

z/OS
3.2

*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 97.](#)

This edition applies to IBM® z/OS® 3.2 (5655-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2025-09-30

© **Copyright International Business Machines Corporation 2007, 2025.**

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

Contents

Tables.....	V
About this document.....	vii
Who should read this document.....	vii
How this document is organized.....	vii
How to use this document.....	vii
Where to find more information.....	viii
IBM Crypto education.....	viii
How to provide feedback to IBM.....	ix
Summary of changes.....	xi
Summary of changes for z/OS 3.2.....	xi
Changes made in Cryptographic Support for z/OS 3.1 (FMID HCR77E0).....	xii
Chapter 1. Overview of z/OS support for PKCS #11.....	1
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.....	10
Object types.....	10
Session objects.....	10
Token objects.....	11
Operating in compliance with FIPS 140.....	11
Requiring signature verification for ICSF module CSFINPV2.....	14
Requiring FIPS 140 algorithm checking from all z/OS PKCS #11 applications.....	15
Requiring FIPS 140-2 algorithm checking from select z/OS PKCS #11 applications.....	15
Requiring FIPS 140-3 algorithm checking from select z/OS PKCS #11 applications.....	16
Preparing to use PKCS #11 applications.....	17
Tasks for the system programmer.....	17
Tasks for the security administrator.....	18
Tasks for the auditor.....	18
Tasks for application programmers	18
Using SMF reporting for FIPS planning.....	18
Optional Crypto Express adapters.....	22
Chapter 2. The C API.....	23
Using the C API.....	23
Deleting z/OS PKCS #11 tokens.....	23
Environment.....	23
Cross memory considerations.....	24
Key types and mechanisms supported.....	25
Additional manifest constants for Dilithium quantum-safe algorithm support	37
Additional manifest constants for Kyber algorithm support.....	37

Objects and attributes supported.....	37
Library, slot, and token information.....	57
Functions supported.....	58
Standard functions supported	58
Non-standard functions supported	67
Non-standard mechanisms supported.....	67
Enterprise PKCS #11 coprocessors.....	68
Key algorithms/usages that are unsupported or disallowed by the Enterprise PKCS #11 coprocessors.....	68
PKCS #11 Coprocessor Access Control Points.....	69
Standard compliance modes.....	74
Considerations for migrating to session-bound objects and FIPS 2021/FIPS 2024.....	75
Function return codes.....	76
Troubleshooting PKCS #11 applications.....	78
Chapter 3. Sample PKCS #11 C programs	79
Running the pre-compiled version of testpkcs11	79
Steps for running the pre-compiled version of testpkcs11.....	80
Building sample PKCS #11 applications from source code.....	81
Chapter 4. ICSF PKCS #11 callable services.....	83
Appendix A. SMP/E installation data sets, directories, and files.....	85
Appendix B. Source code for the testpkcs11 sample program.....	87
Appendix C. Accessibility.....	95
Notices.....	97
Terms and conditions for product documentation.....	98
IBM Online Privacy Statement.....	99
Policy for unsupported hardware.....	99
Minimum supported hardware.....	99
Trademarks.....	100
Glossary.....	101
Index.....	115

Tables

1. Token access levels.....	3
2. Resources in the CSFSERV class for token services.....	4
3. CLEARKEY.token-name resource access and key security policy.....	6
4. Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO).....	26
5. Mechanisms supported by specific cryptographic hardware.....	31
6. Algorithms approved by ICSF FIPS 140-3 checking.....	32
7. CSFPPRF and CSFPPR2 RULE TLS-PRF input values allowed by ICSF FIPS 140-3 algorithm.....	35
8. Restricted algorithms and uses when running in compliance with FIPS 140-2.....	35
9. Common footnotes for object attribute tables.....	37
10. Data object attributes that ICSF supports.....	38
11. X.509 certificate object attributes that ICSF supports.....	39
12. Secret key object attributes that ICSF supports.....	41
13. Public key object attributes that ICSF supports.....	45
14. RSA public key object attributes that ICSF supports.....	48
15. DSA public key object attributes that ICSF supports.....	48
16. Diffie-Hellman public key object attributes that ICSF supports.....	49
17. Elliptic Curve public key object attributes that ICSF supports.....	49
18. Private key object attributes that ICSF supports.....	49
19. RSA private key object attributes that ICSF supports.....	53
20. DSA private key object attributes that ICSF supports.....	53
21. Diffie-Hellman private key object attributes that ICSF supports.....	54
22. Elliptic Curve private key object attributes that ICSF supports.....	54
23. Domain parameter object attributes that ICSF supports.....	55

24. DSA domain parameter object attributes that ICSF supports.....	55
25. Diffie-Hellman domain parameter object attributes that ICSF supports.....	55
26. Dilithium public key object attributes that ICSF supports.....	56
27. Dilithium private key object attributes that ICSF supports.....	56
28. Kyber public key object attributes that ICSF supports.....	56
29. Kyber private key object attributes that ICSF supports.....	57
30. Standard PKCS #11 functions that ICSF supports.....	58
31. List of algorithms/uses not supported/disallowed by Enterprise PKCS #11 coprocessors.....	68
32. PKCS #11 Access Control Points.....	70
33. Environment variables for capturing trace data.....	78

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 (or an equivalent product) 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 23 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 79 discusses how to build and run the testpkcs11 sample.
- Chapter 4, “ICSF PKCS #11 callable services,” on page 83 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 18 and the information that it references.

System programmers should read the section “[Tasks for the system programmer](#)” on page 17 and the information that it references.

Auditors should read the section “[Tasks for the auditor](#)” on page 18 and the information that is 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 \(www.oasis-open.org\)](http://www.oasis-open.org). 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

Detailed explanations and samples pertaining to IBM cryptographic technology are provided in IBM Crypto Education (community.ibm.com/community/user/ibmz-and-linuxone/groups/community-home?CommunityKey=6593e27b-caf6-4f6c-a8a8-10b62a02509c).

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. For more information, see [How to send feedback to IBM](#).

Summary of changes

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

Note: IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under [IBM z/OS Product Documentation Update Policy](http://www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy) (www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy).

Summary of changes for z/OS 3.2

The following content is new, changed, or no longer included in z/OS 3.2.

New

The following content is new.

September 2025 release

- [“Requiring FIPS 140-3 algorithm checking from select z/OS PKCS #11 applications” on page 16](#)
- [“Using SMF reporting for FIPS planning” on page 18](#)
- [“FIPS 140-3 mode programming” on page 24](#)

Changed

The following content is changed.

September 2025 release

- [“Controlling token access and key policy” on page 2](#)
- [“Auditing PKCS #11 functions” on page 9](#)
- [“Operating in compliance with FIPS 140” on page 11](#)
- [“Requiring FIPS 140 algorithm checking from all z/OS PKCS #11 applications” on page 15](#)
- [“Requiring FIPS 140-2 algorithm checking from select z/OS PKCS #11 applications” on page 15](#)
- [“Specifying FIPS 140-2 compliance from within a z/OS PKCS #11 application” on page 16](#)
- [“Tasks for the system programmer” on page 17](#)
- [“Tasks for the security administrator” on page 18](#)
- [“Tasks for application programmers ” on page 18](#)
- [“Optional Crypto Express adapters” on page 22](#)
- [“Environment” on page 23](#)
- [“Cross memory considerations” on page 24](#)
- [“Key types and mechanisms supported” on page 25](#)
- [“Objects and attributes supported” on page 37](#)
- [“Standard functions supported ” on page 58](#)
- [“Enterprise PKCS #11 coprocessors” on page 68](#)
- [“Key algorithms/usages that are unsupported or disallowed by the Enterprise PKCS #11 coprocessors” on page 68](#)
- [“Function return codes” on page 76](#)

Deleted

The following content is deleted.

September 2025 release

- None.

Changes made in Cryptographic Support for z/OS 3.1 (FMID HCR77E0)

The following changes are made for z/OS 3.1. The most recent updates are listed at the top of each section.

New

The following content is new.

June 2025 refresh

- Information about IBM z17.

March 2024 refresh

- Added the following for APAR OA65205, which also applies to z/OS V2R5 (ICSF FMID HCR77D2) and ICSF FMID HCR77D1:
 - [“Considerations for migrating to session-bound objects and FIPS 2021/FIPS 2024” on page 75.](#)
- New EP11 access control points: 0047, 0044, 0045, 0046, 0030, 0048, 0049, 004A, 004B.

Changed

The following content is changed.

March 2024 refresh

Updated the following for APAR OA65205, which also applies to z/OS V2R5 (ICSF FMID HCR77D2) and ICSF FMID HCR77D1:

- [“Operating in compliance with FIPS 140” on page 11.](#)
- [“Objects and attributes supported” on page 37.](#)
- [“PKCS #11 Coprocessor Access Control Points” on page 69.](#)
- [“Standard compliance modes” on page 74.](#)
- [Chapter 3, “Sample PKCS #11 C programs,” on page 79.](#)

September 2023 release

- [“Key types and mechanisms supported” on page 25](#) was updated.

Deleted

The following content was deleted.

September 2023 release

- None.

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 IBM Z 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 \(www.oasis-open.org\)](http://www.oasis-open.org). 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 23.

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 vendor-defined attributes), ICSF uses the context of the request along with a 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>			
CRYPTOZ resource	READ (SAF access level)	UPDATE (SAF access level)	CONTROL (SAF access level)
SO.token-name	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

Table 1. Token access levels (continued)			
CRYPTOZ resource	READ (SAF access level)	UPDATE (SAF access level)	CONTROL (SAF access level)
USER.token-name	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-name and USER.token-name 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-name and CLEARKEY.token-name.) Creating a generic profile that uses generic characters to match both the SO and USER portion of the resource name (for example *.token-name) 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.

Table 2. Resources in the CSFSERV class for token services		
Name of resource	Service	Called by
CSF1TRC	Token or object creation	C_InitToken, C_CreateObject, C_CopyObject
CSF1TRD	Token or object deletion	C_InitToken, C_DestroyObject
CSF1TRL	Token or object find	C_Initialize, C_FindObjects, CSN_FindAllObjects
CSF1SAV	Set object attributes	C_SetAttributeValue

<i>Table 2. Resources in the CSFSERV class for token services (continued)</i>		
Name of resource	Service	Called by
CSF1GAV	Get object attributes	C_GetAttributeValue
CSF1GSK	Generate secret key	C_GenerateKey
CSF1GKP	Generate key pair	C_GenerateKeyPair
CSF1PKS	Private key sign	C_Decrypt, C_DecryptUpdate, C_DecryptFinal, C_Sign, C_SignFinal
CSF1PKV	Public key verify	C_Encrypt, C_EncryptUpdate, C_EncryptFinal, C_Verify, C_VerifyFinal
CSF1SKD	Secret key decrypt	C_Decrypt, C_DecryptUpdate, C_DecryptFinal
CSF1SKE	Secret key encrypt	C_Encrypt, C_EncryptUpdate, C_EncryptFinal
CSFOWH	One-way hash	C_Digest, C_DigestUpdate, C_DigestFinal, C_Sign, C_SignUpdate, C_SignFinal, C_Verify, C_VerifyUpdate, C_VerifyFinal
CSF1WPK	Wrap key	C_WrapKey
CSF1UWK	Unwrap key	C_UnwrapKey
CSF1HMG	Generate MAC	C_Sign
CSF1HMV	Verify MAC	C_Verify
CSF1DVK	Derive key	C_DeriveKey
CSF1DMK	Derive multiple keys	C_DeriveKey
CSFIQA	PKCS #11 initialization	C_Initialize
CSFRNG	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”](#) on page 11 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.

When the CLEARKEY.token-name resource checking determines that a secure key should be created and the FIPSMODE ICSF option setting is 140-3,ENFORCE or 140-3,INDICATE, the request will fail with a return and reason code indicating the Enterprise PKCS #11 coprocessor cannot be used in the current FIPSMODE.

Table 3. CLEARKEY.token-name 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-name checking is always performed as described previously.
2. For create object, 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 because the sensitive key material has appeared in-the-clear outside the bounds of the secure coprocessor.
3. For copy object, no CLEARKEY.token-name checking is performed. By default, the target key's status as clear or encrypted is the same as the source 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-name 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-name 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 23](#).
- PKCS #11 specific ICSF callable services. The C API uses these callable services. For information about these callable services, see [Chapter 4, “ICSF PKCS #11 callable services,” on page 83](#).
- 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(ftpshr.ftp.server.pkcs11.token)
RACDCERT ADDTOKEN(webshr.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(ftpshr.ftp.server.pkcs11.token)
RACDCERT BIND(CERTAUTH LABEL('Local Root CA for Servers'))
TOKEN(webshr.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(ftpshr.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 CRYPTOZ resource, **CLEARKEY.token-name**, 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.
- ICSF writes SMF type 82 subtype 42 records when a PKCS #11 object lifecycle event occurs.
- ICSF writes SMF type 82 subtype 46 records when a PKCS #11 object key usage event occurs.

For descriptions of the SMF records that ICSF writes, see [z/OS Cryptographic Services ICSF System Programmer's Guide](#).

Component trace for PKCS #11 functions

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

- Type 16 (XCFTMSGGS) traces the broadcast of an XCF message related to TKDS I/O.
- Type 17 (XCFTMSGGR) 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 24.

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

The National Institute of Standards and Technology (NIST), the US federal technology agency that works with industry to develop and apply technology, has published the Federal Information Processing Standard Security Requirements for Cryptographic Modules standard (FIPS) that can be required by organizations that use cryptographic-based security systems to protect sensitive or valuable data. Applications that need to comply with the FIPS 140 standard can configure z/OS PKCS #11 clear key services in a way that allows only the use of cryptographic algorithms (including key sizes) approved by the standard.

The FIPSMODE option applies to requests that use PKCS #11 clear keys and does not apply to ICSF PKCS #11 secure key cryptographic requests processed by the Enterprise PKCS #11 cryptographic coprocessor (EP11). Secure key EP11 requests cannot be processed in FIPS 140-3 INDICATE and ENFORCE modes. Secure key EP11 requests are allowed in all other FIPSMODEs.

The FIPSMODE option has no effect on the compliance mode setting of the Enterprise PKCS #11 cryptographic coprocessor (EP11). For more information about compliance modes and the EP11 coprocessor, see [“Standard compliance modes”](#) on page 74.

The FIPSMODE option does not have any effect on the compliance mode of cryptographic coprocessors configured as CEXnC coprocessors. When the FIPSMODE is set to enforce PKCS #11 Services FIPS restrictions, ICSF applications using PKCS #11 services will not be directed to coprocessors except for the cases of non-PSS RSA signature services and Diffie-Hellman shared secret computation performed on coprocessors configured as a crypto express accelerator (CEXnA).

For more information, see:

- ‘Requiring signature verification for ICSF module CSFINPV2’ in ‘Steps for installation and initialization’ in [z/OS Cryptographic Services ICSF System Programmer's Guide](#).
- For information on how to request FIPS 140-3 return/reason code format from the ICSF's PKCS #11 C API, see [“FIPS 140-3 mode programming”](#) on page 24.

For the complete set of ICSF installation options, see [z/OS Cryptographic Services ICSF System Programmer's Guide](#).

