ActivID® ActivClient® SDK
PKCS#11 API Reference Guide

DOCUMENT REFERENCE: AC_SDK_PKCS#11_7.1_RG_07.2016

PRODUCT VERSION: 7.1

JULY 2016



Copyright

© 2008-2016 HID Global Corporation/ASSA ABLOY AB. All rights reserved. This document may not be reproduced, disseminated or republished in any form without the prior written permission of HID Global Corporation.

Trademarks

HID, HID Global, the HID Blue Brick logo, the Chain Design and 4TRESS, ActivIdentity and ActivID are trademarks or registered trademarks of HID Global or its licensor(s)/supplier(s) in the US and other countries and may not be used without permission. The absence of a mark, product, service name or logo from this list does not constitute a waiver of the HID Global trademark or other intellectual property rights concerning that name or logo. All other trademarks, service marks, and product or service names are trademarks or registered trademarks of their respective owners.

Revision History

Date	Description	Document Version
July 2016	Initial release of rebranded document and major technical updates.	1.0

Contacts

Americas		Asia Pacific
United States: +1 510.574.0100	U.S. Federal: +1 571.522.1000	+61 (0) 3.9809.2892
Europe		**Corporate**
+33 (0) 1.42.04.84.00		+1 800.237.7769

Technical Support

If you purchased your product from a third party, then please contact that third party for Technical Support. For products purchased directly from HID Global, please go to http://www.hidglobal.com/support

Typographic and Document Conventions

Typography	Description
blue	Cross-references within the document.
blue, underline	References to external web addresses.
bold	Action steps (paths, buttons, options); field and drop-down list labels; emphasis.
italic	File names, document titles, and file extensions.
`Code snippets`	Highlights `code snippets` within regular content.
`Code samples`	`Highlights code samples`
	**WARNING**: This symbol indicates a critical warning. It applies to actions that if taken or not taken will break the system. Read the warning carefully and follow it.
	**Important**: This symbol indicates something very important to the reader. Ignore this symbol at your own risk.
	**Note**: This symbol indicates a note that should be of interest to the reader. It is not critical. Nevertheless, the reader should pay attention.

Table of Contents

1.0 Introduction 8

1.1 Product Overview 8
1.2 Document Scope and Audience 8
1.3 About the PKCS#11 API 9

2.0 Overview 10

2.1 Required Include Files 10
2.2 How to Deploy 10

3.0 Packing List 11

3.1 Header Files 11
3.2 Static Libraries 11
3.3 Samples 11
3.3.1 Features Demonstrated in the Samples 11
3.3.2 PKCS#11 C Sample 11
3.3.3 PKCS#11 Java Sample 11

4.0 Cryptoki Version 12

5.0 Slot Information 13

5.1 For NFC 13
5.2 For USB Reader 13
5.3 For ActivID ActivKey 14

6.0 Token Information 15

7.0 List of Supported Functions 18

8.0 List of the Constants 22

8.1 Mechanism Information Flags 22
8.2 Mechanisms 22

9.0 Objects 32

9.1 Certificate Object 32
9.1.1 X.509 Certificate Object Attributes 32
9.1.2 Common Certificate Object Attributes 32
9.1.3 Common Storage Object Attributes 33
9.1.4 Vendor Attributes 33
9.2 RSA Public Key Object 34
9.2.1 RSA Public Key Object Attributes 34
9.2.2 Common Public Key Attributes 34
9.2.3 Common Key Attributes 35
9.2.4 Common Storage Object Attributes 36
9.3 RSA Private Key Object 37
9.3.1 RSA Private Key Object Attributes 37

9.3.2 Common Private Key Attributes 38
9.3.3 Common Key Attributes 39
9.3.4 Common Storage Object Attributes 39
9.3.5 Vendor Attributes 40
9.4 Signature Status Flag Definition 40
9.5 Access to PIV Object 41
9.5.1 Common Storage Object Attributes 41
9.5.2 PIV Objects Attributes 41
9.5.2.1 CHUID Object Attributes 42
9.5.2.2 Facial Image Object Attributes 45
9.5.2.3 Fingerprints Object Attributes 46
9.5.2.4 Printed Info Object Attributes 47
9.5.2.5 Security Object Attributes 48
9.5.2.6 Discovery Object Attributes (SP800-73-2) 49
9.5.2.7 CCC Object Attributes 49
9.5.2.8 Key History Object Attributes (SP800-73-3) 51
9.5.2.9 Iris Images Object Attributes 52
9.6 Data Object 53
9.6.1 Common Data Object Attributes 53
9.6.2 Common Storage Object Attributes 53

10.0 Token-Information Flags 54

11.0 Proprietary Flag 56

12.0 Functions 57

12.1 General Purpose Functions 57
12.1.1 C_Initialize 57
12.1.2 C_Finalize 58
12.1.3 C_Get_Info 59
12.1.4 C_GetFunctionList 59
12.2 Slot and Token Management Functions 60
12.2.1 C_GetSlotList 60
12.2.2 C_GetSlotInfo 61
12.2.3 C_GetTokenInfo 61
12.2.4 C_WaitForSlotEvent 62
12.2.5 C_GetMechanismList 63
12.2.6 C_GetMechanismInfo 64
12.2.7 C_InitToken 64
12.2.8 C_InitPIN 65
12.2.9 C_SetPIN 66
12.3 Session Management Functions 67
12.3.1 C_OpenSession 67
12.3.2 C_CloseSession 68
12.3.3 C_CloseAllSessions 69
12.3.4 C_GetSessionInfo 70
12.3.5 C_GetOperationState 70
12.3.6 C_SetOperationState 70
12.3.7 C_Login 70

12.3.8 C_Logout ... 71
12.4 Objet Management Functions ... 72
12.4.1 C_CreateObject ... 72
12.4.2 C_CopyObject ... 73
12.4.3 C_DestroyObject ... 74
12.4.4 C_GetObjectSize ... 74
12.4.5 C_GetAttributeValue ... 75
12.4.6 C_SetAttributeValue ... 76
12.4.7 C_FindObjectsInit ... 77
12.4.8 C_FindObjects ... 78
12.4.9 C_FindObjectsFinal ... 78
12.5 Encryption Functions ... 79
12.5.1 C_EncryptInit ... 79
12.5.2 C_Encrypt ... 80
12.5.3 C_EncryptUpdate ... 81
12.5.4 C_EncryptFinal ... 81
12.6 Decryption Functions ... 81
12.6.1 C_DecryptInit ... 81
12.6.2 C_Decrypt ... 82
12.6.3 C_DecryptUpdate ... 83
12.6.4 C_DecryptFinal ... 84
12.7 Message Digesting Functions ... 85
12.7.1 C_DigestInit ... 85
12.7.2 C_Digest ... 85
12.7.3 C_DigestUpdate ... 86
12.7.4 C_DigestFinal ... 87
12.7.5 C_DigestKey ... 88
12.8 Signing and MACing Functions ... 88
12.8.1 C_SignInit ... 88
12.8.2 C_Sign ... 89
12.8.3 C_SignUpdate ... 90
12.8.4 C_SignFinal ... 90
12.8.5 C_SignRecoverInit ... 91
12.8.6 C_SignRecover ... 91
12.9 Verifying Signatures and MACs Functions ... 91
12.9.1 C_VerifyInit ... 91
12.9.2 C_Verify ... 92
12.9.3 C_VerifyUpdate ... 93
12.9.4 C_VerifyFinal ... 93
12.9.5 C_VerifyRecoverInit ... 93
12.9.6 C_VerifyRecover ... 93
12.10 Dual-Function Cryptographic Functions ... 93
12.10.1 C_DigestEncryptUpdate ... 93
12.10.2 C_DecryptDigestUpdate ... 93
12.10.3 C_SignEncryptUpdate ... 94
12.10.4 C_DecryptVerifyUpdate ... 94
12.11 Key Management Functions ... 94
12.11.1 C_GenerateKey ... 94

12.11.2 C_GenerateKeyPair 95
12.11.3 C_WrapKey 96
12.11.4 C_UnwrapKey 97
12.11.5 C_DeriveKey 99
12.12 Random Number Generation Functions 99
12.12.1 C_SeedRandom 99
12.12.2 C_GenerateRandom 99
12.13 Parallel Function Management Functions 99
12.13.1 C_FunctionStatus 99
12.13.2 C_CancelFunction 99
12.14 Proprietary Functions 99
12.14.1 C_VerifyUnblockPINInit 99
12.14.2 C_VerifyUnblockPIN 100
12.15 Legacy Proprietary Functions 101
12.15.1 C_UnlockPIN 101
12.15.2 C_ResetToken 101
12.15.3 C_GetCUID/C_CetCUID 101

13.0 One-Time Password Support 102

13.1 Mechanisms 102
13.2 Key Objects 102
13.2.1 Key Objects Attributes 102
13.2.2 Proprietary Key Attributes 104
13.3 Mechanism Parameters 104
13.3.1 Parameters Type 104
13.3.2 OTP Mechanisms Flags 105
13.3.3 Parameters Structures 106
13.4 OTP Functions 106

14.0 Unblock PIN Support 108

14.1 Mechanisms 108
14.2 Unblock PIN Object Definition 108
14.3 Object Attributes 108
14.3.1 Common Storage Object Attributes 109
14.4 Static Unlock Key Object Definition (Legacy) 109
14.4.1 Secret Key Attributes 109
14.4.2 Common Key Attributes 110
14.4.3 Common Storage Object Attributes 110

Appendix A: Terms and Acronyms 111

A.1. Terms 111
A.2. Acronyms 113

1.0 Introduction



ActivID® ActivClient® guards against an ever-changing threat landscape by providing organizations with risk-appropriate and secure access to corporate IT assets.

1.1 Product Overview

HID Global's ActivID ActivClient is the latest smart card and USB token middleware from that allows enterprise and government customers to easily use smart cards and USB tokens to protect workstations and networks.

ActivClient enables the use of PKI certificates and keys and one-time passwords and static password credentials on a smart card or USB token to secure:

- Desktop applications
- Network logon
- Remote access
- Web logon
- E-mail
- Electronic transactions

ActivClient provides the following range of services:

- PKI services
- Remote access and One-Time Password (OTP) services
- Remote session services
- Management services (for end users and administrators)
- Development services with a Software Development Kit (SDK)

1.2 Document Scope and Audience

This reference guide describes the HID Global implementation of the PKCS#11 API (version 2.20). It presents an overview of the services provided by the PKCS#11 API, along with the necessary information about the HID Global implementation.

Developers may also refer to the official documentation about the PKCS#11 standard at http://www.emclink.net/emc-plus/rsa-labs/standards-initiatives/pkcs-11-cryptographic-token-interface-standard.htm.

Readers of this document are assumed to be experienced programmers who:

- Are accustomed to reading and writing C code,
- Intend to develop custom application(s) into which cryptographic processing is embedded,

- Have a good understanding of cryptography applications.

1.3 About the PKCS#11 API

ActivClient's implementation of PKCS#11 complies with the PKCS#11 v2.20 documentation regarding behavior and return values of functions.

For additional information about this standard, go to http://www.emclink.net/emc-plus/rsa-labs/standards-initiatives/pkcs-11-cryptographic-token-interface-standard.htm

Prerequisites

The PKCS#11 API is supported on platforms where ActivClient has been installed. Refer to the *ActivID ActivClient for Windows Installation Guide* for pre-requisites for ActivClient installation.

Use this API to create generic PKI-based applications.

Supports: PKI and static passwords

Languages: C; HID Global provides samples in C and Java.

Description: ActivClient SDK's PKCS#11 API is a generic implementation of the PKCS#11 v2.20 standard.

The PKCS#11 API:

- Provides support for:
  - Smart card cryptographic operations.
  - Client-based management of smart cards and smart card readers.
- Is recommended for developers who want to:
  - Perform smart card cryptographic operations, use data storage, or have close control of the smart card state (card insertion and removal and PIN entry and change).
  - Reuse their code on a non-Windows platform.

2.0 Overview

2.1 Required Include Files

Include files are located in the **PKCS #11/C/Headers** folder of the distribution.

2.2 How to Deploy

PKCS#11 API applications can be linked with the static pkcs import library.

For 32-bit platforms, the static import library is located in the **PKCS #11/C/Libraries/x86** folder in the distribution; for the 64-bit platforms, it is located in the **PKCS #11/C/Libraries/x64** folder.

3.0 Packing List

3.1 Header Files

- aipkcs.h
- pkcs11.h
- pkcs11f.h
- pkcs11t.h

3.2 Static Libraries

- acpkcs211.lib, x86 version
- acpkcs211.lib, x64 version

3.3 Samples

3.3.1 Features Demonstrated in the Samples

The samples demonstrate the following scenarios:

- How to use a signature key to sign data.
- How to log on to the card.
- How to write a secret key on the card, use it to encrypt or decrypt some data and then delete it.
- How to generate an OTP.

3.3.2 PKCS#11 C Sample

Samples for both the 32 and 64-bit platforms are provided.

The PKCS#11 C x86 sample works with both ActivClient 32 and 64-bit editions.

3.3.3 PKCS#11 Java Sample

Rather than provide a proprietary Java wrapper for PKCS#11, HID Global recommends that you use the standard and free Oracle Java Wrapper API which is installed by Oracle JRE/JDK version 1.7 and later.

The PKCS#11 Java sample shows how to use the Java Wrapper with the ActivClient PKCS#11 API. This sample runs on both 32 and 64-bit platforms.

4.0 Cryptoki Version

The Cryptoki interface supported is a subset of the v2.2 Cryptoki interface.

`CK_INFO` provides general information about the library. This structure is set by the `C_GetInfo` entry point.

It is defined as follows:

```
typedef struct CK_INFO {
  CK_VERSION cryptokiVersion;
  CK_UTF8CHAR manufacturerID[32];
  CK_FLAGS flags;
  CK_UTF8CHAR libraryDescription[32];
  CK_VERSION libraryVersion;
} CK_INFO;
```

The fields of the structure have the following meanings:

Field	Meaning	Value
cryptokiVersion	Cryptoki interface version number	2.20
manufacturerID	ID of the Cryptoki library manufacturer	HID Global
flags	bit flags reserved for future versions	0
libraryDescription	character-string description of the library	HID Global Cryptoki
libraryVersion	Cryptoki library version number	2.30

5.0 Slot Information

CK_SLOT_INFO provides information about a slot. This structure is set by the C_GetSlotInfo entry point.

It is defined as follows:

```
typedef struct CK_SLOT_INFO {
  CK_UTF8CHAR slotDescription[64];
  CK_UTF8CHAR manufacturerID[32];
  CK_FLAGS flags;
  CK_VERSION hardwareVersion;
  CK_VERSION firmwareVersion;
} CK_SLOT_INFO;
```

5.1 For NFC

Not applicable for this version.

5.2 For USB Reader

Field	Meaning	Value
slotDescription	Character-string description of the slot	PCSC returned value
manufacturerID	ID of the slot manufacturer	PCSC returned value
flags	Provides capabilities of the slot	CKF_TOKEN_PRESENT: True if a card is inserted, else false. CKF_REMOVABLE_DEVICE: true CKF_HW_SLOT: True
hardwareVersion	version number of the slot's hardware*	1.0
firmwareVersion	version number of the slot's firmware*	1.0

*hardwareVersion, firmwareVersion are hardcoded.

5.3 For ActivID ActivKey

Field	Meaning	Value
slotDescription	Character-string description of the slot	PCSC returned value
manufacturerID	ID of the slot manufacturer	PCSC returned value
flags	Provides capabilities of the slot	CKF_TOKEN_PRESENT: True if a card is inserted, else false. CKF_REMOVABLE_DEVICE: true CKF_HW_SLOT: True
hardwareVersion	version number of the slot's hardware*	1.0
firmwareVersion	version number of the slot's firmware*	1.0

*hardwareVersion, firmwareVersion are hardcoded.

6.0 Token Information

`CK_TOKEN_INFO` provides information about a token. This structure is set by the `C_GetTokenInfo` entry point.

It is defined as follows:

```
typedef struct CK_TOKEN_INFO {
  CK_UTF8CHAR label[32];
  CK_UTF8CHAR manufacturerID[32];
  CK_UTF8CHAR model[16];
  CK_CHAR serialNumber[16];
  CK_FLAGS flags;
  CK_ULONG ulMaxSessionCount;
  CK_ULONG ulSessionCount;
  CK_ULONG ulMaxRwSessionCount;
  CK_ULONG ulRwSessionCount;
  CK_ULONG ulMaxPinLen;
  CK_ULONG ulMinPinLen;
  CK_ULONG ulTotalPublicMemory;
  CK_ULONG ulFreePublicMemory;
  CK_ULONG ulTotalPrivateMemory;
  CK_ULONG ulFreePrivateMemory;
  CK_VERSION hardwareVersion;
  CK_VERSION firmwareVersion;
  CK_CHAR utcTime[16];
} CK_TOKEN_INFO;
```

The fields of the structure have the following meanings:

Field	Meaning	Value
Label	application-defined label	“HID Global ActivClient X”, where X is the slot number, i.e. if only one slot is available, X equals 0.
manufacturerID	Name of the card manufacturer.	This value is card dependent.
Model	Card Model	This value is card dependent.
serialNumber	Cad Serial Number (its CUID or	This value is card dependent.

Field	Meaning	Value
	IIN/CIN field)	
Flags	See below.	See Table 2: CK_TOKEN_INFO Flags Definition on page 54.
ulMaxSessionCount	maximum number of sessions that can be opened with the token at one time by a single application	CKA_EFFECTIVELY_INFINITE
ulSessionCount	number of sessions that this application currently has open with the token	Integer between [0, infinite]
ulMaxRwSessionCount	maximum number of read/write sessions that can be opened with the token at one time by a single application	CKA_EFFECTIVELY_INFINITE
ulRwSessionCount	number of read/write sessions that this application currently has open with the token	Integer between [0, infinite]
ulMaxPinLen	Maximum PIN Length	This value is profile dependent
ulMinPinLen	Minimum PIN Length	This value is profile dependent
ulTotalPublicMemory	the total amount of memory on the token in bytes in which public objects may be stored	CK_UNAVAILABLE_INFORMATION (defined as 0)
ulFreePublicMemory	the amount of free (unused) memory on the token in bytes for public objects	CK_UNAVAILABLE_INFORMATION (defined as 0)
ulTotalPrivateMemory	the total amount of memory on the token in bytes in which private objects may be stored	CK_UNAVAILABLE_INFORMATION (defined as 0)
ulFreePrivateMemory	the amount of free (unused) memory on the token in bytes for private objects	CK_UNAVAILABLE_INFORMATION (defined as 0)
hardwareVersion	version number of the hardware*	1.0

Field	Meaning	Value
firmwareVersion	version number of the firmware*	1.0
Utctime	Current Time	Meaningless so empty string.

*hardwareVersion, firmwareVersion are hardcoded.

7.0 List of Supported Functions

The following table lists the functions implemented in the PKCS#11 API.

All the functions conform to the Cryptographic Token Interface Standard available at http://www.emclink.net/emc-plus/rsa-labs/standards-initiatives/pkcs-11-cryptographic-token-interface-standard.htm

Lines in grey correspond to unsupported functions.

