv](#)  
V2R3 changed information FMID HCR77C1 [xiv](#)  
V2R3 changed information FMID HCR77D0 [xiv](#)  
V2R3 deleted information FMID HCR77C1 [xiv](#)  
V2R3 deleted information FMID HCR77D0 [xiv](#)  
V2R3 new information FMID HCR77C1 [xiv](#)  
V2R4 changed information FMID HCR77D1 [xiii](#)  
V2R4 deleted information FMID HCR77D1 [xiii](#)  
V2R4 new information FMID HCR77D1 [xiii](#)

## W

Weak SO  
  description [3](#)  
Weak User  
  description [3](#)

## X

X.509 certificate object  
  attributes that ICSF supports [30](#)  
XORing of a key and data  
  regional cryptographic servers [84](#)

## Z

z/OS PKCS #11 token  
  deleting [19](#)  
  description [1](#)  
  rules for name [1](#)









SC14-7510-07