The ICSF FIPSMODE installation option allows values for FIPS 140-3 and FIPS 140-2. The default FIPSMODE value is the FIPS 140-2 mode FIPSMODE(NO,FAIL(NO)). The FIPSMODE option indicates one of the following:

FIPSMODE(140-3,INDICATE,FAIL(fail-option) or 140-3,ENFORCE,FAIL(fail-option) or 140-3,HYBRID,FAIL(fail-option) or NO,FAIL(fail-option) or YES,FAIL(fail-option) or COMPAT,FAIL(fail-option))

FIPS 140-3 modes: FIPS 140-3 modes make use of return code 0 with non-zero reason codes for clear key requests. Installations should confirm that all applications using PKCS #11 services will function as required before updating the FIPSMODE to a FIPS 140-3 mode. To see how SMF recording can be used to determine how PKCS #11 services are being used, see [“Using SMF reporting for FIPS planning”](#) on page 18.

FIPSMODE(140-3,ENFORCE,fail-mode)

Indicates ICSF is to operate in *FIPS 140-3 ENFORCE MODE*. Successful requests meet ICSF FIPS 140-3 algorithm requirements. In this FIPSMODE,

- Cryptographic requests will be processed successfully when ICSF FIPS 140-3 algorithm approval checking passes. The return code will be zero and reason code will be X'CAA' (3241).
- Requests that do not pass ICSF FIPS 140-3 algorithm approval checking will not be processed. The return code will be 8 and the reason code will be:
 - X'CC6' (3270) when the algorithm is not allowed by ICSF FIPS 140-2 checking.
 - X'CC7' (3271) when the algorithm is not allowed by ICSF FIPS 140-3 checking, but was allowed by ICSF FIPS 140-2 checking.

FIPSMODE(140-3,INDICATE,fail-mode)

Indicates ICSF is to operate in *FIPS 140-3 INDICATE mode*. The clear key PKCS #11 application must check the service reason code to ensure that FIPS 140-3 was met. In this FIPSMODE,

- Cryptographic requests will be processed successfully when the algorithm used meets at least ICSF FIPS 140-2 algorithm checking. The return code will be 0 and the reason code will be:
 - X'CA9' (3240) when the algorithm is allowed by ICSF FIPS 140-2 checking, but not allowed by FIPS 140-3 checking.
 - X'CAA' (3241) when the algorithm is allowed by ICSF FIPS 140-3 and FIPS 140-2 checking.
- Requests that use algorithms not approved for FIPS 140-3 and FIPS 140-2 will not be processed. The return code will be 8 and the reason code will be X'CC6' (3270).

FIPSMODE(140-3, HYBRID,fail-mode)

Indicates ICSF PKCS #11 clear key services will operate in *FIPS 140-3 HYBRID mode*. In this mode, by default, applications operate in FIPS 140-3 INDICATE mode. Selected applications can operate in FIPS 140-2 COMPAT mode.

HYBRID mode is useful when applications cannot tolerate the non-zero reason returned in ENFORCE and INDICATE modes or when applications cannot meet ICSF FIPS requirements.

A resource profile in the CRYPTOZ class with the naming convention *USEFIPS140-2.token-name*, where *token-name* is the token name of the key object used by the application, can be created to so that the application will be subject to algorithm checking of FIPS 140-2 COMPAT mode. When the FIPSMODE is 140-3, HYBRID and the application is subject to FIPS 140-2 COMPAT mode algorithm checking, successful requests will return with return code zero and reason code zero.

To make an application operate in FIPS 140-2 COMPAT mode, create a resource profile in the CRYPT-TOZ class with the naming convention *USEFIPS140-2.token-name*, where *token-name* is the token name of the key object to be used by the application. Grant the user ID (or user ID of the task) that the request will run under READ access to the profile.

When the user ID has READ access to the *USEFIPS140-2.token-name* profile and the key object token name is the same as *token-name*, the request is processed in FIPS 140-2 COMPAT mode.

Applications running under COMPAT mode rules must meet FIPS 140-2 requirements. An application running under COMPAT mode can then also optionally be exempt from any FIPS enforcement. See the explanation for 'COMPAT' below:

Notes concerning FIPS 140-3 return and reason codes:

1. The following services do not use cryptographic algorithms and will always return with return code 0 and reason code 0 when the request was processed successfully:
 - PKCS #11 Get Attribute Value (CSFPGAV and CSFPGAV6).
 - PKCS #11 Set Attribute Value (CSFPSAV and CSFPSAV6).
 - PKCS #11 Token Record Create (CSFPTRC and CSFPTRC6).
 - PKCS #11 Token Record Delete (CSFPTRD and CSFPTRD6).
 - PKCS #11 Token Record List (CSFPTRL and CSFPTRL6).

2. The PKCS #11 Pseudo-Random Function (CSFPPRF) always returns with return code 0 and reason code 0 when rules FIPS and PRNGFIPS are used. C_GenerateRandom always returns with return code 0 and reason code 0 for successful requests. See the PKCS #11 Pseudo-Random Function (CSFPPRF and CSFPPRF6) service in *z/OS Cryptographic Services ICSF Application Programmer's Guide* and C_GenerateRandom in ["Standard functions supported "](#) on page 58 for more information.
3. See *FIPS 140-3 mode programming* in Chapter 2, Using the C API, for an explanation of how to request that the C API interface return the FIPS 140-3 indicator.
4. When the FIPSMODE option is 140-3,ENFORCE or 140-3,INDICATE, EP11 secure key requests cannot be processed. In these two modes, PKCS #11 service requests using secure keys fail with return code 8, reason code X'CE3' (3299) indicating that secure key requests are not allowed in the current FIPSMODE. All other modes allow the use of secure key requests. Successful secure key requests always return with return code 0 and reason code 0. There is no notion of the reason code serving as a FIPS level indicator for secure key requests regardless of the FIPSMODE.
5. In 140-3,INDICATE or 140-3,ENFORCE mode, when the CLEARKEY.token-name resource profile and the CKA_SENSITIVE key object attribute specify that a secure key be created, the request will fail with return code 8, reason code X'CE3' (3299) indicating that secure key requests are not allowed in the current FIPSMODE. For more information on using CLEARKEY profiles, see ["Controlling token access and key policy" on page 2](#).

FIPS 140-2 modes: In FIPS 140-2 modes, all successful requests return with return code zero and reason code 0.

FIPSMODE(YES,FAIL(fail-option))

Indicates ICSF PKCS #11 services will operate in *FIPS 140-2 ENFORCE mode*. Any application using PKCS #11 services must use only those algorithms allowed by ICSF FIPS 140-2 algorithm checking. Applications not allowed by ICSF FIPS 140-2 algorithm checking will fail.

FIPSMODE(COMPAT,FAIL(fail-option))

Indicates ICSF PKCS #11 services will operate in *FIPS 140-2 COMPAT mode*. By default, applications running in FIPS 140-2 COMPAT mode must meet the same FIPS 140-2 algorithm restrictions as YES mode. Optionally, an application can be made exempt from FIPS 140-2 algorithm checking.

To make an application exempt from FIPS 140-2 checking, create a resource profile in the CRYPTOZ class with the naming convention FIPSEXEMPT.token-name, where token-name is the token name of the key object to be used by the application. Grant READ access to the profile that the user ID (or user ID of the task) that the request will run under. The CKA_IBM_FIPS140 attribute must be OFF in the key object for the application to be exempt from checking.

The application will be exempt from ICSF FIPS 140-2 algorithm checking when all of the following are true:

- The CKA_IBM_FIPS140 attribute in the key object is OFF.
- The key object token name is the same as the token-name in a FIPSEXEMPT.token-name resource profile.
- The user ID (or user ID of the task) that the application runs under has READ access to the FIPSEXEMPT.token-name resource profile.

The application will be subject to ICSF FIPS 140-2 algorithm checking when any of the following are true:

- The CKA_IBM_FIPS140 attribute in the key object is ON.
- No FIPSEXEMPT.token-name profile exists for key object token name or the profile exists, but the user ID (or user ID of the task) that the application runs under has an access of NONE to profile.

FIPSMODE(NO,FAIL(fail-option))

Indicates ICSF PKCS #11 services will operate in *FIPS 140-2 on-demand* mode. This is the default mode. In this mode, applications are not required to meet the requirements of ICSF FIPS 140-2 algorithm checking unless the application explicitly requests that checking is done. FIPS 140-2 applications can explicitly request ICSF FIPS 140-2 checking by using a key object with the CKA_IBM_FIPS140 attribute set to ON.

FIPSEXEMPT.token-name profiles are not examined. Applications are not subject to FIPS 140-3 algorithm checking and the FIPS 140-3, HYBRID mode USEFIPS140-2.token-name profiles are not examined.

FAIL(fail-option)

ICSF will perform a series of cryptographic known answer tests at ICSF initialization as required by the FIPS 140 standards. If any of these initialization tests should fail, the action the ICSF initialization process takes will depend on the *fail-option* specified.

YES

Indicates ICSF is to terminate abnormally if there is a failure in any ICSF initialization known answer tests.

NO

Indicates ICSF initialization processing is to continue even if there is a failure in any of the initialization known answer tests. However, PKCS #11 support will be limited or nonexistent.

Requiring signature verification for ICSF module CSFINPV2

If your installation needs to operate z/OS PKCS #11 in compliance with the FIPS 140 standards, 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 algorithm checking from all z/OS PKCS #11 applications

In FIPS140-2 mode FIPSMODE(YES), all z/OS PKCS #11 clear key applications must meet the requirements of ICSF FIPS 140-2 algorithm approval checking. Only those applications that use algorithms and key sizes that pass ICSF FIPS 140-2 approval checking will succeed. Successful cryptographic service calls will return with return code zero and reason code zero.

In FIPSMODE FIPS140-3,ENFORCE, all z/OS PKCS #11 clear key applications must meet the requirements of ICSF FIPS 140-3 approved algorithm checking. Only those applications that use algorithms and key sizes that pass ICSF FIPS 140-3 approval checking will succeed. Successful cryptographic service calls will return with return code zero and a non-zero reason code indicating FIPS 140-3 algorithm approval checking passed.

In FIPS140-3 INDICATE mode, all z/OS PKCS #11 clear key applications must meet at least the requirements of ICSF FIPS 140-2 algorithm approval checking. Only those applications that use algorithms and key sizes that pass ICSF FIPS 140-2 algorithm checking or FIPS 140-3 checking will succeed. Successful cryptographic service calls will return a specific non-zero reason code depending on whether FIPS 140-3 or FIPS 140-2 algorithm checking passed.

For the list of algorithms and mechanisms that meet the requirements of ICSF FIPS 140-3 checking, see [Table 6 on page 32](#).

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

In FIPSMODE(COMPAT), 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 will not be subject to FIPS restrictions.

To specify which applications must comply with FIPS 140-2 restrictions and which applications do not need to comply, create FIPSEXEMPT.token-name resource profiles in the CRYPTOZ class. If no FIPSEXEMPT.token-name resource profiles are created, then all z/OS PKCS #11 applications will be subject to FIPS restrictions. By creating a FIPSEXEMPT.token-name resource profile for a particular token,

however, you can specify whether or not a particular user ID (or user ID of the task the request will run under) should be considered exempt from FIPS restrictions when using that token.

- If a user ID has access authority NONE to the `FIPSEXEMPT.token-name` resource, ICSF will enforce FIPS 140-2 compliance for that user ID.
- If a user ID has access authority READ to the `FIPSEXEMPT.token-name` 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:

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

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

2. Create the `FIPSEXEMPT.token-name` resource profile for each z/OS PKCS #11 token requiring exemption. All key objects associated with the token can be exempt. 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-name` or `USER.token-name` resources for token access and could have unintended consequences.

3. Using the PERMIT command, specify READ access authority for user IDs (or user ID of the task) 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)
```

Note: The `CKA_IBM_FIPS140` attribute must also be OFF in the key object for the application to be exempt from FIPS checking

4. Refresh the CRYPTOZ class in common storage:

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

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

When running in FIPS 140-2 mode `FIPSMODE COMPAT` or `FIPSMODE NO`, 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 8 on page 35](#)) will be enforced for the key regardless of any specifications made at the token level using `FIPSEXEMPT.token-name` resource profiles.

In FIPS 140-2 mode, `FIPSMODE YES` indicates FIPS 140-2 compliance is required by all applications, so setting the Boolean attribute `CKA_IBM_FIPS140` to TRUE is redundant and does not result in an error.

In FIPS 140-3 modes, the `CKA_IBM_FIPS140` attribute is ignored unless the `FIPSMODE` is set to `FIPS 140-3, HYBRID` and CRYPTOZ profiles have been set up so the application is running under `FIPSMODE COMPAT` rules.

Requiring FIPS 140-3 algorithm checking from select z/OS PKCS #11 applications

In `FIPSMODE(140-3, HYBRID)`, you can use resource profiles in the CRYPTOZ class to specify, at a token level, the applications that are exempt from FIPS 140-3 compliance and instead be subject to ICSF FIPS

140-2 COMPAT algorithm checking. In this mode, by default, all applications run in FIPSMODE 140-3 INDICATE mode.

To specify which applications must run with the FIPS 140-3 INDICATE requirements, and which do not, the security administrator must:

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

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

2. Create the USEFIP140-3.token-name resource profile for each z/OS PKCS #11 token requiring exemption from FIPS 140-3 checking. All key objects associated with the token can be subject to FIPS 140-2 COMPAT algorithm checking. The following command creates the profile for the omnipresent session-object token SYSTOK-SESSION-ONLY:

```
RDEF CRYPTOZ USEFIP140-2.SYSTOK-SESSION-ONLY UACC(NONE)
```

Although the use of generic profiles in the CRYPTOZ class is permitted, you should begin the profile name with “USEFIPS140-2”. Failure to do this could result in generic characters unintentionally matching the SO.token-name or USER.token-name resources for token access and could have unintended consequences.

3. Using the PERMIT command, specify READ access authority for user IDs (user IDs that the task will run under) that will run with FIPS 140-2 COMPAT rules and NONE access authority for user IDs that must comply with FIPS 140-3. The following command indicates that all user IDs are exempt, except for the daemon user ID BOGD.

```
PERMIT USEFIPS140-2.SYSTOK-SESSION-ONLY CLASS(CRYPTOZ) ID(*) ACC(READ)  
PERMIT USEFIPS140-2.SYSTOK-SESSION-ONLY CLASS(CRYPTOZ) ID(BOGD) ACC(NONE)
```

4. Refresh the CRYPTOZ class in common storage:

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

Note: In 140-3 modes, the FIPS 140-2 CKA_IBM_FIPS140 attribute has no effect on applications that do not have read access to the USEFIPS140-2 profile.

5. Those applications that have been chosen to run under FIPS 140-2,COMPAT mode can now optionally have their FIPS restrictions loosened further to NONE by using FIPSEXEMPT.token-name profiles.

Note: The CKA_IBM_FIPS140 attribute must be OFF in the key object for the FIPSEXEMPT profile to take effect.

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 standard, the system programmer must configure ICSF to run PKCS #11 services in compliance with FIPS 140. To do this, the system administrator uses the FIPSMODE option to specify the FIPS 140 mode required. For more information on the FIPSMODE option, refer to “Operating in compliance with FIPS 140” on page 11 and “Requiring signature verification for ICSF module CSFINPV2” in *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 79.

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 FIPS, 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 requirements. For more information, see [“Requiring signature verification for ICSF module CSFINPV2”](#) on page 14 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 \(www.oasis-open.org\)](#).

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 79.

Application programmers can use SMF records to observe the system's current use of PKCS #11 FIPS enforcement by ICSF. These records report on the requests that are required to meet FIPS checking and the highest level of FIPS checking they could meet. Before changing the FIPSMODE to a 140-3 mode, these records can be used to identify those applications that are requesting FIPS enforcement and if those applications will meet the requirements of FIPS 140-3 checking. For more information on the FIPSMODE option, see [“Operating in compliance with FIPS 140”](#) on page 11, [z/OS Cryptographic Services ICSF System Programmer's Guide](#), and the examples in [“Using SMF reporting for FIPS planning”](#) on page 18.

Using SMF reporting for FIPS planning

ICSF SMF record types 82 PKCS#11 key usage event (subtype 46) and key life cycle (subtype 42) contain a TLV (Tag-Length_Value) triplet with enhanced FIPS 140 audit information. The enhanced FIPS TLV indicates, for successful requests, the ICSF FIPSMODE, if the usage met the FIPSMODE level, and the maximum level of FIPS checking the usage can meet.

See the information for the tag SMF82_TAG_FIPS_INFO_V2 and the AUDITKEYLIFETKDS/AUDITPKCS11USG ICSF installation options in *z/OS Cryptographic Services ICSF System Programmer's Guide* and the CSFSMFR sample SMF record formatter.

The following are partial formatted examples of the type 82 subtype 46 SMF82_TAG_FIPS_INFO_V2 (535/x217) TLV. These TLVs are created for successful cryptographic requests unless the FIPSMODE option is specified or defaulted to NO and the key object has the CKA_IBM_FIPS140 attribute OFF.

The original FIPS information TLV (SMF82_TAG_FIPS_INFO) is only created for FIPSMODES NO, COMPAT, and YES. A flag in the TLV indicates that SMF82_TAG_FIPS_INFO_V2 was also created and this flag is always ON.

Example 1: FIPSMODE(140-3,ENFORCE) – AES 256 key

Bit 8 indicates ENFORCE is in effect and bits 16-23 indicate the mode is 140-3, so the FIPSMODE is 140-3,ENFORCE. The FIPSMODE requires usage pass FIPS 140-3 algorithm checking. Bit 4 is ON, and is always ON in this mode, because only algorithms that pass ICSF FIPS 140-3 algorithm checking are allowed.