Table 1: Summary of Supported Cryptoki Functions

Category	Function	Description	Supported
General purpose functions	C_Initialize	initializes Cryptoki	Yes
	C_Finalize	clean up miscellaneous Cryptoki-associated resources	Yes
	C_GetInfo	obtains general information about Cryptoki	Yes
	C_GetFunctionList	obtains entry points of Cryptoki library functions	Yes
Slot and token management functions	C_GetSlotList	obtains a list of slots in the system	Yes
	C_GetSlotInfo	obtains information about a particular slot	Yes
	C_GetTokenInfo	obtains information about a particular token	Yes
	C_GetMechanismList	obtains a list of mechanisms supported by a token	Yes
	C_GetMechanismInfo	obtains information about a particular mechanism	Yes
	C_WaitForSlotEvent	waits for a slot event, such as token insertion or token removal	Yes
	C_InitToken	Initializes a token.	Yes
	C_InitPin	initializes the normal user's PIN	Yes
	C_SetPin	modifies the PIN of the user that is currently logged in	Yes
Session management functions	C_OpenSession	opens a connection between an application and a particular token or sets up an application callback for token insertion	Yes
	C_CloseSession	closes a session	Yes
	C_CloseAllSessions	closes all sessions with a token	Yes
	C_GetSessionInfo	obtains information about the session	Yes
	C_Login	logs into a token	Yes
	C_Logout	logs out from a token	Yes

Category	Function	Description	Supported
	C_GetOperationState	obtains a copy of the cryptographic operations state of a session, encoded as a string of bytes.	No
	C_SetOperationState	restores the cryptographic operations state of a session from a string of bytes obtained with **C_GetOperationState**	No
Object management functions	C_GetObjectSize	obtains the size of an object in bytes	Yes
	C_GetAttributeValue	obtains an attribute value of an object	Yes
	C_FindObjectsInit	initializes an object search operation	Yes
	C_FindObjects	continues an object search operation	Yes
	C_FindObjectsFinal	finishes an object search operation	Yes
	C_CreateObject	Create a new object	Yes
	C_CopyObject	Copy an object	Yes
	C_DestroyObject	Destroy an object	Yes
	C_SetAttributeValue	Set an attribute value of an object	Yes
Encryption Functions (only for RSA Keys, no symmetric keys)	C_EncryptInit	initializes an encryption operation	Yes
	C_Encrypt	encrypts single-part data	Yes
	C_EncryptUpdate	continues a multiple-part encryption operation	No
	C_EncryptFinal	finishes a multiple-part encryption operation	No
Decryption Functions (only for RSA Keys, no symmetric keys)	C_DecryptInit	initializes a decryption operation	Yes
	C_Decrypt	decrypts single-part encrypted data	Yes
	C_DecryptUpdate	continues a multiple-part decryption operation	Yes
	C_DecryptFinal	finishes a multiple-part decryption operation	Yes
Message digesting functions	C_DigestInit	initializes a message-digesting operation	Yes
	C_Digest	digests single-part data	Yes
	C_DigestUpdate	continues a multiple-part digesting operation	Yes
	C_DigestFinal	finishes a multiple-part digesting operation	Yes
	C_DigestKey	continues a multiple-part message-digesting operation by digesting the value of a secret key	No
Signing and MACing functions	C_SignInit	initializes a signature operation (also applies to One-Time Password computation)	Yes
	C_Sign	signs single-part data (also applies to One-Time Password computation)	Yes

Category	Function	Description	Supported
	C_SignUpdate	continues a multiple-part signature operation	Yes
	C_SignFinal	finishes a multiple-part signature operation	Yes
	C_SignRecoverInit		No
	C_SignRecover		No
Verifying signatures and MACs functions	C_VerifyInit	Initializes a verification operation	Yes
	C_Verify	verifies a signature in a single-part operation	Yes
	C_VerifyUpdate	continues a multiple-part verification operation	No
	C_VerifyFinal	finishes a multiple-part verification operation	No
	C_VerifyRecoverInit	initializes a signature verification operation, where the data is recovered from the signature	No
	C_VerifyRecover	verifies a signature in a single-part operation, where the data is recovered from the signature	No
Dual-function Cryptographic functions	C_DigestEncryptUpdate		No
	C_DecryptDigestUpdate		No
	C_SignEncryptUpdate		No
	C_DecryptVerifyUpdate		No
Key management (only for RSA Keys, no symmetric keys)	C_GenerateKey	generates a secret key or set of domain parameters	Yes
	C_GenerateKeyPair	generates a public/private key pair	Yes
	C_WrapKey	wraps (i.e., encrypts) a private or secret key	Yes
	C_UnwrapKey	unwraps (i.e. decrypts) a wrapped key	Yes
	C_DeriveKey	derives a key from a base key, creating a new key object	No
Random number generation functions	C_SeedRandom	mixes additional seed material into the token's random number generator.	No
	C_GenerateRandom	generates random or pseudo-random data	No
Parallel function management	C_GetFunctionStatus	legacy, should return CKR_FUNCTION_NOT_PARALLEL	No
	C_CancelFunction	legacy, should return CKR_FUNCTION_NOT_PARALLEL	No
Unblock PIN management (proprietary)	C_VerifyUnblockPINInit	Initializes an Unblock PIN operation.	Yes
	C_VerifyUnblockPIN	Verifies an unblock PIN code in a single part operation.	Yes

Category	Function	Description	Supported
Legacy Proprietary functions	C_UnlockPIN	legacy, deprecated	No
	C_ResetToken	legacy, deprecated	No
	C_GetCUID/C_CetCUID	legacy, deprecated	No

8.0 List of the Constants

8.1 Mechanism Information Flags

Bit Flag	Mask	Meaning
CKF_HW	0x00000001	True if the mechanism is performed by the device; false if the mechanism is performed in software
CKF_ENCRYPT	0x00000100	True if the mechanism can be used with C_EncryptInit
CKF_DECRYPT	0x00000200	True if the mechanism can be used with C_DecryptInit
CKF_DIGEST	0x00000400	True if the mechanism can be used with C_DigestInit
CKF_SIGN	0x00000800	True if the mechanism can be used with C_SignInit
CKF_VERIFY	0x00002000	True if the mechanism can be used with C_VerifyInit. Must be false for this version.
CKF_GENERATE	0x00008000	True if the mechanism can be used with C_GenerateKey. Must be false for this version.
CKF_GENERATE_KEY_PAIR	0x00010000	True if the mechanism can be used with C_GenerateKeyPair. Must be false for this version.
CKF_WRAP	0x00020000	True if the mechanism can be used with C_WrapKey. Must be false for this version.
CKF_UNWRAP	0x00040000	True if the mechanism can be used with C_UnwrapKey. Must be false for this version.
CKF_EXTENSION	0x80000000	True if there is an extension to the flags; false if no extensions. Must be false for this version.

8.2 Mechanisms

The following table lists which Cryptoki mechanisms (CK_MECHANISM_TYPE) are supported by different cryptographic operations.

Lines in grey correspond to unsupported mechanisms.

	Functions						
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR	Digest	Gen. Key/Key Pair	Wrap & Unwrap	Derive
CKM_RSA_PKCS_KEY_PAIR_GEN					✓		
CKM_RSA_X9_31_KEY_PAIR_GEN					✓		

	Functions						
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR	Digest	Gen. Key/Key Pair	Wrap & Unwrap	Derive
CKM_RSA_PKCS	✓[2]	✓[2]	✓			✓	
CKM_RSA_PKCS_OAEP	✓[2]					✓	
CKM_RSA_PKCS_PSS		✓[2]					
CKM_RSA_9796		✓[2]	✓				
CKM_RSA_X_509	✓[2]	✓[2]	✓			✓**	
CKM_RSA_X9_31		✓[2]					
CKM_MD2_RSA_PKCS		✓					
CKM_MD5_RSA_PKCS		✓					
CKM_SHA1_RSA_PKCS		✓					
CKM_SHA256_RSA_PKCS		✓					
CKM_SHA384_RSA_PKCS		✓					
CKM_SHA512_RSA_PKCS		✓					
CKM_RIPEMD128_RSA_PKCS		✓					
CKM_RIPEMD160_RSA_PKCS		✓					
CKM_SHA1_RSA_PKCS_PSS		✓					
CKM_SHA256_RSA_PKCS_PSS		✓					
CKM_SHA384_RSA_PKCS_PSS		✓					
CKM_SHA512_RSA_PKCS_PSS		✓					
CKM_SHA1_RSA_X9_31		✓					
CKM_DSA_KEY_PAIR_GEN					✓		
CKM_DSA_PARAMETER_GEN					✓		
CKM_DSA		✓[2]					
CKM_DSA_SHA1		✓					
CKM_FORTEZZA_TIMESTAMP		✓[2]					
CKM_EC_KEY_PAIR_GEN (CKM_ECDSA_KEY_PAIR_GEN)					✓		
CKM_ECDSA		✓[2]					
CKM_ECDSA_SHA1		✓					

	Functions						
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR	Digest	Gen. Key/Key Pair	Wrap & Unwrap	Derive
CKM_ECDH1_DERIVE							✓
CKM_ECDH1_COFACTOR_DERIVE							✓
CKM_ECMQV_DERIVE							✓
CKM_DH_PKCS_KEY_PAIR_GEN					✓		
CKM_DH_PKCS_PARAMETER_GEN					✓		
CKM_DH_PKCS_DERIVE							✓
CKM_X9_42_DH_KEY_PAIR_GEN					✓		
CKM_X9_42_DH_PKCS_PARAMETER_GEN					✓		
CKM_X9_42_DH_DERIVE							✓
CKM_X9_42_DH_HYBRID_DERIVE							✓
CKM_X9_42_MQV_DERIVE							✓
CKM_KEA_KEY_PAIR_GEN					✓		
CKM_KEA_KEY_DERIVE							✓
CKM_GENERIC_SECRET_KEY_GEN					✓		
CKM_RC2_KEY_GEN					✓		
CKM_RC2_ECB	✓					✓	
CKM_RC2_CBC	✓					✓	
CKM_RC2_CBC_PAD	✓					✓	
CKM_RC2_MAC_GENERAL		✓					
CKM_RC2_MAC		✓					
CKM_RC4_KEY_GEN					✓		
CKM_RC4	✓						
CKM_RC5_KEY_GEN					✓		
CKM_RC5_ECB	✓					✓	
CKM_RC5_CBC	✓					✓	

	Functions						
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR	Digest	Gen. Key/Key Pair	Wrap & Unwrap	Derive
CKM_RC5_CBC_PAD	✓					✓	
CKM_RC5_MAC_GENERAL		✓					
CKM_RC5_MAC		✓					
CKM_AES_KEY_GEN***					✓		
CKM_AES_ECB***	✓					✓	
CKM_AES_CBC***	✓					✓	
CKM_AES_CBC_PAD***	✓					✓	
CKM_AES_MAC_GENERAL***		✓					
CKM_AES_MAC***		✓					
CKM_DES_KEY_GEN***					✓		
CKM_DES_ECB***	✓					✓	
CKM_DES_CBC***	✓					✓	
CKM_DES_CBC_PAD***	✓					✓	
CKM_DES_MAC_GENERAL***		✓					
CKM_DES_MAC***		✓					
CKM_DES2_KEY_GEN***					✓		
CKM_DES3_KEY_GEN***					✓		
CKM_DES3_ECB***	✓					✓	
CKM_DES3_CBC***	✓					✓	
CKM_DES3_CBC_PAD***	✓					✓	
CKM_DES3_MAC_GENERAL***		✓					
CKM_DES3_MAC***		✓					
CKM_CAST_KEY_GEN					✓		
CKM_CAST_ECB	✓					✓	
CKM_CAST_CBC	✓					✓	
CKM_CAST_CBC_PAD	✓					✓	
CKM_CAST_MAC_GENERAL		✓					
CKM_CAST_MAC		✓					

	Functions						
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR	Digest	Gen. Key/Key Pair	Wrap & Unwrap	Derive
CKM_CAST3_KEY_GEN					✓		
CKM_CAST3_ECB	✓					✓	
CKM_CAST3_CBC	✓					✓	
CKM_CAST3_CBC_PAD	✓					✓	
CKM_CAST3_MAC_GENERAL		✓					
CKM_CAST3_MAC		✓					
CKM_CAST128_KEY_GEN (CKM_CAST5_KEY_GEN)					✓		
CKM_CAST128_ECB (CKM_CAST5_ECB)	✓					✓	
CKM_CAST128_CBC (CKM_CAST5_CBC)	✓					✓	
CKM_CAST128_CBC_PAD (CKM_CAST5_CBC_PAD)	✓					✓	
CKM_CAST128_MAC_GENERAL (CKM_CAST5_MAC_GENERAL)		✓					
CKM_CAST128_MAC (CKM_CAST5_MAC)		✓					
CKM_IDEA_KEY_GEN					✓		
CKM_IDEA_ECB	✓					✓	
CKM_IDEA_CBC	✓					✓	
CKM_IDEA_CBC_PAD	✓					✓	
CKM_IDEA_MAC_GENERAL		✓					
CKM_IDEA_MAC		✓					
CKM_CDMF_KEY_GEN					✓		
CKM_CDMF_ECB	✓					✓	
CKM_CDMF_CBC	✓					✓	
CKM_CDMF_CBC_PAD	✓					✓	
CKM_CDMF_MAC_GENERAL		✓					
CKM_CDMF_MAC		✓					

	Functions						
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR	Digest	Gen. Key/Key Pair	Wrap & Unwrap	Derive
CKM_DES_ECB_ENCRYPT_DATA							✓
CKM_DES_CBC_ENCRYPT_DATA							✓
CKM_DES3_ECB_ENCRYPT_DATA							✓
CKM_DES3_CBC_ENCRYPT_DATA							✓
CKM_AES_ECB_ENCRYPT_DATA							✓
CKM_AES_CBC_ENCRYPT_DATA							✓
CKM_SKIPJACK_KEY_GEN					✓		
CKM_SKIPJACK_ECB64	✓						
CKM_SKIPJACK_CBC64	✓						
CKM_SKIPJACK_OFB64	✓						
CKM_SKIPJACK_CFB64	✓						
CKM_SKIPJACK_CFB32	✓						
CKM_SKIPJACK_CFB16	✓						
CKM_SKIPJACK_CFB8	✓						
CKM_SKIPJACK_WRAP						✓	
CKM_SKIPJACK_PRIVATE_WRAP						✓	
CKM_SKIPJACK_RELAYX						✓[3]	
CKM_BATON_KEY_GEN					✓		
CKM_BATON_ECB128	✓						
CKM_BATON_ECB96	✓						
CKM_BATON_CBC128	✓						
CKM_BATON_COUNTER	✓						
CKM_BATON_SHUFFLE	✓						
CKM_BATON_WRAP						✓	
CKM_JUNIPER_KEY_GEN					✓		
CKM_JUNIPER_ECB128	✓						
CKM_JUNIPER_CBC128	✓						

	Functions						
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR	Digest	Gen. Key/Key Pair	Wrap & Unwrap	Derive
CKM_JUNIPER_COUNTER	✓						
CKM_JUNIPER_SHUFFLE	✓						
CKM_JUNIPER_WRAP						✓	
CKM_MD2				✓			
CKM_MD2_HMAC_GENERAL		✓					
CKM_MD2_HMAC		✓					
CKM_MD2_KEY_DERIVATION							✓
CKM_MD5				✓			
CKM_MD5_HMAC_GENERAL		✓					
CKM_MD5_HMAC		✓					
CKM_MD5_KEY_DERIVATION							✓
CKM_SHA_1				✓			
CKM_SHA_1_HMAC_GENERAL		✓					
CKM_SHA_1_HMAC		✓					
CKM_SHA1_KEY_DERIVATION							✓
CKM_SHA256				✓			
CKM_SHA256_HMAC_GENERAL		✓					
CKM_SHA256_HMAC		✓					
CKM_SHA256_KEY_DERIVATION							✓
CKM_SHA384				✓			
CKM_SHA384_HMAC_GENERAL		✓					
CKM_SHA384_HMAC		✓					
CKM_SHA384_KEY_DERIVATION							✓
CKM_SHA512				✓			
CKM_SHA512_HMAC_GENERAL		✓					
CKM_SHA512_HMAC		✓					
CKM_SHA512_KEY_DERIVATION							✓
CKM_RIPEMD128				✓			

	Functions						
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR	Digest	Gen. Key/Key Pair	Wrap & Unwrap	Derive
CKM_RIPEMD128_HMAC_GENERAL		✓					
CKM_RIPEMD128_HMAC		✓					
CKM_RIPEMD160				✓			
CKM_RIPEMD160_HMAC_GENERAL		✓					
CKM_RIPEMD160_HMAC		✓					
CKM_FASTHASH				✓			
CKM_PBE_MD2_DES_CBC					✓		
CKM_PBE_MD5_DES_CBC					✓		
CKM_PBE_MD5_CAST_CBC					✓		
CKM_PBE_MD5_CAST3_CBC					✓		
CKM_PBE_MD5_CAST128_CBC (CKM_PBE_MD5_CAST5_CBC)					✓		
CKM_PBE_SHA1_CAST128_CBC (CKM_PBE_SHA1_CAST5_CBC)					✓		
CKM_PBE_SHA1_RC4_128					✓		
CKM_PBE_SHA1_RC4_40					✓		
CKM_PBE_SHA1_DES3_EDE_CBC					✓		
CKM_PBE_SHA1_DES2_EDE_CBC					✓		
CKM_PBE_SHA1_RC2_128_CBC					✓		
CKM_PBE_SHA1_RC2_40_CBC					✓		
CKM_PBA_SHA1_WITH_SHA1_HMAC					✓		
CKM_PKCS5_PBKD2					✓		
CKM_KEY_WRAP_SET_OAEP						✓	
CKM_KEY_WRAP_LYNKS						✓	
CKM_SSL3_PRE_MASTER_KEY_GEN					✓		
CKM_SSL3_MASTER_KEY_DERIVE							✓

	Functions						
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR	Digest	Gen. Key/Key Pair	Wrap & Unwrap	Derive
CKM_SSL3_MASTER_KEY_DERIVE_DH							✓
CKM_SSL3_KEY_AND_MAC_DERIVE							✓
CKM_SSL3_MD5_MAC		✓					
CKM_SSL3_SHA1_MAC		✓					
CKM_TLS_PRE_MASTER_KEY_GEN					✓		
CKM_TLS_MASTER_KEY_DERIVE							✓
CKM_TLS_MASTER_KEY_DERIVE_DH							✓
CKM_TLS_KEY_AND_MAC_DERIVE							✓
CKM_TLS_PRF							✓
CKM_WTLS_PRE_MASTER_KEY_GEN					✓		
CKM_WTLS_MASTER_KEY_DERIVE							✓
CKM_WTLS_MASTER_KEY_DERIVE_DH_ECC							✓
CKM_WTLS_SERVER_KEY_AND_MAC_DERIVE							✓
CKM_WTLS_CLIENT_KEY_AND_MAC_DERIVE							✓
CKM_WTLS_PRF							✓
CKM_CMS_SIG		✓	✓				
CKM_CONCATENATE_BASE_AND_KEY							✓
CKM_CONCATENATE_BASE_AND_DATA							✓
CKM_CONCATENATE_DATA_AND_BASE							✓
CKM_XOR_BASE_AND_DATA							✓

	Functions						
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR	Digest	Gen. Key/Key Pair	Wrap & Unwrap	Derive
CKM_EXTRACT_KEY_FROM_KEY							✓
CKM_ACTI		✓					

** Only secret keys can be unwrapped AES, DES… The key length should be < k where k is the length of the RSA key.

*** Secret Key mechanisms (AES, DES, and DES3) are only supported with cryptography in software not on the smart card.

9.0 Objects

9.1 Certificate Object

9.1.1 X.509 Certificate Object Attributes

Attribute	Data type	Meaning
CKA_SUBJECT	Byte array	DER-encoding of the certificate subject name This attribute must be specified when the object is created.
CKA_ID	Byte array	Key identifier for public/private key pair (default empty) This attribute may be modified after the object is created.
CKA_ISSUER	Byte array	DER-encoding of the certificate issuer name (default empty) This attribute may be modified after the object is created.
CKA_SERIAL_NUMBER	Byte array	DER-encoding of the certificate serial number (default empty) This attribute may be modified after the object is created.
CKA_VALUE	Byte array	BER-encoding of the certificate This attribute must be specified when the object is created.
CKA_START_DATE	CK_DATE	Start date for the certificate, issuance date (default empty)
CKA_END_DATE	CK_DATE	End date for the certificate, expiration date (default empty)

9.1.2 Common Certificate Object Attributes

Attribute	Data Type	Meaning
CKA_CERTIFICATE_TYPE	CK_CERTIFICATE_TYPE	Type of certificate. Must be specified when object is created with C_CreateObject. Typical value is “X.509 public key certificate”
CKA_CERTIFICATE_CATEGORY	CK_ULONG	Categorization of the certificate. Typical values are:

Attribute	Data Type	Meaning
		1 for user certificate 2 for authority certificate.