```
SRV... CSF1SKE
...
KALG.. AES
KLEN.. 256
...
FIPSV2 28800101
  --80---- Bit 8: FIPSMODE(...,ENFORCE)
  ----01-- Bits 16-23: FIPSMODE(140-3,ENFORCE)
  20----- Bit 2: FIPS 140-3 enforced due ICSF FIPSMODE option
  08----- Bit 4: Algorithm is approved for current mode FIPS 140-3
  -----01
  Bits 24-31: Algorithm can be used in mode that requires FIPS 140-3 approval
```

Example 2: FIPSMODE(140-3,INDICATE) – RSA 1024 key

Bit 9 indicates INDICATE is in effect and bits 16-23 indicate the mode is 140-3, so the FIPSMODE is 140-3,INDICATE. The FIPSMODE requires usage pass FIPS 140-2 algorithm checking at least. Bit 4 is OFF because the usage did not pass ICSF algorithm checking for the current FIPSMODE, which is 140-3. Bits 24-31 indicate that the usage passed ICSF FIPS 140-2 algorithm checking. The use of this key in 140-3,ENFORCE mode would not be allowed.

```
SRV... CSF1PKV
...
KALG.. RSA
KLEN.. 1024
...
FIPSV2 20400100
  --40---- Bit 9: FIPSMODE(...,INDICATE)
  ----01-- Bits 16-23: FIPSMODE(140-3,INDICATE)
  20----- Bit 2: FIPS 140-3 enforced due ICSF FIPSMODE option
  00----- Bit 4: Algorithm is not approved for or not required to meet FIPS 140-3
  -----00 Bits 24-31: Algorithm can be used in mode that requires FIPS 140-2 approval
```

Example 3: FIPSMODE(140-3,HYBRID) - RSA 2048 key

Bit 10 indicates HYBRID is in effect and bits 16-23 indicate the mode is 140-3, so the FIPSMODE is 140-3,HYBRID. The FIPSMODE requires usage passes FIPS 140-2 algorithm checking at least. Bit 4 is ON because the usage passed ICSF algorithm checking for the current FIPSMODE, which is 140-3. Bit 6 is OFF because no USEFIPS140-2 profile applies to the applications usage (a profile that could apply in HYBRID). Bits 24-31 indicate the usage meets FIPS 140-3. The usage would be allowed also in FIPSMODE(140-3,ENFORCE) and FIPSMODE(YES).

```
SRV... CSF10WH
...
KALG.. RSA
KLEN.. 2048
...
FIPSV2 28200101
  --20---- Bit 10: FIPSMODE(...,HYBRID)
  ----01-- Bits 16-23: FIPSMODE(140-3,HYBRID)
  20----- Bit 2: FIPS 140-3 enforced due to ICSF FIPSMODE option
  08----- Bit 4: Algorithm is approved for current mode FIPS 140-3
  -----01 Bits 24-31: Algorithm can be used in mode that requires FIPS 140-3 approval
```

Example 4: FIPSMODE(140-3,HYBRID) - RSA 1024 key

Bit 10 indicates HYBRID is in effect and bits 16-23 indicate the mode is 140-3, so the FIPSMODE is 140-3,HYBRID. The FIPSMODE requires usage passes FIPS 140-2 algorithm checking at least. Bit 4 is OFF because the usage did not pass ICSF algorithm checking for the current FIPSMODE, which is 140-3. Bits 24-31 indicate the usage meets FIPS 140-2. The usage would be allowed in FIPSMODE(YES). The usage would not be allowed in FIPSMODE(140-3,ENFORCE).

```
SRV... CSF10WH
...
KALG.. RSA
KLEN.. 1024
...
FIPSV2 20200100
  --20---- Bit 10: FIPSMODE(...,HYBRID)
  ----01-- Bits 16-23: FIPSMODE(140-3,HYBRID)
  20----- Bit 2: FIPS 140-3 enforced due to ICSF FIPSMODE option
  00----- Bit 4: Algorithm is not approved for or not required to meet current mode
FIPS 140-3
  -----00 Bits 24-31: Algorithm can be used in mode that requires FIPS 140-2 approval
```

Example 5: FIPSMODE(140-3,HYBRID) - RSA 512 key

Bit 10 indicates HYBRID is in effect and bits 16-23 indicate the mode is 140-3, so the FIPSMODE is 140-3,HYBRID. The usage is not required to pass any level of ICSF FIPS checking because the user/token had access to CRYPTOZ profiles that allow all FIPS checking to be bypassed. Bit 4 is OFF because the usage does not require ICSF algorithm checking for the current FIPSMODE, which is 140-3. Bits 24-31 indicate relay that the usage does not pass any level of FIPS checking. The usage is only allowed when no FIPS checking is requested.

```
SRV... CSF10WH
...
KALG.. RSA
KLEN.. 512
...
FIPSV2 062001FF
  --20---- Bit 10: FIPSMODE(...,HYBRID)
  ----01-- Bits 16-23: FIPSMODE(140-3,HYBRID)
  00----- Bit 4: Algorithm is not approved for or not required to meet current mode
FIPS 140-3
  04----- Bit 5: FIPS checking is exempt due to FIPSEXEMPT.token-name profile
  02----- Bit 6: USEFIPS140-2.token-name profile caused FIPS 140-2 COMPAT rules to be
used
  -----FF Bits 24-31: Algorithm cannot be used in a mo Bit 10 indicates HYBRID is in
effect and bits 16-23 indicate the mode is 140-3, so the FIPSMODE is 140-3,HYBRID.
de that requires any FIPS 140 approval
```

Example 6: FIPSMODE(140-3,HYBRID) - RSA 2048 key

Bit 10 indicates HYBRID is in effect and bits 16-23 indicate the mode is 140-3, so the FIPSMODE is 140-3,HYBRID. The usage is not required to pass any level of ICSF FIPS checking because the user/token had access to CRYPTOZ profiles that allow all FIPS checking to be bypassed. Bit 4 is OFF because the usage does not require ICSF algorithm checking for the current FIPSMODE, which is 140-3. Bits 24-31 indicate the usage meets FIPS 140-3. The usage would be allowed in FIPSMODE(140-3,ENFORCE) and (140-3,YES).

```
SRV... CSF10WH
...
KALG.. RSA
KLEN.. 4096
...
FIPSV2 06200101
  --20---- Bit 10: FIPSMODE(...,HYBRID)
  ----01-- Bits 16-23: FIPSMODE(140-3,HYBRID)
  00----- Bit 4: Algorithm is not approved for or not required to meet current mode
FIPS 140-3
  04----- Bit 5: FIPS checking is exempt due to FIPSEXEMPT.token-name profile
  02----- Bit 6: USEFIPS140-2.token-name profile caused FIPS 140-2 COMPAT rules to be
used
  -----01 Bits 24-31: Algorithm can be used in mode that requires FIPS 140-3 approval
```


Example 7: FIPSMODE(YES) – RSA 1024 key

The TLV for SMF82_TAG_FIPS_INFO (270/X117) has a flag bit showing that the newer SMF82_TAG_FIPS_INFO_V2 has also been written.

In SMF82_TAG_FIPS_INFO_V2 TLV (FIPSV2), bit 0 indicates FIPSMODE(YES). Bits 16-23 indicate the mode is 140-2, which is redundant because YES only applies to 140-2. The FIPSMODE requires usage meet FIPS 140-2 algorithm checking. Bit 4 is on because the usage passed FIPS checking for the current FIPSMODE. Bits 24-31 indicate that the usage passes FIPS 140-2 checking. Use of this key in FIPS 140-3,ENFORCE mode will fail. Use of this key in FIPS 140-3,INDICATE mode will be allowed.

```
SRV... CSF1PKS
...
KALG.. RSA
KLEN.. 1024
...
FIPS.. AC000000
      80000000 FIPSMODE(YES)
      20000000 FIPS 140-2 enforced due to system settings
      08000000 Algorithm is approved for FIPS 140-2
      04000000 SMF82_TAG_FIPS_INFO_V2 is available

FIPSV2 B8000000
      80----- Bit 0: FIPSMODE(YES)
      ----00-- Bits 16-23: FIPSMODE(YES), FIPS 140-2
      20----- Bit 2: FIPS 140-2 enforced due ICSF FIPSMODE option
      10----- Bit 3: FIPS 140-2 enforced due to object FIPS attribute
      08----- Bit 4: Algorithm is approved for current mode FIPS 140-2
      -----00 Bits 24-31: Algorithm can be used in mode that requires FIPS 140-2 approval
```

Example 8: FIPSMODE(COMPAT) – 128 bit DES key

Bit 1 indicates FIPSMODE(COMPAT). Bits 16-23 indicate the mode is 140-2, which is redundant because COMPAT only applies to 140-2. The usage is not required to pass any level of ICSF FIPS checking because the user/token had access to a CRYPTOZ profile that allow all FIPS checking to be bypassed. Bit 4 is OFF because the usage did not require ICSF algorithm checking for the current FIPSMODE, which is 140-2.

Bits 24-31 indicate that the usage does not pass any FIPS checking.

This key cannot be used in FIPSMODE 140-3,ENFORCE or FIPSMODE 140-3,INDICATE.

This key could be used in FIPS140-3,HYBRID if the user/token was given read access to CRYPTOZ profiles:

1. A USEFIPS140-2.token-name profile would need to be created so FIPS 140-2 COMPAT mode is in effect for the usage.
2. The FIPSEXEMPT.token-name profile access must remain in place.

The original SMF82_TAG_FIPS_INFO TLV is not written because FIPS algorithm checking was not required.

```
SRV... CSF1SKE
...
KALG.. DES
KLEN.. 128
...
FIPSV2 440000FF
      40----- Bit 1: FIPSMODE(COMPAT)
      ----00-- Bits 16-23: FIPSMODE(COMPAT), FIPS 140-2
      00----- Bit 4: Algorithm is not approved for or not required to meet current mode
FIPS 140-2
      04----- Bit 5: FIPS checking exempt due to FIPSEXEMPT.token-name profile
      -----FF Bits 24-31: Algorithm cannot be used in a mode that requires any FIPS 140
approval
```

Example 9: FIPSMODE(NO)

In SMF82_TAG_FIPS_INFO_V2 TLV (FIPSV2), bits 0 and 1 indicate FIPSMODE(NO). Bits 16-23 indicate the mode is 140-2, which is redundant because NO only applies to 140-2. The key FIPS attribute (Bit 3) requires usage meet FIPS 140-2 algorithm checking. Bit 4 is ON because the passed ICSF algorithm checking for the current FIPSMODE, which is 140-2. The use of this key would

be allowed in FIPS 140-3 modes only if an allowed AES formatting mode was used such as AES GCMIVGEN.

```
KALG.. AES
KLEN.. 256
KUSGT. F9C00000
      00400000 CKA_IBM_FIPS140 attr ON

FIPS ...
FIPS.. 1C000000
      10000000 FIPS enforced at user request
      08000000 Algorithm is approved for FIPS 140-2
      04000000 SMF82_TAG_FIPS_INFO_V2 is available

FIPSV2 18000001
      00----- Bits 0-1: FIPSMODE(NO)
      ----00-- Bits 16-23: FIPSMODE(NO), FIPS 140-2
      10----- Bit 3: FIPS 140-2 enforced due to object FIPS attribute
      08----- Bit 4: Algorithm is approved for current mode FIPS 140-2
      -----01 Bits 24-31: Algorithm can be used in mode that requires FIPS 140-3 approval
```

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 standards are not directed to the CCA cryptographic coprocessors.

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.

FIPS 140-3 mode programming

When the ICSF FIPSMODE option specifies a 140-3 mode, ICSF callable services will return a non-zero reason code for return code zero cryptographic requests. To request FIPS 140-3 return/reason code format for successful C API clear key cryptographic requests, where the non-zero reason code is the FIPS 140-3 algorithm approval indicator, the CKF_IBM_FIPS_1403 flag 0x40000000 must be set on in CK_SESSION_INFO. The CKF_IBM_FIPS_1403 flag applies only when the ICSF FIPSMODE option is a FIPS 140-3 mode option.

When CKF_IBM_FIPS_1403 is ON and the cryptographic request is successful, the FIPS 140-3 indicator will be returned in the function return code. For example, the function return code for an ICSF Return code 0, Reason code X'CAA' meaning FIPS 140-3 algorithm checking passed is:

```
/* Request met requirements for ICSF FIPS 140-3 algorithm checking */  
#define CKR_IBM_ICSF_FIPS_140_3_MET      0xC0000CAA
```

When CKF_IBM_FIPS_1403 is OFF, successful cryptographic requests will always return with CKR_OK regardless of the FIPSMODE option setting.

The C_GenerateRandom function is the only cryptographic function that does not return the FIPS 140-3 indicator. Successful request to generate random data will always return with return code CKR_OK. C_GenerateRandom always calls the PKCS #11 CSFPRF service with rule PRNGFIPS. See C_GenerateRandom in [“Standard functions supported”](#) on page 58 for an explanation of when the random data returned meets FIPS 140-2 or FIPS140-3 requirements.

Successful non-cryptographic requests will always return CKR_OK. Therefore, regardless of the setting of CKF_IBM_FIPS_1403, the following non-cryptographic standard PKCS #11 functions listed in [Table 30](#) on page 58 will always return CKR_OK when successful:

- General purpose functions
- Slot and token management functions
- Session management functions
- Object management functions

In addition, functions that initialize an encryption operation, C_EncryptInit for example, are not cryptographic functions; therefore, will always return CKR_OK when successful.

For ICSF return and reason codes related to FIPS 140-3 modes, see [“Function return codes”](#) on page 76.

For details on the non-zero reason code values issued for successful cryptographic requests, see the FIPSMODE 140-3 options in [“Operating in compliance with FIPS 140”](#) on page 11.

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.

- FIPS140-2 COMPAT mode and FIPS 140-3, HYBRID modes behavior are also controlled by resources in the CRYPTOZ class.

Guideline: If the system is configured for FIPS140-2 COMPAT or 140-3, HYBRID mode and the PRIMARY address space function invoked by a PC is expected to adhere to the FIPS restrictions of the FIPSMODE installation setting, the user ID associated with the caller's unit of work should not be permitted to the CRYPTOZ FIPSEXEMPT and/or USEFIPS140-2 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_PQC_DILITHIUM - A quantum-safe key type.
- CKK_IBM_PQC_KYBER - 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 (CK_MECHANISM_TYPE)	Size factor	Flags
CKM_AES_CBC ³	Bytes	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_AES_CBC_PAD ³	Bytes	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP
CKM_AES_CTS ³	Bytes	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_AES_ECB ³	Bytes	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_AES_GCM ^{3, 7}	Bytes	CKF_ENCRYPT CKF_DECRYPT
CKM_AES_KEY_GEN	Bytes	[CKF_HW] CKF_GENERATE
CKM_BLOWFISH_CBC ^{4, 7}	Bytes	CKF_ENCRYPT CKF_DECRYPT
CKM_BLOWFISH_KEY_GEN ⁷	Bytes	[CKF_HW] CKF_GENERATE
CKM_DES_CBC ⁷	Not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_DES_CBC_PAD ⁷	Not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP
CKM_DES_ECB ⁷	Not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_DES_KEY_GEN ⁷	Not applicable	[CKF_HW] CKF_GENERATE
CKM_DES2_KEY_GEN ⁷	Not applicable	[CKF_HW] CKF_GENERATE
CKM_DES3_CBC ³	Not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT
CKM_DES3_CBC_PAD ³	Not applicable	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP
CKM_DES3_ECB ³	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

Table 4. Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO) (continued)		
Type (CK_MECHANISM_TYPE)	Size factor	Flags
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 ¹ CKF_EC_NAMEDCURVE ² CKF_EC_UNCOMPRESS
CKM_ECDH1_DERIVE	Bits	[CKF_HW] CKF_DERIVE CKF_EC_F_P ¹ CKF_EC_NAMEDCURVE ² CKF_EC_UNCOMPRESS
CKM_ECDSA	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY CKF_EC_F_P ¹ CKF_EC_NAMEDCURVE ² CKF_EC_UNCOMPRESS
CKM_ECDSA_SHA1	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY CKF_EC_F_P ¹ CKF_EC_NAMEDCURVE ² CKF_EC_UNCOMPRESS
CKM_GENERIC_SECRET_KEY_GEN ³	Bits	[CKF_HW] CKF_GENERATE
CKM_IBM_ATTRIBUTEBOUND_WRAP ⁸ (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_MD2	Not applicable	CKF_DIGEST
CKM_MD2_RSA_PKCS ^{5, 6}	Bits	CKF_SIGN CKF_VERIFY
CKM_MD5	Not applicable	CKF_DIGEST
CKM_MD5_HMAC	Not applicable	CKF_SIGN CKF_VERIFY
CKM_MD5_RSA_PKCS ^{5, 6}	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_PBE_SHA1_DES3_EDE_CBC	Not applicable	[CKF_HW] CKF_GENERATE
CKM_RC4 ^{4, 7}	Bits	CKF_ENCRYPT CKF_DECRYPT
CKM_RC4_KEY_GEN ⁷	Bits	[CKF_HW] CKF_GENERATE
CKM_RIPEMD160	Not applicable	CKF_DIGEST

Table 4. Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO) (continued)		
Type (CK_MECHANISM_TYPE)	Size factor	Flags
CKM_RSA_PKCS ^{5, 6}	Bits	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_WRAP CKF_UNWRAP CKF_SIGN CKF_VERIFY
CKM_RSA_PKCS_KEY_PAIR_GEN	Bits	[CKF_HW] CKF_GENERATE_KEY_PAIR
CKM_RSA_PKCS_PSS ⁹	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_RSA_X_509 ^{5, 6, 7}	Bits	[CKF_HW] CKF_ENCRYPT CKF_DECRYPT CKF_SIGN CKF_VERIFY
CKM_SHA_1	Not applicable	[CKF_HW] CKF_DIGEST
CKM_SHA_1_HMAC	Not applicable	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA1_RSA_PKCS ^{5, 6}	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA1_RSA_PKCS_PSS ⁹	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 ^{5, 6}	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA224_RSA_PKCS_PSS ⁹	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 ^{5, 6}	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA256_RSA_PKCS_PSS ⁹	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 ^{5, 6}	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA384_RSA_PKCS_PSS ⁹	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 ^{5, 6}	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SHA512_RSA_PKCS_PSS ⁹	Bits	[CKF_HW] CKF_SIGN CKF_VERIFY
CKM_SSL3_KEY_AND_MAC_DERIVE ⁷	Not applicable	CKF_DERIVE
CKM_SSL3_MASTER_KEY_DERIVE ⁷	Bytes	CKF_DERIVE
CKM_SSL3_MASTER_KEY_DERIVE_DH ⁷	Bytes	CKF_DERIVE
CKM_SSL3_MD5_MAC ⁷	Bits	CKF_SIGN CKF_VERIFY

Table 4. Mechanism information as returned by C_GetMechanismInfo (CK_MECHANISM_INFO) (continued)		
Type (CK_MECHANISM_TYPE)	Size factor	Flags
CKM_SSL3_PRE_MASTER_KEY_GEN ⁷	Bytes	[CKF_HW] CKF_GENERATE
CKM_SSL3_SHA1_MAC ⁷	Bits	CKF_SIGN CKF_VERIFY
CKM_TLS_KEY_AND_MAC_DERIVE ⁷	Not applicable	CKF_DERIVE
CKM_TLS_MASTER_KEY_DERIVE ⁷	Bytes	CKF_DERIVE
CKM_TLS_MASTER_KEY_DERIVE_DH ⁷	Bytes	CKF_DERIVE
CKM_TLS_PRE_MASTER_KEY_GEN ⁷	Bytes	[CKF_HW] CKF_GENERATE
CKM_TLS_PRF ⁷	Not applicable	CKF_DERIVE

Footnotes for Table 4 on page 26

¹ 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.

² 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 }

The following Koblitz named curves are supported:

- secp256k1 – { 1 3 132 0 10 }

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.

³ Mechanism not present on a system that is export controlled.

⁴ Mechanism limited to 56-bit on a system that is export controlled.

⁵ 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.

⁶ 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.

⁷ Mechanism supported for clear keys only.

⁸ Mechanism supported for secure keys only.

⁹ 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.

Table 5. Mechanisms supported by specific cryptographic hardware

Machine type and cryptographic hardware	Mechanisms supported	Notes
IBM z14 or IBM z14 ZR1 - CEX5P	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 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 CKM_PKCS5_PBKD2	This is the base set.
IBM z14 or IBM z14 ZR1 - CEX6P	IBM z14 and IBM z14R1 set and <ul style="list-style-type: none"> • ReencryptSingle function • CKM_IBM_ECDSA_OTHER 	CEX6P at level 3.6.16 is required for Reencrypt Single. CEX6P at level 3.6.19 is required for CKM_IBM_ECDSA_OTHER.
IBM z15 T01 or IBM z15 T02 - CEX6P	IBM z14 or IBM z14R1 set and <ul style="list-style-type: none"> • ReencryptSingle function • CKM_IBM_ECDSA_OTHER 	CEX6P at level 3.7.11 is required for ReencryptSingle. CEX6P at level 3.7.14 is required for CKM_IBM_ECDSA_OTHER.

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

Machine type and cryptographic hardware	Mechanisms supported	Notes
IBM z15 or IBM z15 TO2 - CEX7P	IBM z14 and IBM z14 ZR1 set and <ul style="list-style-type: none"> • CKM_ECDH1_DERIVE • CKK_IBM_PQC_DILITHIUM • CKA_IBM_PROTKEY_EXTRACTABLE • ReencryptSingle function • CKM_IBM_ECDSA_OTHER 	CEX7P at level 4.7.10 is required for CKM_ECDH1_DERIVE and CKA_IBM_PROTKEY_EXTRACTABLE. CEX7P at level 4.7.10 is required for CKK_IBM_PQC_DILITHIUM. CEX7P at level 4.7.21 is required for ReencryptSingle. CEX7P at level 4.7.24 is required for CKM_IBM_ECDSA_OTHER.
IBM z16 or IBM z16 AO2 - CEX8P	IBM z15 or z15 TO2 set and <ul style="list-style-type: none"> • CKM_IBM_KYBER 	
IBM z17	IBM z16 or z16 AO2 set	

The following table lists the algorithms that are approved by ICSF FIPS 140-3 algorithm checking. When the ICSF FIPMODE option is a 140-3 mode, the service return code will be zero and the reason code will indicate that a FIPS 140-3 approved algorithm was used. For additional details, see ‘FIPS 140-3 modes’ in [“Operating in compliance with FIPS 140”](#) on page 11.

Table 6. Algorithms approved by ICSF FIPS 140-3 checking

Algorithm usage	ICSF PKCS #11 C API mechanism	PKCS #11 Callable Service Support	Notes
AES Secret Key Generation	CKM_AES_KEY_GEN	CSFPGSK, mechanism rule ‘KEY’	Can only generate a Generic Secret key.
PBKDF2 Generic Secret Key Gen	CKM_PKCS5_PBKD2	CSFPGSK, mechanism rule ‘PBKDF2’	Must use a Generic Secret key. Iteration count 1000-750,000. HMAC alg 224,256,384, 512 Pw len 8-128 bytes. Salt len 16-128 bytes. Resulting key len 16-32 bytes.
AES data encryption/decryption	CKM_AES_CBC CKM_AES_CBC_PAD CKM_AES_ECB	CSFPSKE, CSFPSKD, Processing rules CBC-CS, CBC-PAD, ECB, CTR, GCMIVGEN, GCM(decryption only), GCMTLS13.	Rules allowed on PKCS #11 Callable service only CBC-CS. GCMIVGEN, GCM, GCMTLS13.
RSA keypair generation	CKM_RSA_PKCS_KEY_PAIR_GEN	CSFPGKP	Key sizes 2048, 3072, and 4096. Exponent X‘10001’.

Table 6. Algorithms approved by ICSF FIPS 140-3 checking (continued)

Algorithm usage	ICSF PKCS #11 C API mechanism	PKCS #11 Callable Service Support	Notes
RSA SigGen	CKM_RSA_PKCS	CSFPPKS, CSFPPS2	Key sizes 2048, 3072, and 4096. Exponent X'10001'.
ECDSA keypair generation	CKM_EC_KEY_PAIR_GEN	CSFPGKP	P-224, P-256, P-384, P-521.
DH keypair generation	CKM_DH_PKCS_KEY_PAIR_GEN	CSFPGKP	Safe primes ffdeh2048 and modp2048 only.
ECDSA SigGen	CKM_ECDSA	CSFPPKS	Nist curves P-224, P-256, P-384, P-521.
ECDSA SigVer	CKM_ECDSA	CSFPPKV	ECDSA Signature Verification primitive not allowed.
EC, RSA Hash & Sign	CKM_SHA224_RSA_PKCS CKM_SHA256_RSA_PKCS CKM_SHA384_RSA_PKCS CKM_SHA512_RSA_PKCS CKM_SHA224_RSA_PKCS_PSS CKM_SHA256_RSA_PKCS_PSS CKM_SHA384_RSA_PKCS_PSS CKM_SHA512_RSA_PKCS_PSS CKM_ECDSA_SHA224 CKM_ECDSA_SHA256 CKM_ECDSA_SHA384 CKM_ECDSA_SHA512	CSFPOWH	RSA Key sizes 2048, 3072, and 4096. Exponent X'10001' with SHA224, SHA256, SHA384, or SHA521. Nist curves P-224, P-256, P-384, P-521.

Table 6. Algorithms approved by ICSF FIPS 140-3 checking (continued)

Algorithm usage	ICSF PKCS #11 C API mechanism	PKCS #11 Callable Service Support	Notes
EC, RSA Hash & Verify	CKM_SHA224_RSA_PKCS CKM_SHA256_RSA_PKCS CKM_SHA384_RSA_PKCS CKM_SHA512_RSA_PKCS CKM_SHA224_RSA_PKCS_PSS CKM_SHA256_RSA_PKCS_PSS CKM_SHA384_RSA_PKCS_PSS CKM_SHA512_RSA_PKCS_PSS CKM_ECDSA_SHA224 CKM_ECDSA_SHA256 CKM_ECDSA_SHA384 CKM_ECDSA_SHA512	CSFPOWH	Key sizes ≥ 1024 including interim values, exponent X0, X3, X10001. Nist curves P-192, P-224, P-256, P-384, P-521. Must be combined with SHA224, SHA256, SHA384, or SHA512.
Hash only	CKM_SHA224 CKM_SHA256 CKM_SHA384 CKM_SHA512	CSFPOWH	SHA224, SHA256, SHA384, or SHA512.
EC and ECDH Key Derivation	CKM_DH_PKCS_DERIVE CKM_ECDH1_DERIVE	CSFPDVK	EC base key. Nist curves P-224, P-256, P-384, P-521. ECDH base key. Safe primes ffdeh2048 and modp2048 only. Derived key type must be a generic secret.
HMAC Generate/Verify	CKM_SHA224_HMAC CKM_SHA256_HMAC CKM_SHA384_HMAC CKM_SHA512_HMAC	CSFPHMG/CSFPHMV	SHA224, SHA256, SHA384, SHA512. Keylen $\geq \frac{1}{2}$ len of SHA hash output. Key must be a generic secret key. Key size must be ≥ 112 bits.

Table 6. Algorithms approved by ICSF FIPS 140-3 checking (continued)			
Algorithm usage	ICSF PKCS #11 C API mechanism	PKCS #11 Callable Service Support	Notes
hashDRBG	No input mechanism	CSFPPRF, mechanism rule PRNGFIPS	With rule PRNGFIPS, the FIPS 1403 indicator is return code 0 and reason code 0. See C_GenerateRandom in “Functions supported” on page 58 . The use of rule PRNG is never FIPS 140-3 approved.
TLS-PRF	CKM_TLS_PRF	CSFPPRF mechanism rule TLS-PRF	For TLS-PRF requirements, see Table 7 on page 35 .
TLS-PRF2	No C API interface/mechanism available	CSFPPR2 Rule TLS-PRF	For TLS-PRF requirements, see Table 7 on page 35 .
TLS-PRF and TLS-PRF2	No C API interface/mechanism available	CSFPPRF and CSFPPR2 Rule RULE_TLS13KDF	All input values allowed by the service are allowed by FIPS 140-3 algorithm checking.

Table 7. CSFPPRF and CSFPPR2 RULE TLS-PRF input values allowed by ICSF FIPS 140-3 algorithm	
Description	FIPS 140-3 requirements
Secret key source key	Must be specified. If a (CSFPPRF) <i>handle</i> , must be generic secret and non-blanks. If a (CSFPPR2) <i>key_structure</i> , must be non-zero length.
TLS-PRF PRF function code	Cannot be PRF function code X'00'.
TLS-PRF label	input <i>parms_list</i> label must be ASCII text for “extended master secret”, “key expansion”, “client finished”, or “server finished”.
Length of data to be generated	Must be non-zero.

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” on page 11](#).

Table 8. Restricted algorithms and uses when running in compliance with FIPS 140-2		
Algorithm	Mechanisms	Usage disallowed
AES GCM	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.
Blowfish	CKM_BLOWFISH_KEY_GEN, CKM_BLOWFISH_CBC	All

Table 8. Restricted algorithms and uses when running in compliance with FIPS 140-2 (continued)		
Algorithm	Mechanisms	Usage disallowed
ChaCha20-Poly1305	CKM_IBM_CHACHA20_POLY1305	All
Diffie Hellman	CKM_DH_PKCS_DERIVE	Prime size less than 1024 bits.
	CKM_DH_PKCS_PARAMETER_GEN	Prime sizes other than 1024 or 2048 bits.
DSA	CKM_DSA_SHA1, CKM_DSA	Prime sizes less than 1024 bits.
DSA	CKM_DSA_PARAMETER_GEN, CKM_DSA_KEY_PAIR_GEN or Sign	Combinations other than the following: <ul style="list-style-type: none"> • Prime size = 1024 bits, subprime size = 160 bits. • Prime size = 2048 bits, subprime size = 224 bits, or 256 bits.
EC Koblitz	CKM_ECDSA, CKM_ECDH1_DERIVE, CKM_EC_KEY_PAIR_GEN, CKM_ECDSA_SHA1	All
HMAC	CKM_SHA_1, CKM_SHA224, CKM_SHA256, CKM_SHA384, CKM_SHA512	Base key sizes less than one half the output size.
Kyber	N/A	All
MD2	CKM_MD2, CKM_MD2_RSA_PKCS	All
MD5	CKM_MD5, CKM_MD5_RSA_PKCS, CKM_MD5_HMAC	All
RC4	CKM_RC4	All
RIPEMD	CKM_RIPEMD160	All
RSA	CKM_RSA_X_509	All
RSA	CKM_RSA_PKCS	Key sizes less than 1024 bits.
RSA	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.
Single DES	CKM_DES_ECB, CKM_DES_CBC, CKM_DES_CBC_PAD	All
SSL3	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
TLS	CKM_TLS_MASTER_KEY_DERIVE, CKM_TLS_MASTER_KEY_DERIVE_DH, CKM_TLS_KEY_AND_MAC_DERIVE	Base key sizes less than 10 bytes.

Table 8. Restricted algorithms and uses when running in compliance with FIPS 140-2 (continued)		
Algorithm	Mechanisms	Usage disallowed
Triple DES	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_PQC_DILITHIUM      0x 80010023
#define CKA_IBM_DILITHIUM_MODE     0x 80000010
```

Additional manifest constants for Kyber algorithm support

```
#define CKK_IBM_PQC_KYBER          0x80010024
#define CKA_IBM_KYBER_MODE         0x8000000E
#define CKD_IBM_HYBRID_NULL        0x80000001
#define CKD_IBM_HYBRID_SHA1_KDF    0x80000002
#define CKD_IBM_HYBRID_SHA224_KDF  0x80000003
#define CKD_IBM_HYBRID_SHA256_KDF  0x80000004
#define CKD_IBM_HYBRID_SHA384_KDF  0x80000005
#define CKD_IBM_HYBRID_SHA512_KDF  0x80000006
#define CK_IBM_KEM_ENCAPSULATE     0x00000001
#define CK_IBM_KEM_DECAPSULATE     0x00000002
```

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), and CKK_DSA, CKK_DH.
- CKO_PRIVATE_KEY - CKK_RSA, CKK_EC (CKK_ECDSA), CKK_DSA, and CKK_DH.
- CKO_SECRET_KEY - CKK_DES, CKK_DES2, CKK_DES3, CKK_AES, CKK_BLOWFISH, CKK_RC4, CKK_GENERIC_SECRET, and CKK_IBM_CHACHA20.

The footnotes described in Table 9 on page 37 are taken from the PKCS #11 specification and apply to the attribute tables that follow.

Table 9. Common footnotes for object attribute tables	
Footnote number	Footnote meaning
1	Must be specified when object is created with C_CreateObject .
2	Must <i>not</i> be specified when object is created with C_CreateObject .
3	Must be specified when object is generated with C_GenerateKey or C_GenerateKeyPair .
4	Must <i>not</i> be specified when object is generated with C_GenerateKey or C_GenerateKeyPair .
5	Must be specified when object is unwrapped with C_UnwrapKey .

Table 9. Common footnotes for object attribute tables (continued)	
Footnote number	Footnote meaning
6	Must <i>not</i> be specified when object is unwrapped with C_UnwrapKey .
7	Cannot be revealed if object has its CKA_SENSITIVE attribute set to TRUE or its CKA_EXTRACTABLE attribute set to FALSE.
8	May be modified after object is created with a C_SetAttributeValue call, or in the process of copying object with a C_CopyObject 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 C_CopyObject 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 C_CopyObject call but not on a C_SetAttributeValue call
12	Attribute cannot be changed once set to CK_TRUE. It becomes a read only attribute.
13	May be changed to CK_TRUE 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 CKA_PRIVATE=TRUE , 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).
16	Attribute is read-only and may not be changed after the object is created.

Table 10. Data object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37 .		
Attribute	Data type	Notes
CKA_CLASS¹¹	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created only.
CKA_TOKEN¹¹	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 only.
CKA_PRIVATE¹¹	CK_BBOOL	Default value on create is TRUE. An application can specify the value when the object is created only.
CKA_MODIFIABLE¹¹	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created only.
CKA_LABEL	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.
CKA_ID	Byte array	Key or other identifier. Default is empty. An application can set or change the value at any time.

Table 10. Data object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37. (continued)

Attribute	Data type	Notes
CKA_VALUE	Byte array	Any value. Default is empty. An application can set or change the value at any time.
CKA_APPLICATION	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.
CKA_OBJECT_ID	Byte array	DER-encoded OID. Default is empty. An application can set or change the value at any time.
CKA_IBM_ICSF_HANDLE (vendor specific attribute - 0x80000009)	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 11. X.509 certificate object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.

Attribute	Data type	Notes
CKA_CLASS¹	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created only.
CKA_TOKEN¹¹	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 only.
CKA_PRIVATE¹¹	CK_BBOOL	Default value on create is FALSE. An application can specify the value when the object is created only.
CKA_MODIFIABLE¹¹	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created only.
CKA_LABEL	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.
CKA_CERTIFICATE_TYPE	CK_CERTIFICATE_TYPE	Always CKC_X_509. An application can specify the value when the object is created only.
CKA_TRUSTED	CK_BBOOL	Always set to TRUE. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_SUBJECT	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 only.

Table 11. X.509 certificate object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.
(continued)

Attribute	Data type	Notes
CKA_ID	Byte array	Key identifier. Default is empty. An application can set or change the value at any time.
CKA_ISSUER	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 only.
CKA_SERIAL_NUMBER	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 only.
CKA_VALUE	Byte array	This is the DER-encoding of the certificate. (Required.) An application can specify the value when the object is created only.
CKA_CERTIFICATE_CATEGORY	CK_ULONG	Categorization of the certificate: 1 Token user 2 Certificate authority 3 Other entity If not specified, ICSF sets it to 2 if the certificate has the BasicConstraints CA flag on. Otherwise it is not set. Note: If specified (or defaulted) to 2, the certificate is considered a CA certificate. The user must have appropriate authority. An application can set or change the value at any time.
CKA_APPLICATION	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 only.
CKA_IBM_DEFAULT (vendor specific attribute - 0x80000002)	CK_BBOOL	Default flag. Default is FALSE. An application can set or change the value at any time.
CKA_IBM_ICSF_HANDLE (vendor specific attribute - 0x80000009)	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. Secret key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.

Attribute	Data type	Notes
CKA_CLASS¹	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN¹¹	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.
CKA_PRIVATE¹¹	CK_BBOOL	Default value on create is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_MODIFIABLE¹¹	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	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.
CKA_ID	Byte array	Default is empty. An application can set or change the value at any time.
CKA_KEY_TYPE^{1, 5}	CK_KEY_TYPE	Type of key: CKK_AES, CKK_BLOWFISH, CKK_DES, CKK_DES2, CKK_DES3, CKK_GENERIC_SECRET, CKK_IBM_CHACHA20, or CKK_RC4. An application can specify the value when the object is created (or generated) only.
CKA_START_DATE⁸	CK_DATE	Start date for the key. Default is empty. An application can set or change the value at any time.
CKA_END_DATE⁸	CK_DATE	End date for the key. Default is empty. An application can set or change the value at any time.
CKA_DERIVE⁸	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.
CKA_LOCAL^{2, 4, 6}	CK_BBOOL	TRUE only if key was generated locally. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_GEN_MECHANISM^{2, 4, 6}	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.

Table 12. Secret key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.
(continued)

Attribute	Data type	Notes
CKA_ENCRYPT⁸	CK_BBOOL	TRUE if key supports encryption ⁹ . Default is FALSE for Generic secret keys. For all other key types, default is TRUE. An application can set or change the value at any time.
CKA_VERIFY⁸	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.
CKA_WRAP⁸	CK_BBOOL	TRUE if key supports wrapping (can be used to wrap other keys). ⁹ Default is FALSE for Generic secret keys. For all other key types, default is TRUE. An application can set or change the value at any time.
CKA_DECRYPT⁸	CK_BBOOL	TRUE if key supports decryption. ⁹ Default is FALSE for Generic secret keys. For all other key types, default is TRUE. An application can set or change the value at any time.
CKA_SIGN⁸	CK_BBOOL	TRUE if key supports signatures where the signature is an appendix to the data. ⁹ Default is TRUE. An application can set or change the value at any time.
CKA_UNWRAP⁸	CK_BBOOL	TRUE if key supports unwrapping (can be used to unwrap other keys) ⁹ . Default is FALSE for Generic secret keys. For all other key types, default is TRUE. An application can set or change the value at any time.
CKA_EXTRACTABLE⁸	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.
CKA_SENSITIVE⁸	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. Note: When CKA_IBM_SECURE is TRUE, CKA_SENSITIVE is set TRUE.
CKA_ALWAYS_SENSITIVE^{2, 4, 6}	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.

Table 12. Secret key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.
(continued)

Attribute	Data type	Notes
CKA_NEVER_EXTRACTABLE^{2, 4, 6}	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.
CKA_VALUE^{1, 4, 6, 7}	Byte array	The key. Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_VALUE_LEN^{2, 3}	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.
CKA_APPLICATION	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.
CKA_IBM_FIPS140 (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. Ignored when ICSF FIPS checking mode is a FIPS 140-3 mode.
CKA_TRUSTED^{10, 14}	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
CKA_WRAP_WITH_TRUSTED¹²	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
CKA_CHECK_VALUE	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.
CKA_IBM_SECURE^{6, 11, 12, 13} (vendor specific attribute - 0x80000006)	CK_BBOOL	CK_TRUE if the key is an Enterprise PKCS #11 coprocessor 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).

Table 12. Secret key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.
(continued)

Attribute	Data type	Notes
CKA_IBM_ALWAYS_SECURE^{2, 4, 6} (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.
CKA_IBM_CARD_COMPLIANCE^{2, 4, 6, 15} (vendor specific attribute - 0x80000007)	CK_ULONG	A bit mask field indicating the Enterprise PKCS #11 coprocessor compliance mode of the secure key: FIPS2009 X'00000001' BSI2009 X'00000002' FIPS2011 X'00000004' BSI2011 X'00000008' BSICC2017 X'00000040' FIPS2021 X'00000080' FIPS2024 X'00000100' 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.
CKA_IBM_SESSION_BOUND⁶ (vendor specific attribute - 0x80000040)	CK_BBOOL	CK_TRUE if the key is a session-bound Enterprise PKCS #11 coprocessor secure key. If a key is session-bound, it may only be used on an Enterprise PKCS #11 coprocessor which supports session-bound keys. This attribute may only be requested during C_CreateObject or C_GenerateKey. The default depends on the configuration of EP11 coprocessors available. The default is CK_TRUE when any of the active EP11 coprocessors require session-bound objects. Otherwise, the default is CK_FALSE.
CKA_IBM_ATTRBOUND^{2, 6} (vendor specific attribute - 0x80010004)	CK_BBOOL	CK_TRUE has the following meaning: <ul style="list-style-type: none"> • The key must only be exported with its boolean usage attributes. • The key may be used as a signing or verification key for attribute bound wrap/unwrap. • The key may be used as a wrapping or unwrapping key for attribute bound wrap/unwrap. • The key may not be used as a wrapping key for non-attribute bound wrap. 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.

Table 12. Secret key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.
(continued)

Attribute	Data type	Notes
CKA_IBM_ICSF_HANDLE (vendor specific attribute - 0x80000009)	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.
CKA_IBM_PROTKEY_EXTRACTABLE¹¹ (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.
CKA_IBM_PROTKEY_NEVER_EXTRACTABLE^{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.
CKA_IBM_IV_DATA^{4, 6, 16} (vendor specific attribute - 0x8000000A)	Byte array	AES GCM data for a specific protocol. Byte array defined as: Byte(s) Meaning when set 1-4 4-character protocol ID. Allowable EBCDIC character protocol ids and associated data: 'GT13' Data is for use with the GCMTLS13 rule and AES GCM encryption/decryption. Byte(s) Meaning when set 5-24 12 byte TLS 1.3 nonce followed by 8 byte hexadecimal sequence number. Sequence number must be hex zero when the object is created.

Table 13. Public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.

Attribute	Data type	Notes
CKA_CLASS¹	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.

Table 13. Public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.
(continued)

Attribute	Data type	Notes
CKA_TOKEN¹¹	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.
CKA_PRIVATE¹¹	CK_BBOOL	Default value on create is FALSE. An application can specify the value when the object is created (or generated) only.
CKA_MODIFIABLE¹¹	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	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.
CKA_TRUSTED^{10, 14}	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
CKA_SUBJECT	Byte array	DER-encoding. Default empty. An application can set or change the value at any time.
CKA_ID	Byte array	Key identifier. Default empty. An application can set or change the value at any time.
CKA_KEY_TYPE^{1, 5}	CK_KEY_TYPE	Type of key. CKK_DH, CKK_DSA, CKK_EC, CKK_RSA, CKK_IBM_PQC_DILITHIUM, and CKK_IBM_PQC_KYBER only. An application can specify the value when the object is created (or generated) only.
CKA_START_DATE⁸	CK_DATE	Start date for the key. Default empty. An application can set or change the value at any time.
CKA_END_DATE⁸	CK_DATE	End date for the key. Default empty. An application can set or change the value at any time.
CKA_DERIVE^{8, 9}	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.
CKA_LOCAL^{2, 4, 6}	CK_BBOOL	TRUE only if key was generated locally. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_KEY_GEN_MECHANISM^{2, 4, 6}	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.

Table 13. Public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.
(continued)

Attribute	Data type	Notes
CKA_ENCRYPT^{8, 9}	CK_BBOOL	TRUE if key supports encryption. Default is TRUE. An application can set or change the value at any time.
CKA_VERIFY^{8, 9}	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. Not modifiable if CKA_IBM_SECURE is TRUE.
CKA_VERIFY_RECOVER	CK_BBOOL	Always FALSE. Attribute is ignored when specified as TRUE.
CKA_WRAP^{8, 9}	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.
CKA_APPLICATION	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.
CKA_IBM_FIPS¹⁴⁰ (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. Ignored when ICSF FIPS checking mode is a FIPS 140-3 mode.
CKA_IBM_SECURE^{11, 12, 13} (vendor specific attribute - 0x80000006)	CK_BBOOL	CK_TRUE if the key is an Enterprise PKCS #11 coprocessor 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.
CKA_IBM_CARD COMPLIANCE^{2, 4, 6, 15} (vendor specific attribute - 0x80000007)	CK_ULONG	A bit mask field indicating the Enterprise PKCS #11 coprocessor compliance mode of the secure key: FIPS2009 X'00000001' BSI2009 X'00000002' FIPS2011 X'00000004' BSI2011 X'00000008' BSICC2017 X'00000040' FIPS2021 X'00000080' FIPS2024 X'00000100' 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.