9.1.3 Common Storage Object Attributes

Attribute	Data Type	Meaning
CKA_TOKEN	CK_BBOOL	CK_TRUE if object is a token object; CK_FALSE if object is a session object. Default is CK_FALSE.
CKA_PRIVATE	CK_BBOOL	CK_TRUE if object is a private object; CK_FALSE if object is a public object. Default value is token-specific, and may depend on the values of other attributes of the object.
CKA_MODIFIABLE	CK_BBOOL	CK_TRUE if object can be modified Default is CK_TRUE.
CKA_LABEL	RFC2279 string	Description of the object (default empty). This attribute may be modified after the object is created.
CKA_CLASS	CK_OBJECT_CLASS	Object class (type). Must be specified when object is created with C_CreateObject.

9.1.4 Vendor Attributes

Attribute	Data Type	Meaning
CKA_CONTAINER_NAME	Byte Array	Certificate container Name This attribute is not supported in this version.
CKA_EXTENDED_KEY_USAGE_SMARTCARD_LOGON	CK_BBOOL	CK_TRUE if the object is a certificate that can be used for PKI Logon (Extended Key Usage contains usage for Smartcard Logon and Subject Alternative Name contains Other name with UPN type and Key Usage contains Digital Signature) This attribute is not supported in this version.
CKA_CERTIFICATE_FASCN	Byte Array	FASC-N from Certificate.(PIV certificates only)
CKA_CERTIFICATE_UUID	Byte Array	UUID from Certificate.(PIV-I certificates only)

Attribute	Data Type	Meaning
CKA_SUBJECT_ALTERNATE_NAME	Byte Array	DER-encoding of the certificate alternate subject name This attribute is not supported in this version.
CKA_CERTIFICATE_SIGNATURE_STATUS	CK_FLAG	Certificate signature status. See 9.4 Signature Status Flag Definition on page 40 for possible values. This attribute is not supported in this version.

9.2 RSA Public Key Object

9.2.1 RSA Public Key Object Attributes

Attribute	Data type	Meaning
CKA_MODULUS	Big integer	Modulus n Must be specified when object is created with C_CreateObject. Must not be specified when object is generated with C_GenerateKey or C_GenerateKeyPair
CKA_MODULUS_BITS	CK_ULONG	Length in bits of modulus n Must not be specified when object is created with C_CreateObject. Must be specified when object is generated with C_GenerateKey or C_GenerateKeyPair.
CKA_PUBLIC_EXPONENT	Big integer	Public exponent e Must be specified when object is created with C_CreateObject.

9.2.2 Common Public Key Attributes

Attribute	Data Type	Meaning
CKA_SUBJECT	Byte Array	DER-encoding of the key subject name (default empty)
CKA_ENCRYPT	CK_BBOOL	CK_TRUE if key supports encryption
CKA_VERIFY	CK_BBOOL	CK_TRUE if key supports verification where the signature is an appendix to the data

Attribute	Data Type	Meaning
CKA_VERIFYRECOVER	CK_BBOOL	CK_TRUE if key supports verification where the data is recovered from the signature
CKA_WRAP	CK_BBOOL	CK_TRUE if key supports wrapping (i.e., can be used to wrap other keys)
CKA_TRUSTED	CK_BBOOL	The key can be trusted for the application that it was created. The wrapping key can be used to wrap keys with CKA_WRAP_WITH_TRUSTED set to CK_TRUE.
CKA_ALLOWED_MECHANISMS	CK_ATTRIBUTE_PTR	For wrapping keys. The attribute template to match against any keys wrapped using this wrapping key. Keys that do not match cannot be wrapped. The number of attributes in the array is the ulValueLen component of the attribute divided by the size of CK_ATTRIBUTE.

9.2.3 Common Key Attributes

Attribute	Data Type	Meaning
CKA_KEY_TYPE	CK_KEY_TYPE	Type of key Must be specified when object is created with C_CreateObject.
CKA_ID	Byte array	Key identifier for key (default empty) 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 during the course of a C_CopyObject call.
CKA_LOCAL	CK_BBOOL	CK_TRUE only if key was either generated locally (i.e., on the token) with a C_GenerateKey or C_GenerateKeyPair call created with a C_CopyObject call as a copy of a key which had its CKA_LOCAL attribute set to CK_TRUE Must not be specified when object is created with C_CreateObject. Must not be specified when object is generated with C_GenerateKey or C_GenerateKeyPair.

Attribute	Data Type	Meaning
CKA_KEY_GEN_MECHANISM	CK_MECHANISM_TYPE	Identifier of the mechanism used to generate the key material. Must not be specified when object is created with C_CreateObject. Must not be specified when object is generated with C_GenerateKey or C_GenerateKeyPair.
CKA_ALLOWED_MECHANISMS	CK_MECHANISM_TYPE _PTR, pointer to a CK_MECHANISM_TYPE array	A list of mechanisms allowed to be used with this key. The number of mechanisms in the array is the ulValueLen component of the attribute divided by the size of CK_MECHANISM_TYPE.

9.2.4 Common Storage Object Attributes

Attribute	Data Type	Meaning
CKA_TOKEN	CK_BBOOL	CK_TRUE if object is a token object; CK_FALSE if object is a session object. Default is CK_FALSE.
CKA_PRIVATE	CK_BBOOL	CK_TRUE if object is a private object; CK_FALSE if object is a public object. Default value is token-specific, and may depend on the values of other attributes of the object.
CKA_MODIFIABLE	CK_BBOOL	CK_TRUE if object can be modified Default is CK_TRUE.
CKA_LABEL	RFC2279 string	Description of the object (default empty). This attribute may be modified after the object is created.
CKA_CLASS	CK_OBJECT_CLASS	Object class (type). Must be specified when object is created with C_CreateObject.

9.3 RSA Private Key Object

9.3.1 RSA Private Key Object Attributes

Attribute	Data type	Meaning
CKA_MODULUS	Big integer	Modulus n Must be specified when object is created with C_CreateObject. Must not be specified when object is generated with C_GenerateKey or C_GenerateKeyPair.
CKA_PUBLIC_EXPONENT	Big integer	Public exponent e Must not be specified when object is generated with C_GenerateKey or C_GenerateKeyPair.
CKA_PRIVATE_EXPONENT	Big integer	Private exponent d Must be specified when object is created with C_CreateObject. Must not be specified when object is generated with C_GenerateKey or C_GenerateKeyPair. Cannot be revealed
CKA_PRIME_1	Big integer	Prime p Must not be specified when object is generated with C_GenerateKey or C_GenerateKeyPair. Cannot be revealed
CKA_PRIME_2	Big integer	Prime q Must not be specified when object is generated with C_GenerateKey or C_GenerateKeyPair. Cannot be revealed
CKA_EXPONENT_1	Big integer	Private exponent d modulo p-1 Must not be specified when object is generated with C_GenerateKey or C_GenerateKeyPair. Cannot be revealed
CKA_EXPONENT_2	Big integer	Private exponent d modulo q-1 Must not be specified when object is generated with C_GenerateKey or C_GenerateKeyPair. Cannot be revealed
CKA_COEFFICIENT	Big integer	CRT coefficient q-1 mod p Must not be specified when object is generated with C_GenerateKey or C_GenerateKeyPair. Cannot be revealed

9.3.2 Common Private Key Attributes

Attribute	Data type	Meaning
CKA_SUBJECT	Byte array	DER-encoding of certificate subject name (default empty) 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 during the course of a C_CopyObject call.
CKA_SENSITIVE	CK_BBOOL	CK_TRUE
CKA_DECRYPT	CK_BBOOL	CK_TRUE if key supports decryption 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 during the course of a C_CopyObject call. Default value is token-specific, and may depend on the values of other attributes
CKA_SIGN	CK_BBOOL	CK_TRUE if key supports signatures where the signature is an appendix to the data 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 during the course of a C_CopyObject call. Default value is token-specific, and may depend on the values of other attributes
CKA_EXTRACTABLE	CK_BBOOL	CK_FALSE
CKA_ALWAYS_SENSITIVE	CK_BBOOL	CK_TRUE
CKA_NEVER_EXTRACTABLE	CK_BBOOL	CK_TRUE
CKA_ALWAYS_AUTHENTICATE	CK_BBOOL	If CK_TRUE, the user has to supply the PIN for each use (sign or decrypt) with the key. Default is CK_FALSE.

9.3.3 Common Key Attributes

Attribute	Data Type	Meaning
CKA_KEY_TYPE	CK_KEY_TYPE	Type of key Must be specified when object is created with C_CreateObject.
CKA_ID	Byte array	Key identifier for key (default empty) 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 during the course of a C_CopyObject call.
CKA_LOCAL	CK_BBOOL	CK_TRUE only if key was either generated locally (i.e., on the token) with a C_GenerateKey or C_GenerateKeyPair call created with a C_CopyObject call as a copy of a key which had its CKA_LOCAL attribute set to CK_TRUE Must not be specified when object is created with C_CreateObject. Must not be specified when object is generated with C_GenerateKey or C_GenerateKeyPair.
CKA_KEY_GEN_MECHANISM	CK_MECHANISM_TYPE	Identifier of the mechanism used to generate the key material. Must not be specified when object is created with C_CreateObject. Must not be specified when object is generated with C_GenerateKey or C_GenerateKeyPair
CKA_ALLOWED_MECHANISMS	CK_MECHANISM_TYPE _PTR, pointer to a CK_MECHANISM_TYPE array	A list of mechanisms allowed to be used with this key. The number of mechanisms in the array is the ulValueLen component of the attribute divided by the size of CK_MECHANISM_TYPE.

9.3.4 Common Storage Object Attributes

Attribute	Data Type	Meaning
CKA_TOKEN	CK_BBOOL	CK_TRUE if object is a token object; CK_FALSE if object is a session object. Default is CK_FALSE.
CKA_PRIVATE	CK_BBOOL	CK_TRUE if object is a private object; CK_FALSE if object is a public object. Default value is token-specific,

Attribute	Data Type	Meaning
		and may depend on the values of other attributes of the object.
CKA_MODIFIABLE	CK_BBOOL	CK_TRUE if object can be modified Default is CK_TRUE.
CKA_LABEL	RFC2279 string	Description of the object (default empty). This attribute may be modified after the object is created
CKA_CLASS	CK_OBJECT_CLASS	Object class (type). Must be specified when object is created with C_CreateObject.

9.3.5 Vendor Attributes

Attribute	Data Type	Meaning
CKA_CONTAINER_NAME	Byte Array	Certificate container Name This attribute is not supported in this version
CKA_EXTENDED_KEY_USAGE_SMARTCARD_LOGON	CK_BBOOL	CK_TRUE if the object is a certificate that can be used for PKI Logon (Extended Key Usage contains usage for Smartcard Logon and Subject Alternative Name contains Othername with UPN type and Key Usage contains Digital Signature))

9.4 Signature Status Flag Definition

```
/* Certificate status possible values */
#define CKF_ACTIVE                      0x00000001
#define CKF_REVOKED                     0x00000002
#define CKF_EXPIRED                     0x00000004
#define CKF_TRUSTED                     0x00000008

/* Certificate revocation reasons */
#define CKF_REASON_UNSPECIFIED              0x00000010
#define CKF_REASON_KEY_COMPROMISED          0x00000020
#define CKF_REASON_CA_COMPROMISED           0x00000040
#define CKF_REASON_AFFILIATION_CHANGED      0x00000080
#define CKF_REASON_SUPERSEDED               0x00000100
#define CKF_REASON_CESSATION_OF_OPERATION   0x00000200
#define CKF_REASON_CERTIFICATE_HOLD         0x00000400
```

```
/* Certificate revocation status including reasons */
#define CKF_REVOKED_UNSPECIFIED (CKF_REVOKED | CKF_REASON_UNSPECIFIED)
#define CKF_REVOKED_KEY_COMPROMISED CKF_REVOKED |CKF_REASON_KEY_COMPROMISED)
#define CKF_REVOKED_CA_COMPROMISED (CKF_REVOKED | CKF_REASON_CA_COMPROMISED)
#define CKF_REVOKED_AFFILIATION_CHANGED (CKF_REVOKED | CKF_REASON_AFFILIATION_CHANGED)
#define CKF_REVOKED_SUPERSEDED (CKF_REVOKED | CKF_REASON_SUPERSEDED)
#define CKF_REVOKED_CESSATION_OF_OPERATION (CKF_REVOKED | CKF_REASON_CESSATION_OF_OPERATION)
#define CKF_REVOKED_CERTIFICATE_HOLD (CKF_REVOKED | CKF_REASON_CERTIFICATE_HOLD)
```

9.5 Access to PIV Object

9.5.1 Common Storage Object Attributes

Attribute	Data Type	Meaning
CKA_TOKEN	CK_BBOOL	CK_TRUE.
CKA_PRIVATE	CK_BBOOL	CK_TRUE if object is a private object; CK_FALSE if object is a public object. Default value is token-specific, and may depend on the values of other attributes of the object.
CKA_MODIFIABLE	CK_BBOOL	CK_FALSE.
CKA_LABEL	RFC2279 string	Description of the object (default empty). This attribute may be modified after the object is created
CKA_CLASS	CK_OBJECT_CLASS	Object class (type). Must be specified when object is created with C_CreateObject.
CKA_OBJECT_ID	Byte Array	DER-encoding of the object identifier indicating the data object type (default OID of the corresponding PIV data object).

9.5.2 PIV Objects Attributes

The different PIV objects are defined as follows:

Attribute	Data Type	Meaning
CKO_CHUID	CK_OBJECT_CLASS	CHUID Object
CKO_FACIAL_IMAGE	CK_OBJECT_CLASS	Facial Image Object

Attribute	Data Type	Meaning
CKO_FINGERPRINTS	CK_OBJECT_CLASS	Fingerprints Object
CKO_PRINTED_INFO	CK_OBJECT_CLASS	Printed Info Object
CKO_SECURITY_OBJECT	CK_OBJECT_CLASS	Security Object
CKO_DISCOVERY_OBJECT	CK_OBJECT_CLASS	Discovery Object
CKO_KEY_HISTORY_OBJECT	CK_OBJECT_CLASS	Key History Object
CKO_IRIS_IMAGE	CK_OBJECT_CLASS	Iris Image Object
CKO_CCC	CK_OBJECT_CLASS	CCC Object

The CKO_OBJECT_CLASS attributes are defined as follows:

```
#define CKO_ACTI                (CKO_VENDOR_DEFINED + 0x800)

#define CKO_CHUID               (CKO_ACTI + 0)
#define CKO_FACIAL_IMAGE        (CKO_ACTI + 1)
#define CKO_FINGERPRINTS        (CKO_ACTI + 2)
#define CKO_PRINTED_INFO        (CKO_ACTI + 3)
#define CKO_SECURITY_OBJECT     (CKO_ACTI + 4)
#define CKO_DISCOVERY_OBJECT    (CKO_ACTI + 5)
#define CKO_CCC                 (CKO_ACTI + 6)
#define CKO_KEY_HISTORY_OBJECT  (CKO_ACTI + 7)
#define CKO_IRIS_IMAGE          (CKO_ACTI + 8)
```

9.5.2.1 CHUID Object Attributes

- Data Object Attributes

Attribute	Data Type	Meaning / Value
CKA_LABEL	RFC2279 string	“CHUID Object”
CKA_APPLICATION	RFC2279 string	“PIV Card Application”
CKA_VALUE	Byte array	CHUID Raw data

- Vendor Attributes

Attribute	Data Type	Meaning / Value
CKA_CHUID_GUID	Byte array	GUID value
CKA_CHUID_EXPIRATION_DATE	CK_DATE	CHUID expiration date
CKA_CHUID_OPT_AGENCY	CK_ULONG	Organizational Agency code
CKA_CHUID_OPT_ORGANIZATION	CK_ULONG	Organization code
CKA_CHUID_OPT_DUNS	Byte array	DUNS value
CKA_CHUID_FASCN	Byte array	FASC-N value
CKA_CHUID_FASCN_AGENCY	RFC2279 String	Agency code of the FASC-N
CKA_CHUID_FASCN_SYSTEM	RFC2279 String	System code of the FASC-N
CKA_CHUID_FASCN_CREDENTIAL_NUMBER	RFC2279 String	Credential number of the FASC-N
CKA_CHUID_FASCN_CREDENTIAL_SERIES	RFC2279 String	Credential series of the FASC-N
CKA_CHUID_FASCN_INDIVIDUAL_CREDENTIAL_ISSUE	RFC2279 String	Individual credential issue of the FASC-N
CKA_CHUID_FASCN_PERSON_IDENTIFIER	RFC2279 String	Person Identifier of the FASC-N
CKA_CHUID_FASCN_ORGANIZATIONAL_CATEGORY	RFC2279 String	Organizational category code of the FASC-N
CKA_CHUID_FASCN_ORGANIZATIONAL_IDENTIFIER	RFC2279 String	Organizational Identifier code of the FASC-N
CKA_CHUID_FASCN_PERSON_ASSOCIATION_CATEGORY	RFC2279 String	Person association category value of the FASC-N
CKA_CHUID_SIGNATURE	Byte array	CHUID signature
CKA_CHUID_SIGNATURE_STATUS	CK_ULONG	CHUID signature status. See 9.4 Signature Status Flag Definition on page 40 for possible values.
CKA_CHUID_SIGNER_CERTIFICATE	Byte array	DER-encoded CHUID Signer certificate value

The CHUID Object Vendor attributes are enumerated as follows:

```
Enum PivCHUIDAttributes
{
CKA_CHUID_GUID = (CKA_VENDOR_DEFINED+0x200),
CKA_CHUID_EXPIRATION_DATE,
CKA_CHUID_OPT_AGENCY,
CKA_CHUID_OPT_ORGANIZATION,
CKA_CHUID_OPT_DUNS,
CKA_CHUID_FASCN,
CKA_CHUID_FASCN_AGENCY,
CKA_CHUID_FASCN_SYSTEM,
CKA_CHUID_FASCN_CREDENTIAL_NUMBER,
CKA_CHUID_FASCN_CREDENTIAL_SERIES,
CKA_CHUID_FASCN_INDIVIDUAL_CREDENTIAL_ISSUE,
CKA_CHUID_FASCN_PERSON_IDENTIFIER,
CKA_CHUID_FASCN_ORGANIZATIONAL_CATEGORY,
CKA_CHUID_FASCN_ORGANIZATIONAL_IDENTIFIER,
CKA_CHUID_FASCN_PERSON_ASSOCIATION_CATEGORY,
CKA_CHUID_SIGNATURE,
CKA_CHUID_SIGNATURE_STATUS,
CKA_CHUID_SIGNER_CERTIFICATE
};
```

9.5.2.2 Facial Image Object Attributes

- Data Object Attributes

Attribute	Data Type	Meaning / Value
CKA_LABEL	RFC2279 string	“Facial Image Object”
CKA_APPLICATION	RFC2279 string	“PIV Card Application”
CKA_VALUE	Byte array	Facial Image Raw data

- Vendor attributes

Attribute	Data Type	Meaning
CKA _FACIAL_IMAGE_CBEFF_HEADER	Byte array	CBEFF Header of the Facial image buffer
CKA_FACIAL_IMAGE_CBEFF_RECORD	Byte array	Facial Image CBEFF record value.
CKA_FACIAL_IMAGE_CBEFF_SIGNATURE	Byte array	Facial image signature
CKA_FACIAL_IMAGE_SIGNATURE_STATUS	CK_ULONG	Facial image signature status See 9.4 Signature Status Flag Definition on page 40 for possible values.
CKA_FACIAL_IMAGE_SIGNER_CERTIFICATE	Byte array	DER encoded value of Facial Image Signer certificate value
CKA_FACIAL_IMAGE_FASCN	Byte array	FASC-N value extracted from facial image signature(PIV only)
CKA_FACIAL_IMAGE_UUID	Byte array	UUID value extracted from facial image signature (PIV-I only)

The Facial Image Object Vendor attributes are enumerated as follows:

```
Enum PivFacialImageAttributes
{
CKA_FACIAL_IMAGE_CBEFF_HEADER = (CKA_VENDOR_DEFINED+0x230),
CKA_FACIAL_IMAGE_CBEFF_RECORD,
CKA_FACIAL_IMAGE_CBEFF_SIGNATURE
CKA_FACIAL_IMAGE_SIGNATURE_STATUS,
CKA_FACIAL_IMAGE_SIGNER_CERTIFICATE,
 CKA_ FACIAL_IMAGE_FASCN,
CKA_ FACIAL_IMAGE_UUID
};
```