Table 13. Public key object attributes that ICSF supports. For the meanings of the footnotes, see [Table 9 on page 37](#). (continued)

Attribute	Data type	Notes
CKA_IBM_SESSION_BOUND 6 (vendor specific attribute - 0x80000040)	CK_BBOOL	<p>CK_TRUE if the key is a session-bound Enterprise PKCS #11 coprocessor secure key. If a key is session-bound, it may only be used on an Enterprise PKCS #11 coprocessor which supports session-bound keys.</p> <p>This attribute may only be requested during C_CreateObject or C_GenerateKeyPair.</p> <p>The default depends on the configuration of EP11 coprocessors available. The default is CK_TRUE when any of the active EP11 coprocessors require session-bound objects. Otherwise, the default is CK_FALSE.</p>
CKA_IBM_ATTRBOUND 11, 12, 13 (vendor specific attribute - 0x80010004)	CK_BBOOL	<p>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.</p> <p>An attribute bound key may be requested by setting this attribute CK_TRUE during key creation or generation.</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>
CKA_IBM_ICSF_HANDLE (vendor specific attribute - 0x80000009)	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 14. RSA public key object attributes that ICSF supports. For the meanings of the footnotes, see [Table 9 on page 37](#).

Attribute	Data type	Notes
CKA_MODULUS 1, 4	Big integer	<p>Modulus n</p> <p>An application can specify the value when the object is created (or generated) only.</p>
CKA_MODULUS_BITS 2, 3	CK_ULONG	<p>Length in bits of modulus n</p> <p>An application can specify the value when the object is created (or generated) only.</p>
CKA_PUBLIC_EXPONENT 1	Big integer	<p>Public exponent e</p> <p>An application can specify the value when the object is created (or generated) only.</p>

Table 15. DSA public key object attributes that ICSF supports. For the meanings of the footnotes, see [Table 9 on page 37](#).

Attribute	Data type	Notes
CKA_PRIME 1, 3	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)

Table 15. DSA public key object attributes that ICSF supports. For the meanings of the footnotes, see [Table 9 on page 37](#). (continued)

Attribute	Data type	Notes
CKA_SUBPRIME^{1, 3}	Big integer	Subprime q (160 bits for $p \leq 1024$ bits, 224 bits or 256 bits for $p > 1024$ bits)
CKA_BASE^{1, 3}	Big integer	Base g
CKA_VALUE^{1, 4}	Big integer	Public value y

Table 16. Diffie-Hellman public key object attributes that ICSF supports. For the meanings of the footnotes, see [Table 9 on page 37](#).

Attribute	Data type	Notes
CKA_PRIME^{1, 3}	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_BASE^{1, 3}	Big integer	Base g
CKA_VALUE^{1, 4}	Big integer	Public value y

Table 17. Elliptic Curve public key object attributes that ICSF supports. For the meanings of the footnotes, see [Table 9 on page 37](#).

Attribute	Data type	Notes
CKA_EC_PARAMS^{1, 3} (CKA_ECDSA_PARAMS)	Byte array	DER-encoding of an ANSI X9.62 Parameters value
CKA_EC_POINT^{1, 4}	Byte array	DER-encoding of an ANSI X9.62 ECPoint value Q

Table 18. Private key object attributes that ICSF supports. For the meanings of the footnotes, see [Table 9 on page 37](#).

Attribute	Data type	Notes
CKA_CLASS¹	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN¹¹	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.
CKA_PRIVATE¹¹	CK_BBOOL	Default value on create is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_MODIFIABLE¹¹	CK_BBOOL	Default value is TRUE. An application can specify the value when the object is created (or generated) only.
CKA_LABEL	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.
CKA_SUBJECT	Byte array	DER-encoding. An application can set or change the value at any time.

Table 18. Private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.
(continued)

Attribute	Data type	Notes
CKA_ID	Byte array	Default is empty. An application can set or change the value at any time.
CKA_KEY_TYPE^{1, 5}	CK_KEY_TYPE	Type of key. CKK_EC, CKK_DH, CKK_DSA, CKK_RSA, CKK_IBM_PQC_DILITHIUM, and CKK_IBM_PQC_KYBER only. An application can specify the value when the object is created (or generated) only.
CKA_START_DATE⁸	CK_DATE	Start date for the key. Default empty. An application can set or change the value at any time.
CKA_END_DATE⁸	CK_DATE	End date for the key. Default empty. An application can set or change the value at any time.
CKA_DERIVE^{8, 9}	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.
CKA_LOCAL^{2, 4, 6}	CK_BBOOL	TRUE only if key was generated locally. Implicitly set by ICSF. An application cannot directly manipulate this value, but can view it.
CKA_KEY_GEN_MECHANISM^{2, 4, 6}	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.
CKA_DECRYPT^{8, 9}	CK_BBOOL	TRUE if key supports decryption. Default is TRUE. An application can set or change the value at any time.
CKA_SIGN^{8, 9}	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.
CKA_SIGN_RECOVER	CK_BBOOL	Always FALSE. Attribute is ignored when specified as TRUE.
CKA_UNWRAP^{8, 9}	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.
CKA_EXTRACTABLE⁸	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.
CKA_SENSITIVE⁸	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. Note: When CKA_IBM_SECURE is TRUE, CKA_SENSITIVE is set TRUE.

Table 18. Private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.
(continued)

Attribute	Data type	Notes
CKA_ALWAYS_SENSITIVE^{2, 4, 6}	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.
CKA_NEVER_EXTRACTABLE^{2, 4, 6}	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.
CKA_APPLICATION	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.
CKA_IBM_FIPS140 (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. Ignored when ICSF FIPS checking mode is a FIPS 140-3 mode.
CKA_WRAP_WITH_TRUSTED¹²	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
CKA_IBM_SECURE^{6, 11, 12, 13} (vendor specific attribute - 0x80000006)	CK_BBOOL	CK_TRUE if the key is an Enterprise PKCS #11 coprocessor 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_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. When this is set TRUE by the caller or ICSF, the key is treated as sensitive (CKA_SENSITIVE is set TRUE). May be changed from FALSE to TRUE during C_CopyObject.
CKA_IBM_ALWAYS_SECURE^{2, 4, 6} (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 18. Private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.
(continued)

Attribute	Data type	Notes
CKA_IBM_CARD_COMPLIANCE^{2,4,6,15} (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>FIPS2009 X'00000001' BSI2009 X'00000002' FIPS2011 X'00000004' BSI2011 X'00000008' BSICC2017 X'00000040' FIPS2021 X'00000080' FIPS2024 X'00000100'</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>
CKA_IBM_SESSION_BOUND 6 (vendor specific attribute - 0x80000040)	CK_BBOOL	<p>CK_TRUE if the key is a session-bound Enterprise PKCS #11 coprocessor secure key. If a key is session-bound, it may only be used on an Enterprise PKCS #11 coprocessor which supports session-bound keys.</p> <p>This attribute may only be requested during C_CreateObject or C_GenerateKeyPair.</p> <p>The default depends on the configuration of EP11 coprocessors available. The default is CK_TRUE when any of the active EP11 coprocessors require session-bound objects. Otherwise, the default is CK_FALSE.</p>
CKA_IBM_ATTRBOUND^{2,6} (vendor specific attribute - 0x80010004)	CK_BBOOL	<p>CK_TRUE has the following meaning:</p> <ul style="list-style-type: none"> • The key must only be exported with its boolean usage attributes. • The key may be used as a signing key for attribute bound wrap. • The key may be used as an unwrapping key for attribute bound unwrap. <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>
CKA_IBM_ICSF_HANDLE (vendor specific attribute - 0x80000009)	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 19. RSA private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.

Attribute	Data type	Notes
CKA_MODULUS ^{1, 4, 6}	Big integer	Modulus n An application can specify the value when the object is created (or generated) only.
CKA_PUBLIC_EXPONENT ^{4, 6}	Big integer	Public exponent e An application can specify the value when the object is created (or generated) only.
CKA_PRIVATE_EXPONENT ^{1, 4, 6, 7}	Big integer	Private exponent d Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_PRIME_1 ^{4, 6, 7}	Big integer	Prime p Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_PRIME_2 ^{4, 6, 7}	Big integer	Prime q Sensitive key part. An application can specify the value when the object is created (or generated) only.
CKA_EXPONENT_1 ^{4, 6, 7}	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.
CKA_EXPONENT_2 ^{4, 6, 7}	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.
CKA_COEFFICIENT ^{4, 6, 7}	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 20. DSA private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.

Attribute	Data type	Notes
CKA_PRIME ^{1, 4, 6}	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_SUBPRIME ^{1, 4, 6}	Big integer	Subprime q (160 bits for $p \leq 1024$ bits, 224 bits or 256 bits for $p > 1024$ bits)

Table 20. DSA private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37. (continued)

Attribute	Data type	Notes
CKA_BASE ^{1, 4, 6}	Big integer	Base g
CKA_VALUE ^{1, 4, 6, 7}	Big integer	Private value x

Table 21. Diffie-Hellman private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.

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

Table 22. Elliptic Curve private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.

Attribute	Data type	Notes
CKA_EC_PARAMS ^{1, 4, 6} (CKA_ECDSA_PARAMS)	Byte array	DER-encoding of an ANSI X9.62 Parameters value
CKA_VALUE ^{1, 4, 6, 7}	Big integer	ANSI X9.62 private value d
CKA_IBM_PROTKEY_EXTRACTABLE ¹¹ (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.
CKA_IBM_PROTKEY_NEVER_EXTRACTABLE ^{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 23. Domain parameter object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.

Attribute	Data type	Notes
CKA_CLASS¹	CKO_OBJECT_CLASS	Object class (type). An application can specify the value when the object is created (or generated) only.
CKA_TOKEN¹¹	CK_BBOOL	Default value on create is FALSE. Object hardened to the TKDS if TRUE.
CKA_PRIVATE¹¹	CK_BBOOL	Default value on create is FALSE
CKA_MODIFIABLE¹¹	CK_BBOOL	Default value is TRUE
CKA_LABEL	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.
CKA_KEY_TYPE¹	CK_KEY_TYPE	Type of key the domain parameters can be used to generate. CKK_DSA and CKK_DH only in this release
CKA_LOCAL^{2, 4}	CK_BBOOL	TRUE only if the parameters were generated locally
CKA_APPLICATION	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.
CKA_IBM_ICSF_HANDLE (vendor specific attribute - 0x80000009)	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 9 on page 37.

Attribute	Data type	Notes
CKA_PRIME^{1, 4}	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_SUBPRIME^{1, 4}	Big integer	Subprime q (160 bits for $p \leq 1024$ bits, 224 bits or 256 bits for $p > 1024$ bits)
CKA_BASE^{1, 4}	Big integer	Base g
CKA_PRIME_BITS^{2, 3}	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 9 on page 37.

Attribute	Data type	Notes
CKA_PRIME^{1, 4}	Big integer	Prime p (512 to 2048 bits in steps of 64 bits)
CKA_BASE^{1, 4}	Big integer	Base g
CKA_PRIME_BITS^{2, 3}	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 9 on page 37.

Attribute	Data type	Notes
CKA_IBM_DILITHIUM_MODE ^{1, 3}	Byte array	DER-encoding of a Dilithium key mode. The following modes are supported: <ul style="list-style-type: none"> • CRYSTALS-Dilithium (6,5) Round 2 - { 1 3 6 1 4 1 2 267 1 6 5 } • CRYSTALS-Dilithium (6,5) Round 3 - { 1 3 6 1 4 1 2 267 7 6 5 } • CRYSTALS-Dilithium (8,7) Round 2 - { 1 3 6 1 4 1 2 267 1 8 7 } • CRYSTALS-Dilithium (8,7) Round 3 - { 1 3 6 1 4 1 2 267 7 8 7 }
CKA_VALUE ^{1, 4}	Big integer	Public value.

Table 27. Dilithium private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.

Attribute	Data type	Notes
CKA_IBM_DILITHIUM_MODE ^{1, 4, 6}	Byte array	DER-encoding of a Dilithium key mode. The following modes are supported: <ul style="list-style-type: none"> • CRYSTALS-Dilithium (6,5) Round 2 - { 1 3 6 1 4 1 2 267 1 6 5 } • CRYSTALS-Dilithium (6,5) Round 3 - { 1 3 6 1 4 1 2 267 7 6 5 } • CRYSTALS-Dilithium (8,7) Round 2 - { 1 3 6 1 4 1 2 267 1 8 7 } • CRYSTALS-Dilithium (8,7) Round 3 - { 1 3 6 1 4 1 2 267 7 8 7 }
CKA_VALUE ^{1, 4, 6, 7}	Big integer	Private value.

Table 28. Kyber public key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.

Attribute	Data type	Notes
CKA_IBM_KYBER_MODE ^{1, 3}	Byte array	DER-encoding of a Kyber key mode. The following modes are supported: <ul style="list-style-type: none"> • CRYSTALS-Kyber-1024 Round 2 - { 1 3 6 1 4 1 2 267 5 4 4 } • CRYSTALS-Kyber-768 Round 2 - { 1 3 6 1 4 1 2 267 5 3 3 }
CKA_VALUE ^{1, 4}	Big integer	Public value.

Table 29. Kyber private key object attributes that ICSF supports. For the meanings of the footnotes, see Table 9 on page 37.

Attribute	Data type	Notes
CKA_IBM_KYBER_MODE^{1, 4, 6}	Byte array	DER-encoding of a Kyber key mode. The following modes are supported: <ul style="list-style-type: none"> CRYSTALS-Kyber-1024 Round 2 - { 1 3 6 1 4 1 2 267 5 4 4 } CRYSTALS-Kyber-768 Round 2 - { 1 3 6 1 4 1 2 267 5 3 3 }
CKA_VALUE^{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 76.

Functions supported

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

Standard functions supported

Table 30 on page 58 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.

Table 30. Standard PKCS #11 functions that ICSF supports	
Function	Usage notes
General purpose functions:	
C_Initialize()	<ul style="list-style-type: none"> • 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. • 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. • A call to C_Initialize() from an application that is not running POSIX-enabled results in error CKR_FUNCTION_FAILED being returned.

Table 30. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
C_Finalize()	dlclose() cannot be used as an implicit C_Finalize(). If an application uses dlclose() without calling C_Finalize(), and reinitializes PKCS #11, a subsequent call to C_Initialize() will result in error CKR_FUNCTION_FAILED being returned.
C_GetInfo()	
C_GetFunctionList()	
Slot and token management functions:	
C_GetSlotList()	<ul style="list-style-type: none"> • 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. • 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. • The tokenPresent argument flag is meaningless as all allocated slots have a token present.
C_GetSlotInfo()	
C_GetTokenInfo()	
C_WaitForSlotEvent()	<ul style="list-style-type: none"> • 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. • The CKF_DONT_BLOCK argument flag is meaningless because this function never blocks. The dynamic slot allocation occurs synchronously.
C_GetMechanismList()	<p>The list of functions returned reflects the capabilities of the current cryptographic hardware configuration.</p> <p>Note: 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>
C_GetMechanismInfo()	The output of this function reflects the capabilities of the current cryptographic hardware configuration.
C_InitToken()	Tokens are protected by the security manager through profiles in the CRYPTOZ class. PINs are not used. The pPin and ulPinLen arguments are ignored.
C_InitPIN()	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.
C_SetPIN()	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.

Table 30. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
Session management functions:	
C_OpenSession()	The Notify and pApplication arguments are ignored.
C_CloseSession()	
C_CloseAllSessions()	
C_GetSessionInfo()	The state field returned is meaningless. It is always set to CK_UNAVAILABLE_INFORMATION.
C_GetOperationState()	Returns CKR_STATE_UNSAVEABLE if a find is active or more than one cryptographic operation is active.
C_SetOperationState()	
C_Login()	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.
C_Logout()	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.
Object management functions:	
C_CreateObject()	
C_CopyObject()	
C_DestroyObject()	
C_GetObjectSize()	
C_GetAttributeValue()	
C_SetAttributeValue()	
C_FindObjectsInit()	ulCount may have the high order bit on to specify to use the RTL rule array keyword when invoking the ICSF CSFPTRL callable service.
C_FindObjects()	Sensitive attributes cannot be used as search criteria when the object is marked sensitive or not exportable. Doing so results in no match found.
C_FindObjectsFinal()	
Encryption functions:	

Table 30. Standard PKCS #11 functions that ICSF supports (continued)

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

Table 30. Standard PKCS #11 functions that ICSF supports (continued)

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

Table 30. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
Signing and message authentication coding (MACing) functions:	
C_SignInit()	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> • CKM_RSA_X_509 • CKM_RSA_PKCS • CKM_MD5_RSA_PKCS • CKM_SHA1_RSA_PKCS • CKM_SHA224_RSA_PKCS • CKM_SHA256_RSA_PKCS • CKM_SHA384_RSA_PKCS • CKM_SHA512_RSA_PKCS • CKM_DSA • CKM_DSA_SHA1 • CKM_MD5_HMAC • CKM_SHA_1_HMAC • CKM_SHA224_HMAC • CKM_SHA256_HMAC • CKM_SHA384_HMAC • CKM_SHA512_HMAC • CKM_SSL3_MD5_MAC • CKM_SSL3_SHA1_MAC • CKM_MD2_RSA_PKCS • CKM_ECDSA • CKM_ECDSA_SHA1 • 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
C_Sign()	
C_SignUpdate()	Multiple-part signature is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
C_SignFinal()	Multiple-part signature is not supported for the CKM_RSA_PKCS and CKM_RSA_X_509 mechanisms.
Functions for verifying signatures and message authentication codes (MACs):	

Table 30. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
C_VerifyInit()	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> • CKM_RSA_X_509 • CKM_RSA_PKCS • CKM_MD5_RSA_PKCS • CKM_SHA1_RSA_PKCS • CKM_SHA224_RSA_PKCS • CKM_SHA256_RSA_PKCS • CKM_SHA384_RSA_PKCS • CKM_SHA512_RSA_PKCS • CKM_DSA • CKM_DSA_SHA1 • CKM_MD5_HMAC • CKM_SHA_1_HMAC • CKM_SHA224_HMAC • CKM_SHA256_HMAC • CKM_SHA384_HMAC • CKM_SHA512_HMAC • CKM_SSL3_MD5_MAC • CKM_SSL3_SHA1_MAC • CKM_MD2_RSA_PKCS • CKM_ECDSA • CKM_ECDSA_SHA1 • 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
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.
Key management functions:	

Table 30. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
C_DeriveKey()	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> • CKM_DH_PKCS_DERIVE • CKM_SSL3_MASTER_KEY_DERIVE • CKM_SSL3_MASTER_KEY_DERIVE_DH • CKM_SSL3_KEY_AND_MAC_DERIVE • CKM_TLS_MASTER_KEY_DERIVE • CKM_TLS_MASTER_KEY_DERIVE_DH • CKM_TLS_KEY_AND_MAC_DERIVE • CKM_TLS_PRF (It is the caller's responsibility to supply an ASCII value for the seed) • CKM_ECDH1_DERIVE <p>Notes:</p> <ul style="list-style-type: none"> • A secure or clear key may be specified as the base key for derivation mechanisms CKM_ECDH1_DERIVE and CKM_DH_PKCS. • Key derivation mechanism CKM_DH_PKCS derive clear keys only. • Key derivation mechanism CKM_ECDH1_DERIVE may derive clear or secure keys (see Table 5 on page 31 for hardware requirements to derive secure keys). • Key derivation mechanisms CKM_ECDH1_DERIVE and CKM_DH_PKCS derive clear keys only.
C_GenerateKey()	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> • CKM_DES_KEY_GEN • CKM_DES2_KEY_GEN • CKM_DES3_KEY_GEN • CKM_PBE_SHA1_DES3_EDE_CBC • CKM_AES_KEY_GEN • CKM_DSA_PARAMETER_GEN • CKM_DH_PKCS_PARAMETER_GEN • CKM_BLOWFISH_KEY_GEN • CKM_RC4_KEY_GEN • CKM_GENERIC_SECRET_KEY_GEN • CKM_SSL3_PRE_MASTER_KEY_GEN • CKM_TLS_PRE_MASTER_KEY_GEN • CKM_PKCS5_PBKD2
C_GenerateKeyPair()	<p>The following mechanisms are supported:</p> <ul style="list-style-type: none"> • CKM_RSA_PKCS_KEY_PAIR_GEN • CKM_DSA_KEY_PAIR_GEN • CKM_DH_PKCS_KEY_PAIR_GEN • CKM_EC_KEY_PAIR_GEN
C_CreateObject	

Table 30. Standard PKCS #11 functions that ICSF supports (continued)

Function	Usage notes
C_CopyObject	
C_SetAttributeValue	
C_WrapKey()	<p>The following mechanisms are supported for wrapping secret keys:</p> <ul style="list-style-type: none"> • CKM_RSA_PKCS • CKM_DES_CBC_PAD • CKM_DES3_CBC_PAD • CKM_AES_CBC_PAD • CKM_IBM_ATTRIBUTEBOUND_WRAP <p>The following mechanisms are supported for wrapping private keys:</p> <ul style="list-style-type: none"> • CKM_DES_CBC_PAD • CKM_DES3_CBC_PAD • CKM_AES_CBC_PAD • CKM_IBM_ATTRIBUTEBOUND_WRAP <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>
C_UnwrapKey()	<p>The following mechanisms are supported for unwrapping secret keys:</p> <ul style="list-style-type: none"> • CKM_RSA_PKCS • CKM_DES_CBC_PAD • CKM_DES3_CBC_PAD • CKM_AES_CBC_PAD • CKM_IBM_ATTRIBUTEBOUND_WRAP <p>The following mechanisms are supported for unwrapping private keys:</p> <ul style="list-style-type: none"> • CKM_DES_CBC_PAD • CKM_DES3_CBC_PAD • CKM_AES_CBC_PAD • CKM_IBM_ATTRIBUTEBOUND_WRAP
Random number generation functions:	
C_SeedRandom()	This function always returns the value CKR_RANDOM_SEED_NOT_SUPPORTED because the z/OS hardware random number generator is self-seeding.
C_GenerateRandom()	When the hardware platform is IBM z16, the CP Assist for Cryptographic Functions (CPACF) Message-Security-Assist Extension 7 is always available and will be the source of pseudo-random bytes. The generation of pseudo random bytes is consistent with NIST SP800-90A.

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 CSFPTRL 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 CSFPTRL 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_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.

Enterprise PKCS #11 coprocessors

Secure key requests cannot be routed to the Enterprise PKCS #11 coprocessors when the ICSF FIPSMODE option is FIPS140-3,INDICATE or FIPS 140-3,ENFORCE. For more information, see [“Operating in compliance with FIPS 140” on page 11.](#)

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 140-2 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.

Table 31. List of algorithms/uses not supported/disallowed by Enterprise PKCS #11 coprocessors		
Algorithm	PKCS #11 Mechanisms or key types	Comments
MD2	CKM_MD2_RSA_PKCS	Secure private key use for signing disallowed
MD5	CKM_MD5_RSA_PKCS, CKM_MD5_HMAC	Secure private or secret key use for signing disallowed

Table 31. List of algorithms/uses not supported/disallowed by Enterprise PKCS #11 coprocessors (continued)		
Algorithm	PKCS #11 Mechanisms or key types	Comments
SSL3	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	
TLS	CKM_TLS_MASTER_KEY_DERIVE, CKM_TLS_MASTER_KEY_DERIVE_DH, CKM_TLS_KEY_AND_MAC_DERIVE	
Diffie Hellman	CKK_DH keys	Prime size less than 1024 bits
DSA	CKK_DSA keys	Combinations other than the following are not supported: <ul style="list-style-type: none"> • Prime size = 1024 bits, subprime size = 160 bits • Prime size = 2048 bits, subprime size = 224 bits or 256 bits
Single DES	CKK_DES keys	
Triple DES	CKK_DES2 keys	
Blowfish	CKK_BLOWFISH keys	
RC4	CKK_RC4 keys	
RSA	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
HMAC	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.
AES GCM	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.

<i>Table 32. PKCS #11 Access Control Points</i>		
Access Control Point name or group	Mechanism/Function requiring enablement	Number
Control Point Management		
Allow addition (activation) of control points	Not applicable	0
Allow removal (deactivation) of Control Points	Not applicable	1
Cryptographic Operations		
Sign with private keys	Sign using CKK_RSA, CKK_DSA, or CKK_ECDSA keys.	2
Sign with HMAC or CMAC	Sign using CKM_SHA_1_HMAC, CKM_SHA224_HMAC, CKM_SHA256_HMAC, CKM_SHA384_HMAC, or CKM_SHA512_HMAC.	3
Verify with HMAC or CMAC	Verify using CKM_SHA_1_HMAC, CKM_SHA224_HMAC, CKM_SHA256_HMAC, CKM_SHA384_HMAC, or CKM_SHA512_HMAC.	4
Encrypt with symmetric keys	Encrypt with CKK_DES3 or CKK_AES keys. Create Object or Copy Object where source is a clear key.	5
Decrypt with private keys	Decrypt with CKK_RSA keys.	6
Decrypt with symmetric keys	Decrypt with CKK_DES3 or CKK_AES keys.	7
Key export with public keys	Wrap Key using a CKK_RSA wrapping key.	8
Key export with symmetric keys	Wrap Key using a CKK_DES3 or CKK_AES wrapping key.	9
Key import with private keys	Unwrap Key using a CKK_RSA unwrapping key.	10
Key import with symmetric keys	Unwrap Key using a CKK_DES3 or CKK_AES unwrapping key. Create Object or Copy Object where source is a clear key.	11
Generate asymmetric key pairs	Generate Key Pair for CKK_RSA, CKK_DSA, or CKK_ECDSA keys.	12
Generate symmetric keys	Generate key for CKK_DES2 or CKK_AES keys.	13
Allow key derivation	Derive key using a CKK_DH key.	47

<i>Table 32. PKCS #11 Access Control Points (continued)</i>		
Access Control Point name or group	Mechanism/Function requiring enablement	Number
Allow protected keys	Encrypt/Decrypt using CKK_AES or CKK_DES keys. Sign using CKK_EC private keys.	64
Allow DSA/PQG parameter generation	Generate Secret Key to generate DSA domain parameters.	71
<i>Cryptographic Algorithms</i>		
Allow non-BSI algorithms (as of 2009)	Not applicable	21
RSA private-key use	Generate Key Pair for CKK_RSA Sign or Decrypt using a CKK_RSA key	30
DSA private-key use	Generate Key Pair for CKK_DSA Sign using a CKK_DSA key	31
EC private-key use	Generate Key Pair for CKK_EC Sign or Derive Key using a CKK_EC key	32
Brainpool (E.U.) EC curve use	Sign or Verify using the Brainpool curves	33
NIST/SECG EC curve use	Sign or Verify using the NIST EC curves	34
Allow non-FIPS-approved algorithms (as of 2011)	Not applicable	35
Allow non-BSI algorithms (as of 2011)	Not applicable	36
DH private-key use	Generate Key Pair for CKK_DH	46
Allow using 25519, c41417, and c448 curves	Sign or verify using 25519 and c448 curves.	55
Prime-field SECG EC curve use	Sign, Verify, or Derive using prime-field SECG EC curves excluding those shared with NIST P-curves.	60
Allow non-BSI algorithms (as of 2017)	Not applicable	61
Quantum-safe key use	Generate Dilithium or Kyber key pair (CSFPGKP).	65
BTC-related including blockchain, altcoins, and digital assets	Not applicable	66
Allow non-ECDSA/non-EdDSA EC signature algorithms	EC-SDSA rule for sign and verify, and all subvariants.	67
Allow non-FIPS-approved algorithms (as of 2021)	Not applicable.	68
Allow non-FIPS-approved algorithms (as of 2024)	Not applicable.	69
Allow fall-back to non-standard SHA3 defaults	Not applicable.	70
Key Size		

Table 32. PKCS #11 Access Control Points (continued)

Access Control Point name or group	Mechanism/Function requiring enablement	Number
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 If in a BSI mode: Any use of CKK_DSA keys, or CKK_RSA keys smaller than 2432 bits If not in a BSI mode: 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 If in a BSI mode: Any use of CKK_RSA keys larger than 2431 bits but less than 3248 bits If not in a BSI mode: Any use of CKK_RSA keys larger than 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 If in a BSI mode: Any use of CKK_RSA keys larger than 3247 bits If not in a BSI mode: Any use of CKK_RSA keys larger than 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
Allow RSA public exponents below 0x10001	Generate Key or Generate Key Pair for CKK_RSA where the exponent is 3.	29
Miscellaneous		
Allow backend to save semi-retained keys	Not applicable	14
Allow keywrap without attribute-binding	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
Allow changes to key objects (usage flags only)	Set Attribute Value or Copy Object where the key usage flags are modified	17
Allow mixing external seed to RNG	Not applicable	18
Allow non-administrators to mark key objects TRUSTED	Set Attribute Value where CKA_TRUSTED is set TRUE	37

Table 32. PKCS #11 Access Control Points (continued)		
Access Control Point name or group	Mechanism/Function requiring enablement	Number
Do not double-check sign/decrypt operations	Not applicable	38
Allow dual-function keys - key wrapping and data encryption	<p>Generate Key or Generate Key Pair where CKA_WRAP / CKA_UNWRAP and CKA_ENCRYPT / CKA_DECRYPT combinations are requested (or defaulted)</p> <p>Wrap Key, Unwrap Key, Encrypt or Decrypt with a previously created key containing the previous combination.</p> <p>Create Object or Copy Object where source is a clear key.</p>	39
Allow dual-function keys - digital signature and data encryption	<p>Create Object, Generate Key or Generate Key Pair where CKA_SIGN / CKA_VERIFY and CKA_ENCRYPT / CKA_DECRYPT combinations are requested (or defaulted)</p> <p>Sign, Verify, Encrypt or Decrypt with a previously created key containing the previous combination</p>	40
Allow dual-function keys - key wrapping and digital signature	<p>Create Object, Generate Key or Generate Key Pair where CKA_SIGN / CKA_VERIFY and CKA_WRAP / CKA_UNWRAP combinations are requested (or defaulted)</p> <p>Sign, Verify, Wrap Key or Unwrap Key with a previously created key containing the previous combination</p>	41
Allow non-administrators to mark public key objects ATTRBOUND	Create Object where CKA_IBM_ATTRBOUND is set TRUE	42
Allow clear passphrases for password-based-encryption	Generate Key using CKM_PBE_SHA1_DES3_EDE_CBC	43
Allow wrapping of stronger keys by weaker keys	Wrap Key where the to-be-wrapped key is stronger than the wrapping key.	44
Allow clear public keys as non-attribute bound wrapping keys	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
Allow use of blobs without sessions	Not applicable.	48
Allow the derivation of a non-AB or raw key from AB key	Allow derive key to derive from attribute-bound to non-attribute-bound or raw.	72
Allow usage of basic login sessions	Not applicable.	73
Allow RSA OAEP	Not applicable.	74
Allow both EXTRACTABLE and PROTKEY_EXTRACTABLE	Allow secret key objects to be created and used with both CKA_EXTRACTABLE and CKA_IBM_PROTKEY_EXTRACTABLE set to CK_TRUE.	75

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, 2011, 2017, and 2024 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.