9.5.2.3 Fingerprints Object Attributes

- Data Object Attributes

Attribute	Data Type	Meaning / Value
CKA_LABEL	RFC2279 string	“Fingerprints Object”
CKA_APPLICATION	RFC2279 string	“PIV Card Application”
CKA_VALUE	Byte array	Fingerprints Raw data

- Vendor Attributes

Attribute	Data Type	Meaning
CKA _FINGERPRINTS_CBEFF_HEADER	Byte array	Fingerprints CBEFF Header value
CKA _FINGERPRINTS_CBEFF_RECORD	Byte array	Fingerprints CBEFF record value
CKA_FINGERPRINTS_CBEFF_SIGNATURE	Byte array	Fingerprints CBEFF signature
CKA_FINGERPRINTS_SIGNATURE_STATUS	CK_ULONG	Fingerprint signature status. See 9.4 Signature Status Flag Definition on page 40 for possible values.
CKA_FINGERPRINTS_SIGNER_CERTIFICATE	Byte array	DER encoded value of Signer certificate value of fingerprint buffer
CKA_FINGERPRINTS_FASCN	Byte array	FASC-N value extracted from fingerprints signature(PIV only)
CKA_FINGERPRINTS_UUID	Byte array	UUID value extracted from fingerprints signature (PIV-I only)

The Fingerprints Object Vendor attributes are enumerated as follows:

```
Enum PivFingerprintsAttributes
{
CKA _FINGERPRINTS_CBEFF_RECORD = (CKA_VENDOR_DEFINED+0x260),
CKA _FINGERPRINTS_CBEFF_RECORD,
CKA_FINGERPRINTS_CBEFF_SIGNATURE,
CKA_FINGERPRINTS_SIGNATURE_STATUS,
```

```
CKA_FINGERPRINTS_SIGNER_CERTIFICATE,
 CKA_FINGERPRINTS_FASCN,
CKA_FINGERPRINTS_UUID
};
```

9.5.2.4 Printed Info Object Attributes

- Data Object Attributes

Attribute	Data Type	Meaning/ Value
CKA_LABEL	RFC2279 string	“Printed Info Object”
CKA_APPLICATION	RFC2279 string	“PIV Card Application”
CKA_VALUE	Byte array	Printed Info Raw data

- Vendor Attributes

Attribute	Data Type	Meaning
CKA _PRINTED_INFORMATION_NAME	RFC2279 string	Name as printed on card
CKA_PRINTED_INFORMATION_EMPLOYEE_INFORMATION_LINE1	RFC2279 string	Information: line 1 as printed on card
CKA_PRINTED_INFORMATION_EMPLOYEE_INFORMATION_LINE2	RFC2279 string	Information: line 2 as printed on card
CKA_PRINTED_INFORMATION_EXPIRATION_DATE	CK_DATE	Card validity date as printed on card
CKA_PRINTED_INFORMATION_AGENCY_CARD_SERIAL_NUMBER	RFC2279 string	Agency card serial number as printed on card
CKA_PRINTED_INFORMATION_ISSUER_IDENTIFICATION	RFC2279 string	Issuer identification as printed on card
CKA_PRINTED_INFORMATION_ORGANIZATION_AFFILIATION_LINE1	RFC2279 string	Organization affiliation line 1 as printed on card
CKA_PRINTED_INFORMATION_ORGANIZATION_AFFILIATION_LINE2	RFC2279 string	Organization affiliation line 2 as printed on card

The Printed Info Object Vendor attributes are enumerated as follows:

```
Enum PivPrintedInfoAttributes
{
```

```
CKA _PRINTED_INFORMATION_NAME = (CKA_VENDOR_DEFINED+0x290),
CKA_PRINTED_INFORMATION_EMPLOYEE_INFORMATION_LINE1,
CKA_PRINTED_INFORMATION_EMPLOYEE_INFORMATION_LINE2,
CKA_PRINTED_INFORMATION_EXPIRATION_DATE,
CKA_PRINTED_INFORMATION_AGENCY_CARD_SERIAL_NUMBER,
CKA_PRINTED_INFORMATION_ISSUER_IDENTIFICATION,
CKA_PRINTED_INFORMATION_ORGANIZATION_AFFILIATION_LINE1,
CKA_PRINTED_INFORMATION_ORGANIZATION_AFFILIATION_LINE2
};
```

9.5.2.5 Security Object Attributes

- Data Object Attributes

Attribute	Data Type	Meaning / Value
CKA_LABEL	RFC2279 string	“Security Object”
CKA_APPLICATION	RFC2279 string	“PIV Card Application”
CKA_VALUE	Byte array	Security Raw data

- Vendor Attributes

Attribute	Data Type	Meaning
CKA_SECURITY_OBJECT_MAPPING_OF_DG_TO_CONTAINERID	Byte array	“DG-number-to-Container-ID” mapping object TLV
CKA_SECURITY_OBJECT	Byte array	Security object itself

The Security Object Vendor attributes are enumerated as follows:

```
Enum PivSecurityAttributes
{
CKA_SECURITY_OBJECT_MAPPING_OF_DG_TO_CONTAINERID = (CKA_VENDOR_DEFINED+0x2C0),
CKA_SECURITY_OBJECT
};
```

9.5.2.6 Discovery Object Attributes (SP800-73-2)

- Data Object Attributes

Attribute	Data Type	Meaning / Value
CKA_LABEL	RFC2279 string	“Discovery Object”
CKA_APPLICATION	RFC2279 string	“PIV Card Application”
CKA_VALUE	Byte array	Discovery Raw data

- Vendor Attributes

Attribute	Data Type	Meaning
CKA_DISCOVERY_OBJECT_PIV_CARD_APPLICATION_AID	Byte array	AID of the PIV Card Application
CKA_DISCOVERY_OBJECT_PIN_USAGE_POLICY	Byte array	PIN Usage Policy list

The Discovery Object Vendor attributes are enumerated as follows:

```
Enum PivDiscoveryAttributes
{
CKA_DISCOVERY_OBJECT_PIV_CARD_APPLICATION_AID = (CKA_VENDOR_DEFINED+0x300),
CKA_DISCOVERY_OBJECT_PIN_USAGE_POLICY
};
```

9.5.2.7 CCC Object Attributes

- Data Object Attributes

Attribute	Data Type	Meaning / Value
CKA_LABEL	RFC2279 string	“CCC Object”
CKA_APPLICATION	RFC2279 string	“PIV Card Application”
CKA_VALUE	Byte array	CCC Raw data

- Vendor Attributes

Attribute	Data Type	Meaning
CKA _CCC_CARD_IDENTIFIER	Byte array	Card identifier
CKA_CCC_CAPABILITY_CONTAINER_VERSION	CK_BYTE	Capability container version
CKA_CCC_CAPABILITY_GRAMMAR_VERSION	CK_BYTE	Capability Grammar version
CKA_CCC_APPLICATION_CARDURL_LIST	Byte array	Application Card URL list (Encoding of the byte array consist in a succession of TLV, where T is the tag 0xF3, L is the length of the associated Value field, and V is the associated Value. The encoding of the Value itself is not part of this specification.)
CKA _CCC_PKCS15	CK_BYTE	PKCS 15 value
CKA _CCC_DATA_MODEL	CK_BYTE	Data Model
CKA _CCC_ACR_TABLE	Byte array	ACR table
CKA _CCC_CARD_APDUS	Byte array	Card APDUs
CKA _CCC_REDIRECTION_TAG	Byte array	Redirection tag
CKA _CCC_CAPABILITY_TUPLES	Byte array	Capability tuples
CKA _CCC_STATUS_TUPLES	Byte array	Status tuples
CKA _CCC_NEXT_CCC	Byte array	Next CCC
CKA_CCC_EXTENDED_APPLICATION_CARDURL_LIST	Byte array	Extended Application Card URL list (Encoding of the byte array consists in a succession of TLV fields, where T is the tag 0xE3, L is the length of the associated Value field, and V is the associated Value. The encoding of the Value itself is not part of this specification.)
CKA _CCC_SECURITY_OBJECT	Byte array	Security Object

The CCC Object Vendor attributes are enumerated as follows:

```
Enum PivCCCAttributes
{
CKA _CCC_CARD_IDENTIFIER = (CKA_VENDOR_DEFINED+0x390),
CKA_CCC_CAPABILITY_CONTAINER_VERSION,
CKA_CCC_CAPABILITY_GRAMMAR_VERSION,
CKA_CCC_APPLICATION_CARDURL_LIST,
CKA _CCC_PKCS15,
CKA _CCC_DATA_MODEL,
CKA _CCC_ACR_TABLE,
```

```
CKA _CCC_CARD_APDUS,
CKA _CCC_REDIRECTION_TAG,
CKA _CCC_CAPABILITY_TUPLES,
CKA _CCC_STATUS_TUPLES,
CKA _CCC_NEXT_CCC,
CKA_CCC_EXTENDED_APPLICATION_CARDURL_LIST,
CKA _CCC_SECURITY_OBJECT
};
```

9.5.2.8 Key History Object Attributes (SP800-73-3)

- Data Object Attributes

Attribute	Data Type	Meaning / Value
CKA_LABEL	RFC2279 string	“Key History Object”
CKA_APPLICATION	RFC2279 string	“PIV Card Application”
CKA_VALUE	Byte array	Key History Raw data

- Vendor Attributes

Attribute	Data Type	Meaning
CKA_KEY_HISTORY_OBJECT_KEYS_WITH_ON_CARD_CERTS	CK_ULONG	Number of retired private keys within the PIV Card Application for which the corresponding certificates are also stored within the PIV Card Application
CKA_KEY_HISTORY_OBJECT_KEYS_WITH_OFF_CARD_CERTS	CK_ULONG	Number of retired private keys within the PIV Card Application for which the corresponding certificates are not stored within the PIV Card Application
CKA_KEY_HISTORY_OBJECT_OFF_CARD_CERT_URL	RFC2279 string	URL that points to a file containing the certificates corresponding to all of the retired private keys within the PIV Card Application, including those for which the corresponding certificate is also stored within the PIV Card Application.

The Key History Object Vendor attributes are enumerated as follows:

```
Enum PivKeyHistoryAttributes
{
CKA_KEY_HISTORY_OBJECT_KEYS_WITH_ON_CARD_CERTS = (CKA_VENDOR_DEFINED+0x3C0),
CKA_KEY_HISTORY_OBJECT_KEYS_WITH_OFF_CARD_CERTS,
CKA_KEY_HISTORY_OBJECT_OFF_CARD_CERT_URL
```

```
};
```

9.5.2.9 Iris Images Object Attributes

Referenced for Future.

Use as the Iris format is not standardized in PIV yet.

- Data Object Attributes

Attribute	Data Type	Meaning / Value
CKA_LABEL	RFC2279 string	“Iris Image Object”
CKA_APPLICATION	RFC2279 string	“PIV Card Application”
CKA_VALUE	Byte array	Iris Image Raw data

- Vendor Attributes

Attribute	Data Type	Meaning
CKA_IRIS_IMAGE_CBEFF_HEADER	Byte array	CBEFF Header of the Iris image buffer
CKA_IRIS_IMAGE_CBEFF_RECORD	Byte array	Iris Image CBEFF record value.
CKA_IRIS_IMAGE_CBEFF_SIGNATURE	Byte array	Iris image signature
CKA_IRIS_IMAGE_SIGNATURE_STATUS	CK_ULONG	Iris image signature status See 9.4 Signature Status Flag Definition on page 40 for possible values.
CKA_IRIS_IMAGE_SIGNER_CERTIFICATE	Byte array	DER encoded value of Iris Image Signer certificate value
CKA_IRIS_IMAGE_FASCN	Byte array	FASC-N value extracted from Iris image signature(PIV only)
CKA_IRIS_IMAGE_UUID	Byte array	UUID value extracted from Iris image signature (PIV-I only)

The Iris Image Object Vendor attributes are enumerated as follows:

```
Enum PivIrisImageAttributes
{
CKA_IRIS_IMAGE_CBEFF_HEADER = (CKA_VENDOR_DEFINED+0x360),
```

```
CKA_IRIS_IMAGE_CBEFF_RECORD,
CKA_IRIS_IMAGE_CBEFF_SIGNATURE,
CKA_IRIS_IMAGE_SIGNATURE_STATUS,
CKA_IRIS_IMAGE_SIGNER_CERTIFICATE,
CKA_IRIS_IMAGE_FASCN,
CKA_IRIS_IMAGE_UUID
};
```

9.6 Data Object

9.6.1 Common Data Object Attributes

Attribute	Data Type	Meaning
CKA_APPLICATION	RFC2279 string	Description of the application that manages the object (default empty)
CKA_OBJECT_ID	Byte Array	DER-encoding of the object identifier indicating the data object type (default empty)
CKA_VALUE	Byte array	Value of the object (default empty)

9.6.2 Common Storage Object Attributes

Attribute	Data Type	Meaning
CKA_TOKEN	CK_BBOOL	CK_TRUE if object is a token object; CK_FALSE if object is a session object. Default is CK_FALSE.
CKA_PRIVATE	CK_BBOOL	CK_TRUE if object is a private object; CK_FALSE if object is a public object. Default value is token-specific, and may depend on the values of other attributes of the object.
CKA_MODIFIABLE	CK_BBOOL	CK_TRUE if object can be modified Default is CK_TRUE.
CKA_LABEL	RFC2279 string	Description of the object (default empty). This attribute may be modified after the object is created.
CKA_CLASS	CK_OBJECT_CLASS	Object class (type). Must be specified when object is created with C_CreateObject.

10.0 Token-Information Flags

Table 2: CK_TOKEN_INFO Flags Definition

Bit Flag	Mask	Supported values	Meaning
CKF_RNG	0x00000001	TRUE	True if the token has its own random number generator
CKF_WRITE_PROTECTED	0x00000002	TRUE	True if the token is write-protected (see below)
CKF_LOGIN_REQUIRED	0x00000004	TRUE	True if there are some cryptographic functions that a user must be logged in to perform
CKF_USER_PIN_INITIALIZED	0x00000008	TRUE/ FALSE	True if the normal user's PIN has been initialized
CKF_RESTORE_KEY_NOT_NEEDED	0x00000020	FALSE	True if a successful save of a session's cryptographic operations state always contains all keys needed to restore the state of the session
CKF_CLOCK_ON_TOKEN	0x00000040	FALSE	True if token has its own hardware clock
CKF_PROTECTED_AUTHENTICATION_PATH	0x00000100	FALSE	True if token has a "protected authentication path", whereby a user can log into the token without passing a PIN through the Cryptoki library
CKF_DUAL_CRYPTO_OPERATIONS	0x00000200	FALSE	True if a single session with the token can perform dual cryptographic operations
CKF_TOKEN_INITIALIZED	0x00000400	TRUE	True if the token has been initialized using C_InitializeToken or an equivalent mechanism outside the scope of this standard. Calling C_InitializeToken when this flag is set will cause the token to be reinitialized.
CKF_USER_PIN_COUNT_LOW	0x00010000	TRUE/FALSE	True if an incorrect user login PIN has been entered at least once since the last successful authentication.
CKF_USER_PIN_FINAL_TRY	0x00020000	TRUE/FALSE	True if supplying an incorrect user

Bit Flag	Mask	Supported values	Meaning
			PIN will it to become locked.
CKF_USER_PIN_LOCKED	0x00040000	TRUE/FALSE	True if the user PIN has been locked. User login to the token is not possible.
CKF_USER_PIN_TO_BE_CHANGED	0x00080000	TRUE/FALSE	True if the user PIN value is the default value set by token initialization or manufacturing, or the PIN has been expired by the card.
CKF_SO_PIN_COUNT_LOW	0x00100000	FALSE	True if an incorrect SO login PIN has been entered at least once since the last successful authentication.
CKF_SO_PIN_FINAL_TRY	0x00200000	FALSE	True if supplying an incorrect SO PIN will it to become locked.
CKF_SO_PIN_LOCKED	0x00400000	FALSE	True if the SO PIN has been locked. User login to the token is not possible.
CKF_SO_PIN_TO_BE_CHANGED	0x00800000	FALSE	True if the SO PIN value is the default value set by token initialization or manufacturing, or the PIN has been expired by the card.

11.0 Proprietary Flag

```
#define CKF_VENDOR_DEFINED          0x80000000
#define CKF_ACTI                    CKF_VENDOR_DEFINED | 0x01000000
#define CKF_ACTI_FIPS_MODE          CKF_ACTI | 0x00000001
```

Proprietary error code:

```
#define CKR_ACTI                    CKR_VENDOR_DEFINED | 0x01000000
#define CKR_ ACTI_FIPS_MODE_FAILED  CKR_ACTI | 0x00000001
```

12.0 Functions

12.1 General Purpose Functions

12.1.1 C_Initialize

C_Initialize initializes the Cryptoki library.

```
CK_DEFINE_FUNCTION(CK_RV, C_Initialize)(
  CK_VOID_PTR pInitArgs
);
```

Parameters:

- *pInitArgs* either has the value NULL_PTR or points to a CK_C_INITIALIZE_ARGS structure containing information on how the library should deal with multi-threaded access.

  If *pInitArgs* is non-NULL_PTR, C_Initialize should cast it to a CK_C_INITIALIZE_ARGS_PTR and then dereference the resulting pointer to obtain the CK_C_INITIALIZE_ARGS fields CreateMutex, DestroyMutex, LockMutex, UnlockMutex, flags, and pReserved. For this version of Cryptoki, the value of pReserved thereby obtained must be NULL_PTR; if it's not, then C_Initialize should return with the value CKR_ARGUMENTS_BAD.

Notes:

- If the CKF_LIBRARY_CANT_CREATE_OS_THREADS flag in the flags field is set, C_Initialize should return with the value CKR_NEED_TO_CREATE_THREADS.
- A call to C_Initialize specifies one of four different ways to support multi-threaded access via the value of the CKF_OS_LOCKING_OK flag in the flags field and the values of the CreateMutex, DestroyMutex, LockMutex, and UnlockMutex function pointer fields:
  - If the flag isn't set, and the function pointer fields aren't supplied (i.e., they all have the value NULL_PTR), that means that the application won't be accessing the Cryptoki library from multiple threads simultaneously.
  - If the flag is set, and the function pointer fields aren't supplied (i.e., they all have the value NULL_PTR), that means that the application will be performing multi-threaded Cryptoki access, and the library needs to use the native operating system primitives to ensure safe multi-threaded access. If the library is unable to do this, C_Initialize should return with the value CKR_CANT_LOCK.
  - If the flag isn't set, and the function pointer fields are supplied (i.e., they all have non-NULL_PTR values), that means that the application will be performing multi-threaded Cryptoki access, and the library needs to use the supplied function pointers for mutex-handling to ensure safe multi-threaded access. If the library is unable to do this, C_Initialize should return with the value CKR_CANT_LOCK.
  - If the flag is set, and the function pointer fields are supplied (i.e., they all have non-NULL_PTR values), that means that the application will be performing multi-threaded Cryptoki access, and the library needs to use either the native operating system primitives or the supplied function pointers for mutex-handling to

ensure safe multi-threaded access. If the library is unable to do this, C_Initialize should return with the value CKR_CANT_LOCK.

- If some, but not all, of the supplied function pointers to C_Initialize are non-NULL_PTR, then C_Initialize should return with the value CKR_ARGUMENTS_BAD.
- A call to C_Initialize with *pInitArgs* set to NULL_PTR is treated like a call to C_Initialize with *pInitArgs* pointing to a CK_C_INITIALIZE_ARGS which has the CreateMutex, DestroyMutex, LockMutex, UnlockMutex, and pReserved fields set to NULL_PTR, and has the flags field set to 0.
- C_Initialize should be the first Cryptoki call made by an application, except for calls to C_GetFunctionList. What this function actually does is implementation-dependent; typically, it might cause Cryptoki to initialize its internal memory buffers, or any other resources it requires.
- If several applications are using Cryptoki, each one should call C_Initialize. Every call to C_Initialize should (eventually) be succeeded by a single call to C_Finalize.
- A Vendor specific flag is used to define whether the crypto engine must be used in FIPS mode - CKF_ACTI_FIPS_MODE:
  - True – The underlying cryptographic engine will be set in FIPS mode
  - False – The underlying cryptographic engine will be set in non FIPS mode.