6. **FIPS 2021** – Card adheres to FIPS restrictions that went into effect in 2021. (More restrictive than FIPS 2011.) In addition to the control points that need to be disabled for FIPS 2011, the following control points also need to be disabled:
 - a. Allow non-FIPS-approved algs (as of 2021).
 - b. Allow using 25519, c41417, and c448 curves.
 - c. Quantum-safe key use.
 - d. BTC-related including blockchain, altcoins, and digital assets.
 - e. Allow support for non-ECDSA/non-EdDSA EC signature algorithms.
 - f. Allow use of blobs without sessions.
 - g. Prime-field SECG EC curves.
 - h. Allow RSA OAEP.
7. **FIPS 2024** – Card adheres to FIPS restrictions that go into effect in 2024. (More restrictive than FIPS 2021.) In addition to the control points that need to be disabled for FIPS 2021, the following control point also needs to be disabled:
 - a. Allow non-FIPS-approved algorithms (as of 2024).

Important: All EP11 coprocessors across all LPARs sharing a TKDS must be configured identically (same compliance mode, same set of control points). While ICSF does not strongly enforce this, an environment that does not meet this requirement is not a supported configuration and might not function as expected.

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 “Considerations for migrating to session-bound objects and FIPS 2021/FIPS 2024” on page 75 for information about FIPS 2021, FIPS 2024, and session-bound objects.

See “Steps for running the pre-compiled version of testpkcs11” on page 80 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](#).

Considerations for migrating to session-bound objects and FIPS 2021/FIPS 2024

Important: A TKDS containing session bound objects **must not** be shared with a system that does not have full support for these objects (for example, the PTF for APAR OA65205 is not applied). If a system sharing the TKDS does not have full session-bound object support, you may be able to add non-session bound objects to a session-bound TKDS which leaves the TKDS with an unsupported mix of objects (both session-bound and non-session-bound). This will drive EP11 coprocessor configuration processing, which can result in all EP11 coprocessors being removed from Active status across all systems sharing the TKDS. To correct this, you will have to delete the offending objects and restart ICSF on all affected systems sharing this TKDS.

It is not possible to convert existing objects from non-session-bound to session-bound and vice versa, which means that you cannot keep any existing non-session-bound secure objects in the TKDS when moving to session-bound objects. You must either:

1. Delete all secure objects in your TKDS which differ from the compliance mode and session binding of your EP11 cards (most likely FIPS 2021/2024).
2. Create a new empty TKDS and generate new session-bound objects.

In the following scenarios, EP11 coprocessors that are prevented from coming active or removed from being active will display messages CSFM737E and CSFM135E with appropriate insert text:

```
CSFM737E coprocessor-name ci, SERIAL NUMBER nnnnnnn,  
UNSUPPORTED CONFIGURATION (reason)
```

```
CSFM135E CRYPTOGRAPHIC FEATURE IS INACTIVE. coprocessor-name cii,  
SERIAL NUMBER nnnnnnn RSN=reason.
```

There are three possible migration scenarios depending on the state of the TKDS that you plan to use for session-bound operations (for example, FIPS 2021/FIPS 2024 compliance modes):

If there are no secure objects in your TKDS, no action is required:

- If no EP11 coprocessors require session-bound objects at ICSF start (a likely scenario if just beginning the planning to exploit EP11 coprocessors):
 - Any EP11 coprocessor that becomes available after ICSF startup which requires session-bound objects will be prevented from becoming active and messages CSFM737E (with a reason of “session-bound required(N)”) and CSFM135E will be issued to indicate the configuration error.
 - Only non-session-bound objects will be allowed to be created.
- If at least one EP11 coprocessor requires session-bound objects at ICSF start (for example, is in FIPS 2021 or FIPS 2024 compliance mode):
 - Any EP11 coprocessor that does not support session-bound objects will be prevented from becoming active and messages CSFM737E (with a reason of “session-bound unsupported”) and CSFM135E will be issued to indicate the configuration error.
 - Only session-bound objects will be allowed to be created.

If there are only secure objects in your TKDS, there might be actions required:

- If there are only non-session-bound secure objects (a likely scenario when you have been using EP11 coprocessors previously):
 - Any EP11 coprocessor that requires session-bound objects will be prevented from becoming active and messages CSFM737E (with a reason of “session-bound required”) and CSFM135E will be issued to indicate the configuration error.
 - Only non-session-bound objects will be allowed to be created.
- If there are only session-bound secure objects (a likely scenario when you have already switched to session-bound objects and EP11 coprocessors):
 - Any EP11 coprocessor that does not support session-bound objects will be prevented from becoming active and messages CSFM737E (with a reason of “session-bound unsupported”) and CSFM135E will be issued to indicate the configuration error.
 - Only session-bound objects will be allowed to be created.

If there are both non-session-bound and session-bound secure objects in your TKDS, this is an unsupported configuration:

- No secure object work can be done:
 - All EP11 coprocessors will be prevented from becoming active or will be made inactive and messages CSFM737E (with a reason of “TKDS has mixed session binding”) and CSFM135E will be issued to indicate the configuration error.
 - No new secure objects will be allowed to be created.
- Either all session-bound or all non-session-bound objects must be removed from the TKDS and ICSF must be restarted to return secure object support.

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 function return codes are returned as vendor-defined return 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  
Or ICSF Return Code 0 with non-zero reason code */  
#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 non-zero return code / reason codes:

```
/* Algorithm or key size is not valid in FIPS mode */  
#define CKR_IBM_ICSF_NOT_VALID_FIPS 0xC0080BFE  
  
/* Request did not met requirements for ICSF FIPS 140-2 or 140-3 algorithm checking */  
#define CKR_IBM_ICSF_FIPS_NONE_MET 0xC0080CC6  
  
/* Request did not met requirements for ICSF FIPS 140-3 algorithm checking */  
#define CKR_IBM_ICSF_FIPS_140_2_MET_FAIL 0xC0080CC7  
  
/* PKCS #11 secure key processing not allowed in current FIPSMODE */  
#define CKR_IBM_ICSF_FIPS_140_3_NO_SECURE_KEY 0xC0080CE3  
  
/* 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
```

The following constants are defined for successful ICSF cryptographic request return code 0 when the CKF_IBM_FIPS_1403 flag is on in CK_SESSION_INFO:

```
/* Request met requirements for ICSF FIPS 140-2 algorithm checking, but not FIPS 140-3 */  
#define CKR_IBM_ICSF_FIPS_140_2_MET 0xC0000CA9  
  
/* Request met requirements for ICSF FIPS 140-3 algorithm checking  
*/  
#define CKR_IBM_ICSF_FIPS_140_3_MET 0xC0000CAA
```

For information on how to set the CKF_IBM_FIPS_1403 flag in CK_SESSION_INFO so that the function return codes for successful cryptographic requests are returned using the FIPS 140-3 return code zero / non-zero reason codes defined above, see [“FIPS 140-3 mode programming”](#) on page 24.

For details on the non-zero reason code values issued for cryptographic requests, see the FIPSMODE 140-3 options in [“Operating in compliance with FIPS 140”](#) on page 11.

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 33. Environment variables for capturing trace data

Environment variable	Usage	Valid values
CSN_PKCS11_TRACE	Specifies the level of tracing to be performed.	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: 7 Debug information 6 Informational conditions 5 Normal but significant conditions 4 Warning conditions 3 Error conditions 2 Critical conditions 1 Immediate action required Any other value causes tracing to be inactive (the default).
CSN_PKCS11_TRACE_FILE	Specifies the name of the trace file. Defaults to /tmp/csnpkcs11.%.trc. 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: /tmp/csnpkcs11.%.trc and the current process identifier is 247, the trace file name is /tmp/csnpkcs11.247.trc	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.

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 79.](#)

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 81 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 87.

- **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 74 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 (in decimal) to use as the new compliance mode. FIPS 2021 and FIPS 2024 modes are not supported, nor is converting from non-session bound objects to session bound objects. 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 76](#).

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 is 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 80](#).

Chapter 4. ICSF PKCS #11 callable services

The PKCS #11 C language API (described in [Chapter 2, “The C API,” on page 23](#)) 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 79.

```
/*
*****
/* 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 <csnprintf.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";
#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;
    }

    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 Documentation for z/OS \(www.ibm.com/docs/en/zos\)](http://www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS documentation see [How to Send Feedback to IBM](#) to leave documentation feedback.

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 IBM Documentation. 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 and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS \(www.ibm.com/software/support/systemsz/lifecycle\)](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 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).

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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'.

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

CEX8A

Crypto Express8 Accelerator

CEX8C

Crypto Express8 CCA Coprocessor

CEX8P

Crypto Express8 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 Express5 Coprocessor

An asynchronous cryptographic coprocessor available on IBM z13.

Crypto Express6 Coprocessor

An asynchronous cryptographic coprocessor available on IBM z14.

Crypto Express7 Coprocessor

An asynchronous cryptographic coprocessor available on IBM z15, IBM z16, and IBM z17.

Crypto Express8 Coprocessor

An asynchronous cryptographic coprocessor available on IBM z16 and IBM z17.

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. This algorithm has been superseded by ML-DSA.

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 reversion 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.

Kyber

A quantum-safe cryptographic algorithm. This algorithm has been superseded by ML-KEM.

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.

ML-DSA

A quantum-safe cryptographic algorithm. This algorithm supersedes Dilithium.

ML-KEM

A quantum-safe cryptographic algorithm. This algorithm supersedes Kyber.

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 ML-DSA, ML-KEM, CRYSTALS-Dilithium Digital Signature Algorithm and CRYSTALS-Kyber key encapsulation mechanism (KEM). ML-DSA and ML-KEM supersede Dilithium and Kyber respectively.

RACE Integrity Primitives Evaluatiuon Message Digest

A hash algorithm.

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 Measurement Facility.

RMI

Resource Manager Interface.

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 [95](#)
- additional manifest constants
 - Dilithium quantum-safe algorithms [37](#)
 - Kyber algorithms [37](#)
- assistive technologies [95](#)
- auditing PKCS #11 functions [9](#)

C

- C application program interface (API), using [23](#)
- CCA (Common Cryptographic Architecture) [1](#)
- CK_RV CSN_FindAllObjects() function [67](#)
- Common Cryptographic Architecture (CCA) [1](#)
- component trace entries for TKDS events [10](#)
- constants, manifest
 - where defined [23](#)
- contact
 - z/OS [95](#)
- crypto education [viii](#)
- Cryptoki [1](#)
- CRYPTOZ class [3](#)
- CSFSERV class
 - resources for token services [4](#)

D

- data object
 - attributes that ICSF supports [37](#)
- deleting token [23](#)
- Diffie-Hellman domain parameter object
 - attributes that ICSF supports [37](#)
- Diffie-Hellman private key object
 - attributes that ICSF supports [37](#)
- Diffie-Hellman public key object
 - attributes that ICSF supports [37](#)
- Dilithium quantum-safe algorithms
 - additional manifest constants [37](#)
- DLLs provided by ICSF [85](#)
- domain parameter object
 - attributes that ICSF supports [37](#)
- DSA domain parameter object
 - attributes that ICSF supports [37](#)
- DSA private key object
 - attributes that ICSF supports [37](#)
- DSA public key object
 - attributes that ICSF supports [37](#)
- dynamic link libraries (DLLs) provided by ICSF [85](#)

E

- Elliptic Curve private key object
 - attributes that ICSF supports [37](#)
- Elliptic Curve public key object

- Elliptic Curve public key object (*continued*)
 - attributes that ICSF supports [37](#)
- environment variables for tracing [78](#)

F

- FIPS 140
 - operating in compliance with [11](#)
- FIPS 140-2
 - algorithms restricted when complying with [37](#)
- FIPSMODE installation option [11](#)
- function
 - non-standard PKCS #11 supported by ICSF [67](#)
 - standard PKCS #11 supported by ICSF [58](#)
- function return code
 - unique to z/OS, list of [76](#)

H

- header file for C API [85](#)

I

- installation option keyword
 - FIPSMODE [11](#)

K

- key types supported [25](#)
- keyboard
 - navigation [95](#)
 - PF keys [95](#)
 - shortcut keys [95](#)
- Kyber algorithms
 - additional manifest constants [37](#)

L

- Language Environment [23](#)
- library
 - information that can be set and queried [57](#)

M

- manifest constants
 - where defined [23](#)
- mechanism
 - information returned by C_GetMechanismInfo [26](#)
 - which cryptographic hardware supports [30](#)

N

- navigation
 - keyboard [95](#)

O

object

data

- attributes that ICSF supports [37](#)

- Diffie-Hellman domain parameter

- attributes that ICSF supports [37](#)

- Diffie-Hellman private key

- attributes that ICSF supports [37](#)

- Diffie-Hellman public key

- attributes that ICSF supports [37](#)

- domain parameter

- attributes that ICSF supports [37](#)

- DSA domain parameter

- attributes that ICSF supports [37](#)

- DSA private key

- attributes that ICSF supports [37](#)

- DSA public key

- attributes that ICSF supports [37](#)

- Elliptic Curve private key

- attributes that ICSF supports [37](#)

- Elliptic Curve public key

- attributes that ICSF supports [37](#)

- private key

- attributes that ICSF supports [37](#)

- RSA private key

- attributes that ICSF supports [37](#)

- RSA public key

- attributes that ICSF supports [37](#)

- secret key

- attributes that ICSF supports [37](#)

- X.509 certificate

- attributes that ICSF supports [37](#)

object type

- supported by ICSF, list of [37](#)

P

PIN [3](#)

private key object

- attributes that ICSF supports [37](#)

program, sample

- building and using [79](#)

- source code for [87](#)

Public Key Cryptography Standards (PKCS) [1](#)

public key object

- attributes that ICSF supports [37](#)

R

RSA private key object

- attributes that ICSF supports [37](#)

RSA public key object

- attributes that ICSF supports [37](#)

S

sample C program, PKCS #11

- building and using [79](#)

sample program, testpkcs11

- source code for [87](#)

secret key object

- attributes that ICSF supports [37](#)

session object [10](#)

shortcut keys [95](#)

slot

- information that can be queried [57](#)

slot ID [1](#)

SMF records written for PKCS #11 functions [9](#)

SO R/W

- description [3](#)

SO role [3](#)

SRB mode [23](#)

Strong SO

- description [3](#)

summary of changes [xi](#), [xii](#)

T

tasks

- running the sample program

- steps [80](#)

TCB mode [23](#)

testpkcs11 program

- building and using [79](#)

- source code for [87](#)

TKDS (token data set)

- description [2](#)

token

- access levels [3](#)

- deleting [23](#)

- information that can be set and queried [57](#)

- managing, interfaces for [7](#)

token data set (TKDS)

- description [2](#)

token object [11](#)

token, on z/OS

- description [1](#)

- rules for name [1](#)

trace data [78](#)

trace entries for TKDS events [10](#)

trademarks [100](#)

troubleshooting applications [78](#)

U

updatecomp program

- building and using [79](#)

user interface

- ISPF [95](#)

- TSO/E [95](#)

User R/O

- description [4](#)

User R/W

- description [4](#)

User role [3](#)

W

Weak SO

- description [3](#)

Weak User

- description [4](#)

X

X.509 certificate object
attributes that ICSF supports [37](#)

Z

z/OS PKCS #11 token
deleting [23](#)
description [1](#)
rules for name [1](#)



SC14-7510-70