  To setup this mode all cryptographic operations must be stopped and all cryptographic structure must be freed.

  The C_Initialize function will return CKR_ ACTI_FIPS_MODE_FAILED if setting up the underlying crypto library in FIPS mode fails for whatever reason. Future calls to cryptographic functions on the token will return CKR_GENERAL_ERROR.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_ALREADY_INITIALIZED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_ ACTI_FIPS_MODE_FAILED.

12.1.2 C_Finalize

C_Finalize is called to indicate that an application is finished with the Cryptoki library. It should be the last Cryptoki call made by an application.

```
  CK_DEFINE_FUNCTION(CK_RV, C_Finalize)(
 CK_VOID_PTR pReserved
);
```

Parameters:

- The *pReserved* parameter must be NULL_PTR.

Notes:

- If several applications are using Cryptoki, each one should call C_Finalize. Each application's call to C_Finalize should be preceded by a single call to C_Initialize; in between the two calls, an application can make calls to other Cryptoki functions.
- Despite the fact that the parameters supplied to C_Initialize can in general allow for safe multi-threaded access to a Cryptoki library, the behavior of C_Finalize is nevertheless undefined if it is called by an application while other threads of the application are making Cryptoki calls.

  The exception to this exceptional behavior of C_Finalize occurs when a thread calls C_Finalize while another of the application's threads is blocking on Cryptoki's C_WaitForSlotEvent function. When this happens, the blocked thread becomes unblocked and returns the value CKR_CRYPTOKI_NOT_INITIALIZED.

  See C_WaitForSlotEvent for more information.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK.

12.1.3 C_Get_Info

C_GetInfo returns general information about Cryptoki.

```
CK_DEFINE_FUNCTION(CK_RV, C_GetInfo)(
  CK_INFO_PTR pInfo
);
```

Parameters:

pInfo points to the location that receives the information.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK.

12.1.4 C_GetFunctionList

C_GetFunctionList obtains a pointer to the Cryptoki library's list of function pointers.

```
CK_DEFINE_FUNCTION(CK_RV, C_GetFunctionList)(
  CK_FUNCTION_LIST_PTR_PTR ppFunctionList
);
```

Parameters:

- *ppFunctionList* points to a value which will receive a pointer to the library's CK_FUNCTION_LIST structure, which in turn contains function pointers for all the Cryptoki API routines in the library. The pointer thus obtained point into memory which is owned by the Cryptoki library, and which is not writable.

  Whether or not this is the case, no attempt should be made to write to this memory.

Notes:

- C_GetFunctionList is the only Cryptoki function which an application may call before calling C_Initialize. It is provided to make it easier and faster for applications to use shared Cryptoki libraries and to use more than one Cryptoki library simultaneously.

Return values:

CKR_ARGUMENTS_BAD, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK.

12.2 Slot and Token Management Functions

12.2.1 C_GetSlotList

C_GetSlotList is used to obtain a list of slots in the system.

```
CK_DEFINE_FUNCTION(CK_RV, C_GetSlotList)(
  CK_BBOOL tokenPresent,
  CK_SLOT_ID_PTR pSlotList,
  CK_ULONG_PTR pulCount
);
```

Parameters:

- *tokenPresent* indicates whether the list obtained includes only those slots with a token present (CK_TRUE), or all slots (CK_FALSE);
- *pulCount* points to the location that receives the number of slots.

Notes:

- There are two ways for an application to call C_GetSlotList:
  - If pSlotList is NULL_PTR, then all that C_GetSlotList does is return (in *pulCount) the number of slots, without actually returning a list of slots. The contents of the buffer pointed to by pulCount on entry to C_GetSlotList has no meaning in this case, and the call returns the value CKR_OK.
  - If pSlotList is not NULL_PTR, then *pulCount must contain the size (in terms of CK_SLOT_ID elements) of the buffer pointed to by pSlotList. If that buffer is large enough to hold the list of slots, then the list is returned in it, and CKR_OK is returned. If not, then the call to C_GetSlotList returns the value CKR_BUFFER_TOO_SMALL. In either case, the value *pulCount is set to hold the number of slots.

- C_GetSlotList must be called twice at least: once to retrieve the slot count and another to get the slot list.

Return values:

CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK.

12.2.2 C_GetSlotInfo

C_GetSlotInfo obtains information about a particular slot in the system.

```
CK_DEFINE_FUNCTION(CK_RV, C_GetSlotInfo)(
  CK_SLOT_ID slotID,
  CK_SLOT_INFO_PTR pInfo
);
```

Parameters:

- *slotID* is the ID of the slot.
- *pInfo* points to the location that receives the slot information.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_SLOT_ID_INVALID.

12.2.3 C_GetTokenInfo

C_GetTokenInfo obtains information about a particular token in the system.

```
CK_DEFINE_FUNCTION(CK_RV, C_GetTokenInfo)(
  CK_SLOT_ID slotID,
  CK_TOKEN_INFO_PTR pInfo
);
```

Parameters:

- *slotID* is the ID of the token's slot.
- *pInfo* points to the location that receives the token information.

Return values:

CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY,

CKR_OK, CKR_SLOT_ID_INVALID, CKR_TOKEN_NOT_PRESENT, CKR_TOKEN_NOT_RECOGNIZED, CKR_ARGUMENTS_BAD.

12.2.4 C_WaitForSlotEvent

C_WaitForSlotEvent waits for a slot event, such as token insertion or token removal, to occur.

```
CK_DEFINE_FUNCTION(CK_RV, C_WaitForSlotEvent)(
  CK_FLAGS flags,
  CK_SLOT_ID_PTR pSlot,
  CK_VOID_PTR pReserved
);
```

Parameters:

- *flags* determines whether or not the C_WaitForSlotEvent call blocks (i.e., waits for a slot event to occur).
- *pSlot* points to a location which will receive the ID of the slot that the in which the event occurred.
- The *pReserved* parameter must be NULL_PTR.

Notes:

- At present, the only flag defined for use in the flags argument is CKF_DONT_BLOCK:
- Internally, each Cryptoki application has a flag for each slot which is used to track whether or not any unrecognized events involving that slot have occurred. When an application initially calls C_Initialize, every slot's event flag is cleared. Whenever a slot event occurs, the flag corresponding to the slot in which the event occurred is set.
- If C_WaitForSlotEvent is called with the CKF_DONT_BLOCK flag set in the flags argument, and some slot's event flag is set, then that event flag is cleared, and the call returns with the ID of that slot in the location pointed to by pSlot. If more than one slot's event flag is set at the time of the call, one such slot is chosen by the library to have its event flag cleared and to have its slot ID returned.
- If C_WaitForSlotEvent is called with the CKF_DONT_BLOCK flag set in the flags argument, and no slot's event flag is set, then the call returns with the value CKR_NO_EVENT. In this case, the contents of the location pointed to by pSlot when C_WaitForSlotEvent are undefined.
- If C_WaitForSlotEvent is called with the CKF_DONT_BLOCK flag clear in the flags argument, then the call behaves as above, except that it will block. That is, if no slot's event flag is set at the time of the call, C_WaitForSlotEvent will wait until some slot's event flag becomes set. If a thread of an application has a C_WaitForSlotEvent call blocking when another thread of that application calls C_Finalize, the C_WaitForSlotEvent call returns with the value CKR_CRYPTOKI_NOT_INITIALIZED.
- Although the parameters supplied to C_Initialize can in general allow for safe multi-threaded access to a Cryptoki library, C_WaitForSlotEvent is exceptional in that the behavior of Cryptoki is undefined if multiple threads of a single application make simultaneous calls to C_WaitForSlotEvent.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_NO_EVENT, CKR_OK.

12.2.5 C_GetMechanismList

C_GetMechanismList is used to obtain a list of mechanism types supported by a token.

```
CK_DEFINE_FUNCTION(CK_RV, C_GetMechanismList)(
  CK_SLOT_ID slotID,
  CK_MECHANISM_TYPE_PTR pMechanismList,
  CK_ULONG_PTR pulCount
);
```

Parameters

- *slotID* is the ID of the token's slot.
- *pulCount* points to the location that receives the number of mechanisms.

Notes:

- There are two ways for an application to call C_GetMechanismList:
  - If pMechanismList is NULL_PTR, then all that C_GetMechanismList does is return (in *pulCount) the number of mechanisms, without actually returning a list of mechanisms. The contents of *pulCount on entry to C_GetMechanismList has no meaning in this case, and the call returns the value CKR_OK.
  - If pMechanismList is not NULL_PTR, then *pulCount must contain the size (in terms of CK_MECHANISM_TYPE elements) of the buffer pointed to by pMechanismList. If that buffer is large enough to hold the list of mechanisms, then the list is returned in it, and CKR_OK is returned. If not, then the call to C_GetMechanismList returns the value CKR_BUFFER_TOO_SMALL. In either case, the value *pulCount is set to hold the number of mechanisms.
- C_GetMechanismList must be called twice at least: Once to retrieve the Mechanism count and another to get the mechanism list.

Return values:

CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_SLOT_ID_INVALID, CKR_TOKEN_NOT_PRESENT, CKR_TOKEN_NOT_RECOGNIZED, CKR_ARGUMENTS_BAD.

12.2.6 C_GetMechanismInfo

C_GetMechanismInfo obtains information about a particular mechanism possibly supported by a token.

```
CK_DEFINE_FUNCTION(CK_RV, C_GetMechanismInfo)(
  CK_SLOT_ID slotID,
  CK_MECHANISM_TYPE type,
  CK_MECHANISM_INFO_PTR pInfo
);
```

Parameters:

- *slotID* is the ID of the token's slot.
- *type* is the type of mechanism;
- *pInfo* points to the location that receives the mechanism information.

Return values:

CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_MECHANISM_INVALID, CKR_OK, CKR_SLOT_ID_INVALID, CKR_TOKEN_NOT_PRESENT, CKR_TOKEN_NOT_RECOGNIZED, CKR_ARGUMENTS_BAD.

12.2.7 C_InitToken

C_InitToken initializes a token.

```
CK_DEFINE_FUNCTION(CK_RV, C_InitToken)(
  CK_SLOT_ID slotID,
  CK_UTF8CHAR_PTR pPin,
  CK_ULONG ulPinLen,
  CK_UTF8CHAR_PTR pLabel
);
```

Parameters:

- *slotID* is the ID of the token's slot.
- *pPin* points to the SO's initial PIN (which need not be null-terminated).
- *ulPinLen* is the length in bytes of the PIN.
- pLabel points to the 32-byte label of the token (which must be padded with blank characters, and which must not be null-terminated). This standard allows PIN values to contain any valid UTF8 character, but the token may impose subset restrictions.

Notes:

- If the token has not been initialized (i.e. new from the factory), then the pPin parameter becomes the initial value of the SO PIN. If the token is being reinitialized, the pPin parameter is checked against the existing SO PIN to authorize the initialization operation. In both cases, the SO PIN is the value pPin after the function completes successfully. If the SO PIN is lost, then the card must be reinitialized using a mechanism outside the scope of this standard. The CKF_TOKEN_INITIALIZED flag in the CK_TOKEN_INFO structure indicates the action that will result from calling C_InitToken. If set, the token will be reinitialized, and the client must supply the existing SO password in pPin and the existing user PIN may be asked.
- When a token is initialized, all objects that can be destroyed are destroyed (i.e., all except for “indestructible” objects such as keys built into the token). Also, access by the normal user is disabled until the SO sets the normal user’s PIN. Depending on the token, some “default” objects may be created, and attributes of some objects may be set to default values.
- A token cannot be initialized if Cryptoki detects that any application has an open session with it; when a call to C_InitToken is made under such circumstances, the call fails with error CKR_SESSION_EXISTS. Unfortunately, it may happen when C_InitToken is called that some other application does have an open session with the token, but Cryptoki cannot detect this, because it cannot detect anything about other applications using the token. If this is the case, then the consequences of the C_InitToken call are undefined.
- The C_InitToken function may not be sufficient to properly initialize complex tokens. In these situations, an initialization mechanism outside the scope of Cryptoki must be employed. The definition of “complex token” is product specific.
- If the token has a “protected authentication path”, as indicated by the CKF_PROTECTED_AUTHENTICATION_PATH flag in its CK_TOKEN_INFO being set, then that means that there is some way for a user to be authenticated to the token without having the application send a PIN through the Cryptoki library. One such possibility is that the user enters a PIN on a PINpad on the token itself, or a PIN prompt is displayed for user to type their PIN. To initialize a token with such a protected authentication path, the pPin parameter to C_InitToken should be NULL_PTR. During the execution of C_InitToken, the SO’s PIN will be entered through the protected authentication path.

Return values:

CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_PIN_INCORRECT, CKR_PIN_LOCKED, CKR_SESSION_EXISTS, CKR_SLOT_ID_INVALID, CKR_TOKEN_NOT_PRESENT, CKR_TOKEN_NOT_RECOGNIZED, CKR_TOKEN_WRITE_PROTECTED, CKR_ARGUMENTS_BAD.

12.2.8 C_InitPIN

C_InitPIN initializes the normal user’s PIN.

```
CK_DEFINE_FUNCTION(CK_RV, C_InitPIN)(
  CK_SESSION_HANDLE hSession,
```

```
  CK_UTF8CHAR_PTR pPin,
  CK_ULONG ulPinLen
);
```

Parameters:

- *hSession* is the session's handle.
- *pPin* points to the normal user's PIN;
- *ulPinLen* is the length in bytes of the PIN. This standard allows PIN values to contain any valid UTF8 character, but the token may impose subset restrictions.

Notes:

- C_InitPIN can only be called in the "R/W SO Functions" state. An attempt to call it from a session in any other state fails with error CKR_USER_NOT_LOGGED_IN.
- If the token has a "protected authentication path", as indicated by the CKF_PROTECTED_AUTHENTICATION_PATH flag in its CK_TOKEN_INFO being set, then that means that there is some way for a user to be authenticated to the token without having the application send a PIN through the Cryptoki library. One such possibility is that the user enters a PIN on a PINpad on the token itself, or a PIN prompt is displayed for user to type their PIN. To initialize the normal user's PIN on a token with such a protected authentication path, the pPin parameter to C_InitPIN should be NULL_PTR. During the execution of C_InitPIN, the SO will enter the new PIN through the protected authentication path.

Return values:

CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_PIN_INVALID, CKR_PIN_LEN_RANGE, CKR_SESSION_CLOSED, CKR_SESSION_READ_ONLY, CKR_SESSION_HANDLE_INVALID, CKR_TOKEN_NOT_PRESENT, CKR_TOKEN_WRITE_PROTECTED, CKR_USER_NOT_LOGGED_IN, CKR_ARGUMENTS_BAD.

12.2.9 C_SetPIN

C_SetPIN modifies the PIN of the user that is currently logged in, or the CKU_USER PIN if the session is not logged in.

```
CK_DEFINE_FUNCTION(CK_RV, C_SetPIN)(
  CK_SESSION_HANDLE hSession,
  CK_UTF8CHAR_PTR pOldPin,
  CK_ULONG ulOldLen,
  CK_UTF8CHAR_PTR pNewPin,
  CK_ULONG ulNewLen
);
```

Parameters:

- *hSession* is the session's handle;
- *pOldPin* points to the old PIN;
- *ulOldLen* is the length in bytes of the old PIN;
- *pNewPin* points to the new PIN;
- *ulNewLen* is the length in bytes of the new PIN. This standard allows PIN values to contain any valid UTF8 character, but the token may impose subset restrictions.

Notes:

- C_SetPIN can only be called in the "R/W Public Session" state, "R/W SO Functions" state, or "R/W User Functions" state. An attempt to call it from a session in any other state fails with error CKR_SESSION_READ_ONLY.
- If the token has a "protected authentication path", as indicated by the CKF_PROTECTED_AUTHENTICATION_PATH flag in its CK_TOKEN_INFO being set, then that means that there is some way for a user to be authenticated to the token without having the application send a PIN through the Cryptoki library. One such possibility is that the user enters a PIN on a PINpad on the token itself, or a PIN prompt is displayed for user to type their PIN. To modify the current user's PIN on a token with such a protected authentication path, the pOldPin and pNewPin parameters to C_SetPIN should be NULL_PTR. During the execution of C_SetPIN, the current user will enter the old PIN and the new PIN through the protected authentication path.

Return values:

CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_PIN_INCORRECT, CKR_PIN_INVALID, CKR_PIN_LEN_RANGE, CKR_PIN_LOCKED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_TOKEN_NOT_PRESENT, CKR_TOKEN_WRITE_PROTECTED, CKR_ARGUMENTS_BAD.

12.3 Session Management Functions

12.3.1 C_OpenSession

C_OpenSession opens a session between an application and a token in a particular slot.

```
CK_DEFINE_FUNCTION(CK_RV, C_OpenSession)(
  CK_SLOT_ID slotID,
  CK_FLAGS flags,
  CK_VOID_PTR pApplication,
  CK_NOTIFY Notify,
```

```
  CK_SESSION_HANDLE_PTR phSession
);
```

Parameters:

- *slotID* is the slot's ID;
- *flags* indicates the type of session;
- *pApplication* should be NULL_PTR;
- *Notify* should be NULL_PTR;
- *phSession* points to the location that receives the handle for the new session.

Notes:

- When opening a session with C_OpenSession, the flags parameter consists of the logical OR of zero or more bit flags defined in the CK_SESSION_INFO data type.
- For legacy reasons, the CKF_SERIAL_SESSION bit must always be set.
- If a call to C_OpenSession does not have this bit set, the call returns unsuccessfully with the error code CKR_PARALLEL_NOT_SUPPORTED.
- There may be a limit on the number of concurrent sessions an application may have with the token, which may depend on whether the session is "read-only" or "read/write". An attempt to open a session which does not succeed because there are too many existing sessions of some type should return CKR_SESSION_COUNT.
- All supported token are write-protected (as indicated in the CK_TOKEN_INFO structure), only read-only sessions can be opened with those tokens.
- If the application calling C_OpenSession already has a R/W SO session open with the token, then any attempt to open a R/O session with the token fails with error code CKR_SESSION_READ_WRITE_SO_EXISTS.

Return values:

CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_SESSION_COUNT, CKR_SESSION_PARALLEL_NOT_SUPPORTED, CKR_SLOT_ID_INVALID, CKR_TOKEN_NOT_PRESENT, CKR_TOKEN_NOT_RECOGNIZED, CKR_TOKEN_WRITE_PROTECTED, CKR_ARGUMENTS_BAD.

12.3.2 C_CloseSession

C_CloseSession closes a session between an application and a token.

```
CK_DEFINE_FUNCTION(CK_RV, C_CloseSession)(
  CK_SESSION_HANDLE hSession
);
```

Parameters:

- *hSession* is the session's handle.

Notes:

- When a session is closed, all session objects created by the session are destroyed automatically, even if the application has other sessions "using" the objects.
- If this function is successful and it closes the last session between the application and the token, the login state of the token for the application returns to public sessions. Any new sessions to the token opened by the application will be either R/O Public or R/W Public sessions.

Return values:

CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_TOKEN_NOT_PRESENT.

12.3.3 C_CloseAllSessions

C_CloseAllSessions closes all sessions an application has with a token.

```
CK_DEFINE_FUNCTION(CK_RV, C_CloseAllSessions)(
  CK_SLOT_ID slotID
);
```

Parameters:

- *slotID* specifies the token's slot.

Notes:

- When a session is closed, all session objects created by the session are destroyed automatically.
- After successful execution of this function, the login state of the token for the application returns to public sessions. Any new sessions to the token opened by the application will be either R/O Public or R/W Public sessions.

Return values:

CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_SLOT_ID_INVALID, CKR_TOKEN_NOT_PRESENT.

12.3.4 C_GetSessionInfo

C_GetSessionInfo obtains information about a session.

```
CK_DEFINE_FUNCTION(CK_RV, C_GetSessionInfo)(
  CK_SESSION_HANDLE hSession,
  CK_SESSION_INFO_PTR pInfo
);
```

Parameters:

- *hSession* is the session's handle.
- *pInfo* points to the location that receives the session information.

Return values:

CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_ARGUMENTS_BAD.

12.3.5 C_GetOperationState

Function not supported.

12.3.6 C_SetOperationState

Function not supported.

12.3.7 C_Login

C_Login logs a user into a token.

```
CK_DEFINE_FUNCTION(CK_RV, C_Login)(
  CK_SESSION_HANDLE hSession,
  CK_USER_TYPE userType,
  CK_UTF8CHAR_PTR pPin,
  CK_ULONG ulPinLen
);
```

Parameters:

- *hSession* is the session handle;
- *userType* is the user type; Only CKU_USER type is supported.
- *pPin* points to the user's PIN; NULL_PTR is not supported

- *ulPinLen* is the length of the PIN. PIN values contain any valid UTF8 character

Notes:

- If the call succeeds, each of the application's sessions will enter either the "R/W SO Functions" state, the "R/W User Functions" state, or the "R/O User Functions" state.
- If the token has a “protected authentication path”, as indicated by the CKF_PROTECTED_AUTHENTICATION_PATH flag in its CK_TOKEN_INFO being set, then that means that there is some way for a user to be authenticated to the token without having the application send a PIN through the Cryptoki library. One such possibility is that the user enters a PIN on a PINpad on the token itself, or a PIN prompt is displayed for user to type their PIN. To log into a token with a protected authentication path, the pPin parameter to `C_Login` should be NULL_PTR. When `C_Login` returns, whatever authentication method supported by the token will have been performed; a return value of CKR_OK means that the user was successfully authenticated, and a return value of CKR_PIN_INCORRECT means that the user was denied access.
- If there are any active cryptographic or object finding operations in an application’s session, and then `C_Login` is successfully executed by that application, it may or may not be the case that those operations are still active. Therefore, before logging in, any active operations should be finished.
- If the application calling `C_Login` has a R/O session open with the token, then it will be unable to log the SO into a session. An attempt to do this will result in the error code CKR_SESSION_READ_ONLY_EXISTS.
- `C_Login` may be called repeatedly, without intervening `C_Logout` calls, if (and only if) a key with the CKA_ALWAYS_AUTHENTICATE attribute set to CK_TRUE exists, and the user needs to do cryptographic operation on this key.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_PIN_INCORRECT, CKR_PIN_LOCKED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY_EXISTS, CKR_USER_ALREADY_LOGGED_IN, CKR_USER_ANOTHER_ALREADY_LOGGED_IN, CKR_USER_PIN_NOT_INITIALIZED, CKR_USER_TOO_MANY_TYPES, CKR_USER_TYPE_INVALID, CKR_TOKEN_NOT_PRESENT.

12.3.8 C_Logout

`C_Logout` logs a user out from a token.

```
CK_DEFINE_FUNCTION(CK_RV, C_Logout)(
  CK_SESSION_HANDLE hSession
);
```

Parameters:

- *hSession* is the session's handle.

Notes:

When C_Logout successfully executes, any of the application's handles to private objects become invalid. In addition, all private session objects from sessions belonging to the application are destroyed.

Return values:

CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN, CKR_TOKEN_NOT_PRESENT.

12.4 Objet Management Functions

12.4.1 C_CreateObject

C_CreateObject creates a new object.

```
CK_DEFINE_FUNCTION(CK_RV, C_CreateObject)(
  CK_SESSION_HANDLE hSession,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulCount,
  CK_OBJECT_HANDLE_PTR phObject
);
```

Parameters:

- *hSession* is the session's handle;
- *pTemplate* points to the object's template;
- *ulCount* is the number of attributes in the template;
- *phObject* points to the location that receives the new object's handle.

Notes:

- If C_CreateObject is used to create a key object, the key object will have its CKA_LOCAL attribute set to CK_FALSE. If that key object is a secret key then the new key will have the CKA_ALWAYS_SENSITIVE attribute set to CK_FALSE, and the CKA_NEVER_EXTRACTABLE attribute set to CK_FALSE. If that key object is a private key then the new key will have the CKA_ALWAYS_SENSITIVE attribute set to CK_TRUE, and the CKA_NEVER_EXTRACTABLE attribute set to CK_TRUE.
- Only session objects can be created during a read-only session. Only public objects can be created unless the normal user is logged in.

Return values:

CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_READ_ONLY, CKR_ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_DOMAIN_PARAMS_INVALID, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_TEMPLATE_INCOMPLETE, CKR_TEMPLATE_INCONSISTENT, CKR_TOKEN_WRITE_PROTECTED, CKR_USER_NOT_LOGGED_IN.

12.4.2 C_CopyObject

`C_CopyObject` copies an object, creating a new object for the copy.

```
CK_DEFINE_FUNCTION(CK_RV, C_CopyObject)(
  CK_SESSION_HANDLE hSession,
  CK_OBJECT_HANDLE hObject,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulCount,
  CK_OBJECT_HANDLE_PTR phNewObject
);
```

Parameters:

- *hSession* is the session's handle;
- *hObject* is the object's handle;
- *pTemplate* points to the template for the new object;
- *ulCount* is the number of attributes in the template;
- *phNewObject* points to the location that receives the handle for the copy of the object.

Notes:

- The template may specify new values for any attributes of the object that can ordinarily be modified. It may also specify new values of the CKA_TOKEN and CKA_PRIVATE attributes (e.g., to copy a session object to a token object). If the template specifies a value of an attribute which is incompatible with other existing attributes of the object, the call fails with the return code CKR_TEMPLATE_INCONSISTENT.
- If a call to `C_CopyObject` cannot support the precise template supplied to it, it will fail and return without creating any object.
- Only session objects can be created during a read-only session. Only public objects can be created unless the normal user is logged in.

Return values:

CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_READ_ONLY, CKR_ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OBJECT_HANDLE_INVALID, CKR_OK, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_TEMPLATE_INCONSISTENT, CKR_TOKEN_WRITE_PROTECTED, CKR_USER_NOT_LOGGED_IN.

12.4.3 C_DestroyObject

`C_DestroyObject` destroys an object.

```
CK_DEFINE_FUNCTION(CK_RV, C_DestroyObject)(
  CK_SESSION_HANDLE hSession,
  CK_OBJECT_HANDLE hObject
);
```

Parameters:

- *hSession* is the session's handle;
- *hObject* is the object's handle.

Notes:

Only session objects can be destroyed during a read-only session. Only public objects can be destroyed unless the normal user is logged in.

Return values:

CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OBJECT_HANDLE_INVALID, CKR_OK, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_TOKEN_WRITE_PROTECTED.

12.4.4 C_GetObjectSize

`C_GetObjectSize` gets the size of an object in bytes.

```
CK_DEFINE_FUNCTION(CK_RV, C_GetObjectSize)(
  CK_SESSION_HANDLE hSession,
  CK_OBJECT_HANDLE hObject,
  CK_ULONG_PTR pulSize
);
```

Parameters:

- *hSession* is the session's handle;

- *hObject* is the object's handle;
- *pulSize* points to the location that receives the size in bytes of the object.

Notes:

This function is only supported for CKO_CERTIFICATE and CKO_PRIVATE_KEY objects.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_INFORMATION_SENSITIVE, CKR_OBJECT_HANDLE_INVALID, CKR_OK, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

12.4.5 C_GetAttributeValue

C_GetAttributeValue obtains the value of one or more attributes of an object.

```
CK_DEFINE_FUNCTION(CK_RV, C_GetAttributeValue)(
  CK_SESSION_HANDLE hSession,
  CK_OBJECT_HANDLE hObject,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulCount
);
```

Parameters:

- *hSession* is the session's handle;
- *hObject* is the object's handle;
- *pTemplate* points to a template that specifies which attribute values are to be obtained, and receives the attribute values;
- *ulCount* is the number of attributes in the template.

Notes:

- For each (type, pValue, ulValueLen) triple in the template, C_GetAttributeValue performs the following algorithm:
  - If the specified attribute (i.e., the attribute specified by the type field) for the object cannot be revealed because the object is sensitive or unextractable, then the ulValueLen field in that triple is modified to hold the value -1 (i.e., when it is cast to a CK_LONG, it holds -1).
  - Otherwise, if the specified attribute for the object is invalid (the object does not possess such an attribute), then the ulValueLen field in that triple is modified to hold the value -1.
  - Otherwise, if the pValue field has the value NULL_PTR, then the ulValueLen field is modified to hold the exact length of the specified attribute for the object.

- Otherwise, if the length specified in ulValueLen is large enough to hold the value of the specified attribute for the object, then that attribute is copied into the buffer located at pValue, and the ulValueLen field is modified to hold the exact length of the attribute.
- Otherwise, the ulValueLen field is modified to hold the value -1.

- If case 1 applies to any of the requested attributes, then the call return the value CKR_ATTRIBUTE_SENSITIVE.
- If case 2 applies to any of the requested attributes, then the call return the value CKR_ATTRIBUTE_TYPE_INVALID.
- If case 5 applies to any of the requested attributes, then the call return the value CKR_BUFFER_TOO_SMALL.

Return values:

CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_SENSITIVE, CKR_ATTRIBUTE_TYPE_INVALID, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OBJECT_HANDLE_INVALID, CKR_OK, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

12.4.6 C_SetAttributeValue

C_SetAttributeValue modifies the value of one or more attributes of an object.

```
CK_DEFINE_FUNCTION(CK_RV, C_SetAttributeValue)(
  CK_SESSION_HANDLE hSession,
  CK_OBJECT_HANDLE hObject,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulCount
);
```

Parameters:

- *hSession* is the session's handle;
- *hObject* is the object's handle;
- *pTemplate* points to a template that specifies which attribute values are to be modified and their new values;
- *ulCount* is the number of attributes in the template.

Notes:

Only session objects can be modified during a read-only session.

The template may specify new values for any attributes of the object that can be modified. If the template specifies a value of an attribute which is incompatible with other existing attributes of the object, the call fails with the return code CKR_TEMPLATE_INCONSISTENT.

Return values:

CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_READ_ONLY, CKR_ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OBJECT_HANDLE_INVALID, CKR_OK, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_TEMPLATE_INCONSISTENT, CKR_TOKEN_WRITE_PROTECTED, CKR_USER_NOT_LOGGED_IN.

12.4.7 C_FindObjectsInit

`C_FindObjectsInit` initializes a search for token and session objects that match a template.

```
CK_DEFINE_FUNCTION(CK_RV, C_FindObjectsInit)(
  CK_SESSION_HANDLE hSession,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulCount
);
```

Parameters:

- *hSession* is the session's handle;
- *pTemplate* points to a search template that specifies the attribute values to match;
- *ulCount* is the number of attributes in the search template.

Notes:

- To find all objects, set ulCount to 0.
- After calling `C_FindObjectsInit`, the application may call `C_FindObjects` one or more times to obtain handles for objects matching the template, and then eventually call `C_FindObjectsFinal` to finish the active search operation.
- The object search operation will only find objects that the session can view.

Return values:

CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

12.4.8 C_FindObjects

C_FindObjects continues a search for token and session objects that match a template, obtaining additional object handles.

```
CK_DEFINE_FUNCTION(CK_RV, C_FindObjects)(
  CK_SESSION_HANDLE hSession,
  CK_OBJECT_HANDLE_PTR phObject,
  CK_ULONG ulMaxObjectCount,
  CK_ULONG_PTR pulObjectCount
);
```

Parameters:

- *hSession* is the session's handle;
- *phObject* points to the location that receives the list (array) of additional object handles; *ulMaxObjectCount* is the maximum number of object handles to be returned;
- *pulObjectCount* points to the location that receives the actual number of object handles returned.

Notes:

- If there are no more objects matching the template, then the location that *pulObjectCount* points to receive the value 0.
- The search must have been initialized with C_FindObjectsInit.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

12.4.9 C_FindObjectsFinal

C_FindObjectsFinal terminates a search for token and session objects.

```
CK_DEFINE_FUNCTION(CK_RV, C_FindObjectsFinal)(
  CK_SESSION_HANDLE hSession
);
```

Parameters:

hSession is the session's handle.

Return values:

CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

12.5 Encryption Functions

12.5.1 C_EncryptInit

C_EncryptInit initializes an encryption operation.

```
CK_DEFINE_FUNCTION(CK_RV, C_EncryptInit)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hKey
);
```

Parameters:

- *hSession* is the session's handle;
- *pMechanism* points to the encryption mechanism;
- *hKey* is the handle of the encryption key.

Notes:

- The CKA_ENCRYPT attribute of the encryption key, which indicates whether the key supports encryption, must be CK_TRUE.
- After calling C_EncryptInit, the application can either call C_Encrypt to encrypt data in a single part; or call C_EncryptUpdate zero or more times, followed by C_EncryptFinal, to encrypt data in multiple parts.
- The encryption operation is active until the application uses a call to C_Encrypt or C_EncryptFinal to actually obtain the final piece of ciphertext. To process additional data (in single or multiple parts), the application must call C_EncryptInit again.

Return values:

CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_KEY_FUNCTION_NOT_PERMITTED, CKR_KEY_HANDLE_INVALID, CKR_KEY_SIZE_RANGE, CKR_KEY_TYPE_INCONSISTENT, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN.

Note: Support of this function is no longer conditional.

12.5.2 C_Encrypt

C_Encrypt encrypts single-part data.

```
CK_DEFINE_FUNCTION(CK_RV, C_Encrypt)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pData,
  CK_ULONG ulDataLen,
  CK_BYTE_PTR pEncryptedData,
  CK_ULONG_PTR pulEncryptedDataLen
);
```

Parameters:

- *hSession* is the session's handle;
- *pData* points to the data;
- *ulDataLen* is the length in bytes of the data;
- *pEncryptedData* points to the location that receives the encrypted data;
- *pulEncryptedDataLen* points to the location that holds the length in bytes of the encrypted data.

Notes:

- The encryption operation must have been initialized with C_EncryptInit. A call to C_Encrypt always terminates the active encryption operation unless it returns CKR_BUFFER_TOO_SMALL or is a successful call (i.e., one which returns CKR_OK) to determine the length of the buffer needed to hold the ciphertext.
- C_Encrypt cannot be used to terminate a multi-part operation, and must be called after C_EncryptInit without intervening C_EncryptUpdate calls.
- For some encryption mechanisms, the input plaintext data has certain length constraints (either because the mechanism can only encrypt relatively short pieces of plaintext, or because the mechanism's input data must consist of an integral number of blocks). If these constraints are not satisfied, then C_Encrypt will fail with return code CKR_DATA_LEN_RANGE.
- The plaintext and ciphertext can be in the same place, i.e., it is OK if pData and pEncryptedData point to the same location.
- For most mechanisms, C_Encrypt is equivalent to a sequence of C_EncryptUpdate operations followed by C_Encrypt Final.

Return values:

CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DATA_INVALID, CKR_DATA_LEN_RANGE, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

Note: Support of this function is no longer conditional.

12.5.3 C_EncryptUpdate

Function not supported.

12.5.4 C_EncryptFinal

Function not supported.

12.6 Decryption Functions

12.6.1 C_DecryptInit

C_DecryptInit initializes a decryption operation.

```
CK_DEFINE_FUNCTION(CK_RV, C_DecryptInit)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hKey
);
```

Parameters:

- *hSession* is the session's handle;
- *pMechanism* points to the decryption mechanism;
- *hKey* is the handle of the decryption key.

Notes:

- The CKA_DECRYPT attribute of the decryption key, which indicates whether the key supports decryption, must be CK_TRUE.
- After calling C_DecryptInit, the application can either call C_Decrypt to decrypt data in a single part; or call C_DecryptUpdate zero or more times, followed by C_DecryptFinal, to decrypt data in multiple parts. The decryption operation is active until the application uses a call to C_Decrypt or C_DecryptFinal to actually

obtain the final piece of plaintext. To process additional data (in single or multiple parts), the application must call C_DecryptInit again

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_KEY_FUNCTION_NOT_PERMITTED, CKR_KEY_HANDLE_INVALID, CKR_KEY_SIZE_RANGE, CKR_KEY_TYPE_INCONSISTENT, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN.

12.6.2 C_Decrypt

C_Decrypt decrypts encrypted data in a single part.

```
CK_DEFINE_FUNCTION(CK_RV, C_Decrypt)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pEncryptedData,
  CK_ULONG ulEncryptedDataLen,
  CK_BYTE_PTR pData,
  CK_ULONG_PTR pulDataLen
);
```

Parameters:

- *hSession* is the session's handle;
- *pEncryptedData* points to the encrypted data;
- *ulEncryptedDataLen* is the length of the encrypted data;
- *pData* points to the location that receives the recovered data;
- *pulDataLen* points to the location that holds the length of the recovered data.

Notes:

- The decryption operation must have been initialized with C_DecryptInit. A call to C_Decrypt always terminates the active decryption operation unless it returns CKR_BUFFER_TOO_SMALL or is a successful call (i.e., one which returns CKR_OK) to determine the length of the buffer needed to hold the plaintext.
- C_Decrypt cannot be used to terminate a multi-part operation, and must be called after C_DecryptInit without intervening C_DecryptUpdate calls.
- The ciphertext and plaintext can be in the same place, i.e., it is OK if pEncryptedData and pData point to the same location.

- If the input ciphertext data cannot be decrypted because it has an inappropriate length, then either CKR_ENCRYPTED_DATA_INVALID or CKR_ENCRYPTED_DATA_LEN_RANGE may be returned.
- For most mechanisms, C_Decrypt is equivalent to a sequence of C_DecryptUpdate operations followed by C_DecryptFinal.

Return values:

CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_ENCRYPTED_DATA_INVALID, CKR_ENCRYPTED_DATA_LEN_RANGE, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN.

12.6.3 C_DecryptUpdate

C_DecryptUpdate continues a multiple-part decryption operation, processing another encrypted data part.

```
CK_DEFINE_FUNCTION(CK_RV, C_DecryptUpdate)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pEncryptedPart,
  CK_ULONG ulEncryptedPartLen,
  CK_BYTE_PTR pPart,
  CK_ULONG_PTR pulPartLen
);
```

Parameters:

- *hSession* is the session's handle;
- *pEncryptedPart* points to the encrypted data part;
- *ulEncryptedPartLen* is the length of the encrypted data part;
- *pPart* points to the location that receives the recovered data part;
- *pulPartLen* points to the location that holds the length of the recovered data part.

Notes:

- The decryption operation must have been initialized with C_DecryptInit. This function may be called any number of times in succession. A call to C_DecryptUpdate which results in an error other than CKR_BUFFER_TOO_SMALL terminates the current decryption operation.
- The ciphertext and plaintext can be in the same place, i.e., it is OK if pEncryptedPart and pPart point to the same location.

Return values:

CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_ENCRYPTED_DATA_INVALID, CKR_ENCRYPTED_DATA_LEN_RANGE, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN.

12.6.4 C_DecryptFinal

`C_DecryptFinal` finishes a multiple-part decryption operation.

```
CK_DEFINE_FUNCTION(CK_RV, C_DecryptFinal)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pLastPart,
  CK_ULONG_PTR pulLastPartLen
);
```

Parameters:

- *hSession* is the session's handle;
- *pLastPart* points to the location that receives the last recovered data part, if any;
- *pulLastPartLen* points to the location that holds the length of the last recovered data part.

Notes:

- The decryption operation must have been initialized with `C_DecryptInit`. A call to `C_DecryptFinal` always terminates the active decryption operation unless it returns CKR_BUFFER_TOO_SMALL or is a successful call (i.e., one which returns CKR_OK) to determine the length of the buffer needed to hold the plaintext.
- If the input ciphertext data cannot be decrypted because it has an inappropriate length, then either CKR_ENCRYPTED_DATA_INVALID or CKR_ENCRYPTED_DATA_LEN_RANGE may be returned.

Return values:

CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_ENCRYPTED_DATA_INVALID, CKR_ENCRYPTED_DATA_LEN_RANGE, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN.

12.7 Message Digesting Functions

12.7.1 C_DigestInit

C_DigestInit initializes a message-digesting operation.

```
CK_DEFINE_FUNCTION(CK_RV, C_DigestInit)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism
);
```

Parameters:

- *hSession* is the session's handle;
- *pMechanism* points to the digesting mechanism.

Notes:

After calling C_DigestInit, the application can either call C_Digest to digest data in a single part; or call C_DigestUpdate zero or more times, followed by C_DigestFinal, to digest data in multiple parts. The message-digesting operation is active until the application uses a call to C_Digest or C_DigestFinal to actually obtain the message digest. To process additional data (in single or multiple parts), the application must call C_DigestInit again.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN.

12.7.2 C_Digest

C_Digest digests data in a single part.

```
CK_DEFINE_FUNCTION(CK_RV, C_Digest)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pData,
  CK_ULONG ulDataLen,
  CK_BYTE_PTR pDigest,
  CK_ULONG_PTR pulDigestLen
);
```

Parameters:

- *hSession* is the session's handle,

- *pData* points to the data;
- *ulDataLen* is the length of the data;
- *pDigest* points to the location that receives the message digest;
- *pulDigestLen* points to the location that holds the length of the message digest.

Notes:

- The digest operation must have been initialized with `C_DigestInit`. A call to `C_Digest` always terminates the active digest operation unless it returns CKR_BUFFER_TOO_SMALL or is a successful call (i.e., one which returns CKR_OK) to determine the length of the buffer needed to hold the message digest.
- `C_Digest` cannot be used to terminate a multi-part operation, and must be called after `C_DigestInit` without intervening `C_DigestUpdate` calls.
- The input data and digest output can be in the same place, i.e., it is OK if pData and pDigest point to the same location.
- `C_Digest` is equivalent to a sequence of `C_DigestUpdate` operations followed by `C_DigestFinal`.

Note: This function is performed with software library, not with the smart card.

Return values:

CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

12.7.3 C_DigestUpdate

`C_DigestUpdate` continues a multiple-part message-digesting operation, processing another data part.

```
CK_DEFINE_FUNCTION(CK_RV, C_DigestUpdate)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pPart,
  CK_ULONG ulPartLen
);
```

Parameters:

- *hSession* is the session's handle,
- *pPart* points to the data part;
- *ulPartLen* is the length of the data part.

Notes:

The message-digesting operation must have been initialized with C_DigestInit. Calls to this function and C_Digest may be interspersed any number of times in any order. A call to C_DigestUpdate which results in an error terminates the current digest operation.

Note: This function is performed with software library, not with the smart card.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

12.7.4 C_DigestFinal

C_DigestFinal finishes a multiple-part message-digesting operation, returning the message digest.

```
CK_DEFINE_FUNCTION(CK_RV, C_DigestFinal)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pDigest,
  CK_ULONG_PTR pulDigestLen
);
```

Parameters:

- *hSession* is the session's handle;
- *pDigest* points to the location that receives the message digest;
- *pulDigestLen* points to the location that holds the length of the message digest.

Notes:

The digest operation must have been initialized with C_DigestInit. A call to C_DigestFinal always terminates the active digest operation unless it returns CKR_BUFFER_TOO_SMALL or is a successful call (i.e., one which returns CKR_OK) to determine the length of the buffer needed to hold the message digest.

Return values:

CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

12.7.5 C_DigestKey

Function not supported.

12.8 Signing and MACing Functions

12.8.1 C_SignInit

C_SignInit initializes a signature operation, where the signature is an appendix to the data.

```
CK_DEFINE_FUNCTION(CK_RV, C_SignInit)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hKey
);
```

Parameters:

- *hSession* is the session's handle;
- *pMechanism* points to the signature mechanism;
- *hKey* is the handle of the signature key.

Notes:

- The CKA_SIGN attribute of the signature key, which indicates whether the key supports signatures with appendix, must be CK_TRUE.
- After calling C_SignInit, the application can either call C_Sign to sign in a single part; or call C_SignUpdate one or more times, followed by C_SignFinal, to sign data in multiple parts. The signature operation is active until the application uses a call to C_Sign or C_SignFinal to actually obtain the signature. To process additional data (in single or multiple parts), the application must call C_SignInit again.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_KEY_FUNCTION_NOT_PERMITTED, CKR_KEY_HANDLE_INVALID, CKR_KEY_SIZE_RANGE, CKR_KEY_TYPE_INCONSISTENT, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN.

12.8.2 C_Sign

C_Sign signs data in a single part, where the signature is an appendix to the data.

```
CK_DEFINE_FUNCTION(CK_RV, C_Sign)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pData,
  CK_ULONG ulDataLen,
  CK_BYTE_PTR pSignature,
  CK_ULONG_PTR pulSignatureLen
);
```

Parameters:

- *hSession* is the session's handle;
- *pData* points to the data;
- *ulDataLen* is the length of the data;
- *pSignature* points to the location that receives the signature;
- *pulSignatureLen* points to the location that holds the length of the signature.

Notes:

- The signing operation must have been initialized with C_SignInit. A call to C_Sign always terminates the active signing operation unless it returns CKR_BUFFER_TOO_SMALL or is a successful call (i.e., one which returns CKR_OK) to determine the length of the buffer needed to hold the signature.
- C_Sign cannot be used to terminate a multi-part operation, and must be called after C_SignInit without intervening C_SignUpdate calls.
- For most mechanisms, C_Sign is equivalent to a sequence of C_SignUpdate operations followed by C_SignFinal.

Return values:

CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DATA_INVALID, CKR_DATA_LEN_RANGE, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN, CKR_FUNCTION_REJECTED.

12.8.3 C_SignUpdate

C_SignUpdate continues a multiple-part signature operation, processing another data part.

```
CK_DEFINE_FUNCTION(CK_RV, C_SignUpdate)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pPart,
  CK_ULONG ulPartLen
);
```

Parameters:

- *hSession* is the session's handle,
- *pPart* points to the data part;
- *ulPartLen* is the length of the data part.

Notes:

The signature operation must have been initialized with C_SignInit. This function may be called any number of times in succession. A call to C_SignUpdate which results in an error terminates the current signature operation.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DATA_LEN_RANGE, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN.

12.8.4 C_SignFinal

C_SignFinal finishes a multiple-part signature operation, returning the signature.

```
CK_DEFINE_FUNCTION(CK_RV, C_SignFinal)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pSignature,
  CK_ULONG_PTR pulSignatureLen
);
```

Parameters:

- *hSession* is the session's handle;
- *pSignature* points to the location that receives the signature;
- *pulSignatureLen* points to the location that holds the length of the signature.

Notes:

The signing operation must have been initialized with C_SignInit. A call to C_SignFinal always terminates the active signing operation unless it returns CKR_BUFFER_TOO_SMALL or is a successful call (i.e., one which returns CKR_OK) to determine the length of the buffer needed to hold the signature.

Return values:

CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DATA_LEN_RANGE, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN, CKR_FUNCTION_REJECTED.

12.8.5 C_SignRecoverInit

Function not supported.

12.8.6 C_SignRecover

Function not supported.

12.9 Verifying Signatures and MACs Functions

12.9.1 C_VerifyInit

C_VerifyInit initializes a verification operation, where the signature is an appendix to the data.

```
CK_DEFINE_FUNCTION(CK_RV, C_VerifyInit)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hKey
);
```

Parameters:

- *hSession* is the session's handle;
- *pMechanism* points to the structure that specifies the verification mechanism;
- *hKey* is the handle of the verification key.

Notes:

- The CKA_VERIFY attribute of the verification key, which indicates whether the key supports verification where the signature is an appendix to the data, must be CK_TRUE.

- After calling C_VerifyInit, the application can either call C_Verify to verify a signature on data in a single part; or call C_VerifyUpdate one or more times, followed by C_VerifyFinal, to verify a signature on data in multiple parts. The verification operation is active until the application calls C_Verify or C_VerifyFinal. To process additional data (in single or multiple parts), the application must call C_VerifyInit again.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_KEY_FUNCTION_NOT_PERMITTED, CKR_KEY_HANDLE_INVALID, CKR_KEY_SIZE_RANGE, CKR_KEY_TYPE_INCONSISTENT, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN.

12.9.2 C_Verify

C_Verify verifies a signature in a single-part operation, where the signature is an appendix to the data.

```
CK_DEFINE_FUNCTION(CK_RV, C_Verify)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pData,
  CK_ULONG ulDataLen,
  CK_BYTE_PTR pSignature,
  CK_ULONG ulSignatureLen
);
```

Parameters:

- *hSession* is the session's handle;
- *pData* points to the data;
- *ulDataLen* is the length of the data;
- *pSignature* points to the signature;
- *ulSignatureLen* is the length of the signature.

Notes:

- The verification operation must have been initialized with C_VerifyInit. A call to C_Verify always terminates the active verification operation.
- A successful call to C_Verify should return either the value CKR_OK (indicating that the supplied signature is valid) or CKR_SIGNATURE_INVALID (indicating that the supplied signature is invalid). If the signature can be seen to be invalid purely on the basis of its length, then CKR_SIGNATURE_LEN_RANGE should be returned. In any of these cases, the active signing operation is terminated.

- C_Verify cannot be used to terminate a multi-part operation, and must be called after C_VerifyInit without intervening C_VerifyUpdate calls.
- For most mechanisms, C_Verify is equivalent to a sequence of C_VerifyUpdate operations followed by C_VerifyFinal.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DATA_INVALID, CKR_DATA_LEN_RANGE, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SIGNATURE_INVALID, CKR_SIGNATURE_LEN_RANGE.

12.9.3 C_VerifyUpdate

Function not supported.

12.9.4 C_VerifyFinal

Function not supported.

12.9.5 C_VerifyRecoverInit

Function not supported.

12.9.6 C_VerifyRecover

Function not supported.

12.10 Dual-Function Cryptographic Functions

12.10.1 C_DigestEncryptUpdate

Function not supported.

12.10.2 C_DecryptDigestUpdate

Function not supported.

12.10.3 C_SignEncryptUpdate

Function not supported.

12.10.4 C_DecryptVerifyUpdate

Function not supported.

12.11 Key Management Functions

12.11.1 C_GenerateKey

`C_GenerateKey` generates a secret key or set of domain parameters, creating a new object.

```
CK_DEFINE_FUNCTION(CK_RV, C_GenerateKey)(
  CK_SESSION_HANDLE hSession
  CK_MECHANISM_PTR pMechanism,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulCount,
  CK_OBJECT_HANDLE_PTR phKey
);
```

Parameters:

- *hSession* is the session's handle;
- *pMechanism* points to the generation mechanism;
- *pTemplate* points to the template for the new key or set of domain parameters;
- *ulCount* is the number of attributes in the template;
- *phKey* points to the location that receives the handle of the new key or set of domain parameters.

Notes:

- If the generation mechanism is for domain parameter generation, the CKA_CLASS attribute will have the value CKO_DOMAIN_PARAMETERS; otherwise, it will have the value CKO_SECRET_KEY.
- Since the type of key or domain parameters to be generated is implicit in the generation mechanism, the template does not need to supply a key type. If it does supply a key type which is inconsistent with the generation mechanism, `C_GenerateKey` fails and returns the error code CKR_TEMPLATE_INCONSISTENT. The CKA_CLASS attribute is treated similarly.
- If a call to `C_GenerateKey` cannot support the precise template supplied to it, it will fail and return without creating an object.
- The object created by a successful call to `C_GenerateKey` will have its CKA_LOCAL attribute set to CK_TRUE.

Return values:

CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_READ_ONLY, CKR_ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_TEMPLATE_INCOMPLETE, CKR_TEMPLATE_INCONSISTENT, CKR_TOKEN_WRITE_PROTECTED, CKR_USER_NOT_LOGGED_IN.

12.11.2 C_GenerateKeyPair

`C_GenerateKeyPair` generates a public/private key pair, creating new key objects.

```
CK_DEFINE_FUNCTION(CK_RV, C_GenerateKeyPair)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_ATTRIBUTE_PTR pPublicKeyTemplate,
  CK_ULONG ulPublicKeyAttributeCount,
  CK_ATTRIBUTE_PTR pPrivateKeyTemplate,
  CK_ULONG ulPrivateKeyAttributeCount,
  CK_OBJECT_HANDLE_PTR phPublicKey,
  CK_OBJECT_HANDLE_PTR phPrivateKey
);
```

Parameters:

- *hSession* is the session's handle;
- *pMechanism* points to the key generation mechanism;
- *pPublicKeyTemplate* points to the template for the public key;
- *ulPublicKeyAttributeCount* is the number of attributes in the public-key template;
- *pPrivateKeyTemplate* points to the template for the private key;
- *ulPrivateKeyAttributeCount* is the number of attributes in the private-key template;
- *phPublicKey* points to the location that receives the handle of the new public key;
- *phPrivateKey* points to the location that receives the handle of the new private key.

Notes:

- Since the types of keys to be generated are implicit in the key pair generation mechanism, the templates do not need to supply key types. If one of the templates does supply a key type which is inconsistent with the key generation mechanism, `C_GenerateKeyPair` fails and returns the error code CKR_TEMPLATE_INCONSISTENT. The CKA_CLASS attribute is treated similarly.

- If a call to C_GenerateKeyPair cannot support the precise templates supplied to it, it will fail and return without creating any key objects.
- A call to C_GenerateKeyPair will never create just one key and return. A call can fail, and create no keys; or it can succeed, and create a matching public/private key pair.
- The key objects created by a successful call to C_GenerateKeyPair will have their CKA_LOCAL attributes set to CK_TRUE.

Return values:

CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_READ_ONLY, CKR_ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_DOMAIN_PARAMS_INVALID, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_TEMPLATE_INCOMPLETE, CKR_TEMPLATE_INCONSISTENT, CKR_TOKEN_WRITE_PROTECTED, CKR_USER_NOT_LOGGED_IN.

12.11.3 C_WrapKey

C_WrapKey wraps (i.e., encrypts) a private or secret key.

```
CK_DEFINE_FUNCTION(CK_RV, C_WrapKey)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hWrappingKey,
  CK_OBJECT_HANDLE hKey,
  CK_BYTE_PTR pWrappedKey,
  CK_ULONG_PTR pulWrappedKeyLen
);
```

Parameters:

- *hSession* is the session's handle;
- *pMechanism* points to the wrapping mechanism;
- *hWrappingKey* is the handle of the wrapping key;
- *hKey* is the handle of the key to be wrapped;
- *pWrappedKey* points to the location that receives the wrapped key;
- *pulWrappedKeyLen* points to the location that receives the length of the wrapped key.

Notes:

- The CKA_WRAP attribute of the wrapping key, which indicates whether the key supports wrapping, must be CK_TRUE. The CKA_EXTRACTABLE attribute of the key to be wrapped must also be CK_TRUE.
- If the key to be wrapped cannot be wrapped for some token-specific reason, despite its having its CKA_EXTRACTABLE attribute set to CK_TRUE, then `C_WrapKey` fails with error code CKR_KEY_NOT_WRAPPABLE. If it cannot be wrapped with the specified wrapping key and mechanism solely because of its length, then `C_WrapKey` fails with error code CKR_KEY_SIZE_RANGE.
- `C_WrapKey` can be used in the following situations:
  - To wrap any secret key with a public key that supports encryption and decryption.
  - To wrap any secret key with any other secret key. Consideration must be given to key size and mechanism strength or the token may not allow the operation.
  - To wrap a private key with any secret key (for legacy).

Return values:

CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_KEY_HANDLE_INVALID, CKR_KEY_NOT_WRAPPABLE, CKR_KEY_SIZE_RANGE, CKR_KEY_UNEXTRACTABLE, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN, CKR_WRAPPING_KEY_HANDLE_INVALID, CKR_WRAPPING_KEY_SIZE_RANGE, CKR_WRAPPING_KEY_TYPE_INCONSISTENT.

12.11.4 C_UnwrapKey

`C_UnwrapKey` unwraps (i.e. decrypts) a wrapped key, creating a new private key or secret key object.

```
CK_DEFINE_FUNCTION(CK_RV, C_UnwrapKey)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hUnwrappingKey,
  CK_BYTE_PTR pWrappedKey,
  CK_ULONG ulWrappedKeyLen,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulAttributeCount,
  CK_OBJECT_HANDLE_PTR phKey
);
```

Parameters:

- *hSession* is the session's handle;

- *pMechanism* points to the unwrapping mechanism;
- *hUnwrappingKey* is the handle of the unwrapping key;
- *pWrappedKey* points to the wrapped key;
- *ulWrappedKeyLen* is the length of the wrapped key;
- *pTemplate* points to the template for the new key;
- *ulAttributeCount* is the number of attributes in the template;
- *phKey* points to the location that receives the handle of the recovered key.

Notes:

- The CKA_UNWRAP attribute of the unwrapping key, which indicates whether the key supports unwrapping, must be CK_TRUE.
- If the new key is a secret key, it will have the CKA_ALWAYS_SENSITIVE attribute set to CK_FALSE, and the CKA_NEVER_EXTRACTABLE attribute set to CK_FALSE. The CKA_EXTRACTABLE attribute is by default set to CK_TRUE.
- If the new key is a private key, it will have the CKA_ALWAYS_SENSITIVE attribute set to CK_TRUE, and the CKA_NEVER_EXTRACTABLE attribute set to CK_TRUE. The CKA_EXTRACTABLE attribute is by default set to CK_FALSE.
- Some mechanisms may modify, or attempt to modify the contents of the pMechanism structure at the same time that the key is unwrapped.
- If a call to `C_UnwrapKey` cannot support the precise template supplied to it, it will fail and return without creating any key object.
- The key object created by a successful call to `C_UnwrapKey` will have its CKA_LOCAL attribute set to CK_FALSE.

Return values:

CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_READ_ONLY, CKR_ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_DOMAIN_PARAMS_INVALID, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_TEMPLATE_INCOMPLETE, CKR_TEMPLATE_INCONSISTENT, CKR_TOKEN_WRITE_PROTECTED, CKR_UNWRAPPING_KEY_HANDLE_INVALID, CKR_UNWRAPPING_KEY_SIZE_RANGE, CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT, CKR_USER_NOT_LOGGED_IN, CKR_WRAPPED_KEY_INVALID, CKR_WRAPPED_KEY_LEN_RANGE.

12.11.5 C_DeriveKey

Function not supported.

12.12 Random Number Generation Functions

12.12.1 C_SeedRandom

Function not supported.

12.12.2 C_GenerateRandom

Function not supported.

12.13 Parallel Function Management Functions

12.13.1 C_FunctionStatus

Function not supported.

12.13.2 C_CancelFunction

Function not supported.

12.14 Proprietary Functions

12.14.1 C_VerifyUnblockPINInit

`C_VerifyUnblockPINInit` initializes an Unblock PIN verification operation.

```
CK_DEFINE_FUNCTION(CK_RV, C_VerifyUnblockPINInit)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hKey
);
```

Parameters:

- *hSession* is the session's handle;
- *pMechanism* points to the structure that specifies the verification mechanism;
- *hKey* is the handle of the verification key.

Notes:

After calling C_VerifyUnblockPINInit, the application can call C_VerifyUnblockPIN to verify the unblock code in a single part. The verification operation is active until the application calls C_VerifyUnblockPIN. To process additional data, the application must call C_VerifyUnblockPINInit again.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_KEY_FUNCTION_NOT_PERMITTED, CKR_KEY_HANDLE_INVALID, CKR_KEY_SIZE_RANGE, CKR_KEY_TYPE_INCONSISTENT, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN.

12.14.2 C_VerifyUnblockPIN

C_VerifyUnblockPIN verifies the unblock code supplied in a single-part operation.

```
CK_DEFINE_FUNCTION(CK_RV, C_VerifyUnblockPIN)(
  CK_SESSION_HANDLE hSession,
  CK_UTF8CHAR_PTR pPin,
  CK_ULONG ulPinLen,
  CK_BYTE_PTR pSignature,
  CK_ULONG ulSignatureLen
);
```

Parameters:

- *hSession* is the session's handle;
- *pPIN* points to the new user's PIN;
- *ulPinLen* is the length of the PIN;
- *pSignature* points to the unblock code;
- *ulSignatureLen* is the length of the unblock code.

Notes:

- The verification operation must have been initialized with C_VerifyUnblockPINInit. A call to C_VerifyUnblockPIN always terminates the active verification operation.
- A successful call to C_VerifyUnblockPIN should return either the value CKR_OK (indicating that the supplied unblock code is valid, and PIN is unblocked) or CKR_SIGNATURE_INVALID (indicating that the supplied unblock code is invalid, and PIN is still blocked). If the signature can be seen to be invalid purely on the basis of its length, then CKR_SIGNATURE_LEN_RANGE should be returned. In any of these cases, the active signing operation is terminated.

Return values:

CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DATA_INVALID, CKR_DATA_LEN_RANGE, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SIGNATURE_INVALID, CKR_SIGNATURE_LEN_RANGE.

12.15 Legacy Proprietary Functions

12.15.1 C_UnlockPIN

Function is deprecated.

12.15.2 C_ResetToken

Function is deprecated.

12.15.3 C_GetCUID/C_CetCUID

Function is deprecated.

13.0 One-Time Password Support

This section describes support of One-Time Password (OTP) through PKCS#11 API in order to supersede the ACOMX API.

13.1 Mechanisms

The following table shows which OTP mechanisms are supported by different cryptographic operations.

Lines in grey correspond to unsupported mechanisms.

	Functions						
Mechanism	Encrypt & Decrypt	Sign & Verify	SR & VR1	Digest	Gen. Key/Key Pair	Wrap & Unwrap	Derive
CKM_SECURID_KEY_GEN					✓		
CKM_SECURID		✓					
CKM_HOTP_KEY_GEN					✓		
CKM_HOTP		✓					
CKM_ACTI_KEY_GEN					✓		
CKM_ACTI		✓					

13.2 Key Objects

OTP key objects (object class CKO_OTP_KEY) hold secret keys used by OTP tokens. Key type for ActivIdentity ACTI secret keys is CKK_ACTI.

13.2.1 Key Objects Attributes

This table lists the common OTP key attributes:

Attribute	**Data type**	**Meaning**
CKA_OTP_FORMAT	CK_ULONG	Format of OTP values produced with this key: • CK_OTP_FORMAT_DECIMAL = Decimal (default) (UTF8-encoded) • CK_OTP_FORMAT_HEXADECIMAL = Hexadecimal (UTF8-encoded) • CK_OTP_FORMAT_ALPHANUMERIC =

Attribute	Data type	Meaning
		Alphanumeric (UTF8-encoded) • CK_OTP_FORMAT_BINARY = Only binary values.
CKA_OTP_LENGTH	CK_ULONG	Default length of OTP values (in the CKA_OTP_FORMAT) produced with this key.
CKA_OTP_USER_FRIENDLY_MODE	CK_BOOL	CK_TRUE (meaning the token is capable of returning OTPs suitable for human consumption)
CKA_OTP_CHALLENGE_REQUIREMENT	CK_ULONG	Parameter requirements when generating OTP values with this key: • CK_OTP_PARAM_MANDATORY = A challenge must be supplied. • CK_OTP_PARAM_OPTIONAL = A challenge may be supplied but need not be. (should never happen in this middleware) • CK_OTP_PARAM_IGNORED = A challenge, if supplied, will be ignored.
CKA_OTP_TIME_REQUIREMENT	CK_ULONG	Parameter requirements when generating OTP values with this key: • CK_OTP_PARAM_MANDATORY = A time value must be supplied. • CK_OTP_PARAM_OPTIONAL = A time value may be supplied but need not be. • CK_OTP_PARAM_IGNORED = A time value, if supplied, will be ignored.
CKA_OTP_COUNTER_REQUIREMENT	CK_ULONG	Parameter requirements when generating OTP values with this key: • CK_OTP_PARAM_MANDATORY = A counter value must be supplied. • CK_OTP_PARAM_OPTIONAL = A counter value may be supplied but need not be. • CK_OTP_PARAM_IGNORED = A counter value, if supplied, will be ignored.
CKA_OTP_COUNTER	Byte array	Value of the associated internal counter. Default value is empty (i.e. ulValueLen = 0). For ACTI keys, value shall be an 8 bytes unsigned integer in big endian (i.e. network byte order) form.
CKA_OTP_TIME	RFC 2279 string	Value of the associated internal UTC time in the form YYYYMMDDhhmmss. Default value is empty (i.e. ulValueLen= 0).
CKA_OTP_SERVICE_IDENTIFI	RFC 2279	Text string that identifies a service that may validate OTPs

Attribute	Data type	Meaning
ER	string	generated by this key. Default value is empty (i.e. ulValueLen = 0).

For OTP tokens with multiple keys, the keys may be enumerated using C_FindObjects.

The CKA_OTP_SERVICE_IDENTIFIER attribute is used to distinguish between keys.

13.2.2 Proprietary Key Attributes

Attribute	Data type	Meaning
CKA_OTP_DERIVATION	CK_BOOL	Set to CK_TRUE if the derivation flag is set (flag indicating whether the secret key used in the synchronous authentication algorithm is derived at each authentication). Default value: set to CK_FALSE.
CKA_OTP_COUNTER_INCREMENT	CK_BOOL	Set to CK_TRUE if the counter increment flag is set (flag indicating whether the counter used in the synchronous authentication algorithm is increased at each authentication). Default value: set to CK_FALSE.

Proprietary key attributes are defined as follows:

```
#define CKA_OTP_DERIVATION          CKA_VENDOR_DEFINED + 0x00000330
#define CKA_OTP_COUNTER_INCREMENT   CKA_VENDOR_DEFINED + 0x00000331
```

13.3 Mechanism Parameters

13.3.1 Parameters Type

Parameter	Data type	Meaning
CK_OTP_CHALLENGE	Byte array	Challenge to use when computing or verifying challenge-based OTP values.
CK_OTP_TIME	RFC 2279 string	UTC time value in the form YYYYMMDDhhmmss to use when computing or verifying time-based OTP values.
CK_OTP_COUNTER	Byte array	Counter value to use when computing or verifying counter-based OTP values.
CK_OTP_FLAGS	CK_FLAGS	Bit flags indicating the characteristics of the sought OTP as defined below.
CK_OTP_OUTPUT_LENGTH	CK_ULONG	Desired output length (overrides any default value). A

Parameter	Data type	Meaning
		Cryptoki library will return CKR_MECHANISM_PARAM_INVALID if a provided length value is not supported.
CK_OTP_VALUE	Byte array	An actual OTP value. This parameter type is intended for C_Sign output, see below.

13.3.2 OTP Mechanisms Flags

Attribute	Mask	Meaning
CKF_EXCLUDE_TIME	0x00000002	True (i.e. set) if the OTP computation must not include a time value. Will have an effect only on mechanisms that do include a time value in the OTP computation and then only if the mechanism (and token) allows exclusion of this value. A Cryptoki library shall return CKR_MECHANISM_PARAM_INVALID if exclusion of the value is not allowed.
CKF_EXCLUDE_COUNTER	0x00000004	True (i.e. set) if the OTP computation must not include a counter value. Will have an effect only on mechanisms that do include a counter value in the OTP computation and then only if the mechanism (and token) allows exclusion of this value. A Cryptoki library shall return CKR_MECHANISM_PARAM_INVALID if exclusion of the value is not allowed.
CKF_EXCLUDE_CHALLENGE	0x00000008	True (i.e. set) if the OTP computation must not include a challenge. Will have an effect only on mechanisms that do include a challenge in the OTP computation and then only if the mechanism (and token) allows exclusion of this value. A Cryptoki library shall return CKR_MECHANISM_PARAM_INVALID if exclusion of the value is not allowed.
CKF_USER_FRIENDLY_OTP	0x00000020	True (i.e. set) if the OTP returned shall be in a form suitable for human consumption. If this flag is set, and the call is successful, then the returned CK_OTP_VALUE shall be a UTF8-encoded printable string. A Cryptoki library shall return CKR_MECHANISM_PARAM_INVALID if this flag is set when CKA_OTP_USER_FRIENDLY_MODE for the key in question is CK_FALSE.

13.3.3 Parameters Structures

CK_OTP_PARAM is a structure that includes the type, value, and length of an OTP parameter. It is defined as follows:

```
typedef struct CK_OTP_PARAM {
CK_PARAM_TYPE type;
CK_VOID_PTR pValue;
CK_ULONG ulValueLen;
} CK_OTP_PARAM;
```

The fields of the structure have the following meanings:

- *type* is the parameter type
- *pValue* is the pointer to the value of the parameter
- *ulValueLen* is the length in bytes of the value

CK_OTP_PARAMS is a structure that is used to provide parameters for OTP mechanisms in a generic fashion. It is defined as follows:

```
typedef struct CK_OTP_PARAMS {
CK_OTP_PARAM_PTR pParams;
CK_ULONG ulCount;
} CK_OTP_PARAMS;
```

The fields of the structure have the following meanings:

- *pParams* is the pointer to an array of OTP parameters
- *ulCount* is the number of parameters in the array

13.4 OTP Functions

The functions used to compute an OTP are:

- `C_SignInit`
- `C_Sign`

in the context and conditions that have previously been described.

When calling `C_SignInit` with the CKM_ACTI mechanism that takes a CK_OTP_PARAMS structure as a parameter, the CK_OTP_PARAMS structure shall be populated in accordance with the CKA_OTP_X_REQUIREMENT key attributes for the identified key, where X is CHALLENGE, TIME, or COUNTER.

CK_OTP_SIGNATURE_INFO is a structure that is returned by all OTP mechanisms in successful calls to C_Sign. The structure informs applications of actual parameter values used in particular OTP computations in addition to the OTP value itself.

It is used by all mechanisms for which the key belongs to the class CKO_OTP_KEY and is defined as follows:

```
typedef struct CK_OTP_SIGNATURE_INFO {
CK_OTP_PARAM_PTR pParams;
CK_ULONG ulCount;
} CK_OTP_SIGNATURE_INFO;
```

The fields of the structure have the following meanings:

- *pParams* is the pointer to an array of OTP parameter values
- *ulCount* is the number of parameters in the array

After successful calls to C_Sign with an OTP mechanism, the pSignature parameter will be set to point to a CK_OTP_SIGNATURE_INFO structure. One of the parameters in this structure will be the OTP value itself, identified with the CK_OTP_VALUE tag. Other parameters may be present for informational purposes, e.g. the actual time used in the OTP calculation.

In order to simplify OTP validations, authentication protocols may permit authenticating parties to send some or all of these parameters in addition to OTP values themselves. Applications should therefore check for their presence in returned CK_OTP_SIGNATURE_INFO values whenever such circumstances apply.

Since C_Sign follows the convention described above (7.8.2) on producing output, a call to C_Sign with pSignature set to NULL_PTR will return (in the pulSignatureLen parameter) the required number of bytes to hold the CK_OTP_SIGNATURE_INFO structure as well as all the data in all its CK_OTP_PARAM components.

If an application allocates a memory block based on this information, it shall therefore not subsequently de-allocate components of such a received value but rather de-allocate the complete CK_OTP_PARAMS structure itself. A Cryptoki library that is called with a non-NULL pSignature pointer will assume that it points to a contiguous memory block of the size indicated by the pulSignatureLen parameter.

When signing or verifying using the CKM_ACTI mechanism, pData shall be set to NULL_PTR and ulDataLen shall be set to 0.

14.0 Unblock PIN Support

Unblock PIN relies on proprietary mechanisms.

14.1 Mechanisms

The following table shows which Unblock PIN mechanisms are supported by different cryptographic operations.

	Functions	
Mechanism	C_VerifyUnblockPINInit & C_VerifyUnblockPIN	UnlockPIN (legacy)
CKM_UNBLOCK_PIN_STATIC	✓	✓
CKM_UNBLOCK_PIN_DYNAMIC	✓	

These proprietary mechanisms are defined as follows:

```
#define CKM_UNBLOCK_PIN_STATIC      CKM_ACTI + 0x00000801
#define CKM_UNBLOCK_PIN_DYNAMIC     CKM_ACTI + 0x00000802
```

14.2 Unblock PIN Object Definition

Unblock PIN object is defined as:

```
#define CKO_UNBLOCK_PIN             CKO_VENDOR_DEFINED + 0x00000810
```

14.3 Object Attributes

Attribute	Data type	Meaning
CKA_UNBLOCK_PIN_CHALLENGE_REQUIREMENT	CK_ULONG	Parameter requirements when unblock PIN code values with challenge / response: CK_OTP_PARAM_MANDATORY = Challenge/Response unblock PIN. The challenge must be retrieved with C_GetAttributeValue on attribute CKA_UNBLOCK_PIN_CHALLENGE and encrypted using the unblock key before the response is send using C_VerifyUnblockPIN. CK_OTP_PARAM_OPTIONAL = RFU. CK_OTP_PARAM_IGNORED = Static unblock PIN. C_VerifyUnblockPIN should be called directly with the unlock code

Attribute	Data type	Meaning
CKA_UNBLOCK_PIN_CHALLENGE	Byte array	Only defined for challenge/response unblock. Value of the challenge to be encrypted using the unblock key.

This proprietary attribute is defined as follows:

```
#define CKA_UNBLOCK_PIN_CHALLENGE_REQUIREMENT    CKA_VENDOR_DEFINED + 0x00000340
#define CKA_UNBLOCK_PIN_CHALLENGE                CKA_VENDOR_DEFINED + 0x00000341
```

14.3.1 Common Storage Object Attributes

Attribute	Data Type	Meaning
CKA_TOKEN	CK_BBOOL	CK_TRUE
CKA_PRIVATE	CK_BBOOL	CK_FALSE
CKA_MODIFIABLE	CK_BBOOL	CK_FALSE
CKA_LABEL	RFC2279 string	Empty
CKA_CLASS	CK_OBJECT_CLASS	CKO_UNBLOCK_PIN

14.4 Static Unlock Key Object Definition (Legacy)

For standalone cards that have a static unlock key that is still available to read, the following session object is returned.

14.4.1 Secret Key Attributes

Attribute	Data Type	Meaning
CKA_SENSITIVE	CK_BBOOL	CK_FALSE
CKA_ENCRYPT	CK_BBOOL	CK_TRUE
CKA_DECRYPT	CK_BBOOL	CK_ TRUE
CKA_WRAP	CK_BBOOL	CK_TRUE
CKA_UNWRAP	CK_BBOOL	CK_TRUE
CKA_EXTRACTABLE	CK_BBOOL	CK_TRUE
CKA_ALWAYS_SENSITIVE	CK_BBOOL	CK_FALSE
CKA_NEVER_EXTRACTABLE	CK_BBOOL	CK_FALSE

Attribute	Data Type	Meaning
CKA_ALWAYS_AUTHENTICATE	CK_BBOOL	CK_FALSE
CKA_VALUE	Byte Array	Value of the unlock key

14.4.2 Common Key Attributes

Attribute	Data Type	Meaning
CKA_KEY_TYPE	CK_KEY_TYPE	CKK_GENERIC_SECRET.
CKA_ID	Byte array	empty
CKA_LOCAL	CK_BBOOL	CK_TRUE
CKA_KEY_GEN_MECHANISM	CK_MECHANISM_TYPE	CK_UNAVAILABLE_INFORMATION
CKA_ALLOWED_MECHANISMS	CK_MECHANISM_TYPE _PTR, pointer to a CK_MECHANISM_TYPE array	None

14.4.3 Common Storage Object Attributes

Attribute	Data Type	Meaning
CKA_TOKEN	CK_BBOOL	CK_FALSE
CKA_PRIVATE	CK_BBOOL	CK_TRUE
CKA_MODIFIABLE	CK_BBOOL	CK_FALSE
CKA_LABEL	RFC2279 string	“Unlock Key”
CKA_CLASS	CK_OBJECT_CLASS	CKO_SECRET_KEY

Appendix A: Terms and Acronyms

This appendix lists terms and acronyms used throughout the full set of the set of technical publications for this product. Not all terms and acronyms appear in all documents in the set.

A.1. Terms

Certificate Authority (CA)

The CA issues and manages security credentials and public keys for message encryption in a networked environment. As part of a Public Key Infrastructure (PKI), a CA checks with a registration authority (RA) to verify information provided by the requestor of a digital certificate. If the RA verifies the requestor's information, the CA issues a certificate.

ActivID Credential Management System (CMS)

Formally known as ActivID Card Management System, ActivID CMS is a web-based, smart card, credential and application lifecycle management system. ActivID CMS augments and works in concert with an enterprise's primary identity management infrastructure components, including popular directory, database, and PKI components.

Challenge

Random number generated by the server API for authentication of a user in the asynchronous (challenge/response) mode.

Cryptographic Service Provider (CSP)

An independent software module that performs cryptography algorithms for authentication, encoding, and encryption.

Discovery mode

Discovery mode enables a calling application to find out the size of the data that will be returned to by making a preliminary discovery call and then making a second call after it allocates a buffer large enough to accommodate the data that will be returned.

End-point card

The PIV standard defines two interfaces for communicating with PIV cards:

- The PIV transitional interface.
- The PIV end-point interface.

A PIV end-point card is a card that implements the second of these interfaces.

Note: The PIV transitional interface is not supported by the PIV API.

Federal Information Processing Standard (FIPS 140-2)

FIPS 140-2 is the standard for crypto-module security. FIPS 140-2 level 3 adds additional requirements to FIPS 140-2 level 2. These requirements concern physical security and a trusted path for entering a Cryptographic Service Provider, such as a PIN. FIPS 140-2 level 3 uses local ports and the key pad to enforce such security.

Federal Information Processing Standard 201 (FIPS 201)

FIPS 201 is the standard for Personal Identity Verification (PIV) cards defined for US Government employees and contractors.

Force change PIN flag

Flag which indicates whether the user must change the PIN on first use of the card.

Integrated circuit chip (ICC)	The chip on the smart card.
Mini Driver	Smart card middleware for the Microsoft platform that works with the Microsoft Base Smart Card CSP (Cryptographic Service Provider). The ActivClient Mini Driver replaces the ActivClient CSP available in previous versions. The Mini Driver architecture provides stronger cryptographic services.
My Digital ID Card (MDIDC)	This CMS component allows end users to access the self-service CMS functions, which includes card and credential management.
One-Time Password (OTP)	A one-time password is a password used only once to authenticate to remote applications. One-Time Passwords are only present on smart cards issued with SKI credentials.
Personal Identification Number (PIN)	The Personal Identification Number (PIN) code used to access an HID Global device's services such as Windows PKI logon, remote access and email signature. HID Global devices can only be used after a correct PIN is entered.
Public Key Infrastructure (PKI)	PKI describes the laws, policies, standards, and software that regulate or manipulate certificates and public and private keys.
Registration Authority (RA)	RA is an authority in a network that verifies user requests for a digital certificate and instructs the CA to issue it. An RA is part of a PKI, a networked system that enables companies and users to exchange information safely and securely.
Symmetric Key Infrastructure (SKI)	SKI keys are used to perform strong authentication on remote applications. SKI keys encrypt passwords in: • Synchronous mode (generates 1 password without any challenge. The server uses the same method to create a password than the smart card) • Asynchronous: encrypts a challenge
Standalone smart card	Smart card with pre-loaded applets issued by the manufacturer.
Unlock code	Value that the card holder needs to provide in order to unlock a locked smart card. Depending upon the smart card unlock mechanism, the unlock code may or may not be different from the unlock key.
Verification	Process in which a signature that was produced by the signing operation is verified.
Weak PIN	A weak PIN is a PIN in which: • The length is less than three characters or digits, or • The difference between each character or digit and the following one is a constant. For example, a PIN that is a sequence of the same number (1111) or an increasing/decreasing sequence of numbers (1234, 4321) is a weak PIN.

 All rights reserved.

A.2. Acronyms

CA Certificate Authority

CAC Common Access Card (for the United States Department of Defense)

CSP Cryptographic Service Provider

CUID Card Unique Identifier

CUID is a number that uniquely identifies a card.

FIPS Federal Information Processing Standard

GAL Global Address List

GP GlobalPlatform

Replaces OpenPlatform (OP).

OTP One-Time Password

PKI Public Key Infrastructure

PIV Personal Identity Verification.

Smart card issued by the United States government to federal employees and contractors.

RA Registration Authority

SKI Symmetric Key Infrastructure

