SafeNet PCIe HSM 6.2.1

SDK Reference Guide

gemalto
security to be free

Document Information

Product Version	6.2.1
Document Part Number	007-011329-010
Release Date	26 July 2016

Revision History

Revision	Date	Reason
A	26 July 2016	Initial release.

Trademarks, Copyrights, and Third-Party Software

Acknowledgements

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org)

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

This product includes software developed by the University of California, Berkeley and its contributors.

This product uses Brian Gladman's AES implementation.

Refer to the End User License Agreement for more information.

Disclaimer

All information herein is either public information or is the property of and owned solely by Gemalto and/or its subsidiaries who shall have and keep the sole right to file patent applications or any other kind of intellectual property protection in connection with such information.

Nothing herein shall be construed as implying or granting to you any rights, by license, grant or otherwise, under any intellectual and/or industrial property rights of or concerning any of Gemalto's information.

This document can be used for informational, non-commercial, internal, and personal use only provided that:

- The copyright notice, the confidentiality and proprietary legend and this full warning notice appear in all copies.
- This document shall not be posted on any publicly accessible network computer or broadcast in any media, and no modification of any part of this document shall be made.

Use for any other purpose is expressly prohibited and may result in severe civil and criminal liabilities.

The information contained in this document is provided "AS IS" without any warranty of any kind. Unless otherwise expressly agreed in writing, Gemalto makes no warranty as to the value or accuracy of information contained herein.

The document could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Furthermore, Gemalto reserves the right to make any change or improvement in the specifications data, information, and the like described herein, at any time.

Gemalto hereby disclaims all warranties and conditions with regard to the information contained herein, including all implied warranties of merchantability, fitness for a particular purpose, title and non-infringement. In no event shall Gemalto be liable, whether in contract, tort or otherwise, for any indirect, special or consequential damages or any damages whatsoever including but not limited to damages resulting from loss of use, data, profits, revenues, or customers, arising out of or in connection with the use or performance of information contained in this document.

Gemalto does not and shall not warrant that this product will be resistant to all possible attacks and shall not incur, and disclaims, any liability in this respect. Even if each product is compliant with current security standards in force on the date of their design, security mechanisms' resistance necessarily evolves according to the state of the art in security and notably under the emergence of new attacks. Under no circumstances, shall Gemalto be held liable for any third party actions and in particular in case of any successful attack against systems or equipment incorporating Gemalto products. Gemalto disclaims any liability with respect to security for direct, indirect, incidental or consequential damages that result from any use of its products. It is further stressed that independent testing and verification by the person using the product is particularly encouraged, especially in any application in which defective, incorrect or insecure functioning could result in damage to persons or property, denial of service, or loss of privacy.

Regulatory Compliance

This product complies with the following regulatory regulations. To ensure compliancy, ensure that you install the products as specified in the installation instructions and use only Gemalto-supplied or approved accessories.

USA, FCC

This device complies with Part 15 of the FCC rules. Operation is subject to the following conditions:

- This device may not cause harmful interference.
- This device must accept any interference received, including interference that may cause undesired operation.

Note: This equipment has been tested and found to comply with the limits for a “Class B” digital device, pursuant to part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation.

If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna
- Increase the separation between the equipment and receiver
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected
- Consult the dealer or an experienced radio/TV technician for help

Changes or modifications not expressly approved by Gemalto could void the user’s authority to operate the equipment.

Canada

This class B digital apparatus meets all requirements of the Canadian interference- causing equipment regulations.

Europe

This product is in conformity with the protection requirements of EC Council Directive 2004/108/EC. Conformity is declared to the following applicable standards for electro-magnetic compatibility immunity and susceptibility; CISPR22 and IEC801. This product satisfies the CLASS B limits of EN 55022.

CONTENTS

PREFACE

About the SDK Reference Guide

This document describes how to use the SafeNet SDK to create applications that interact with SafeNet HSMs. It contains the following chapters:

- "SafeNet SDK Overview" on page 18
- "PKCS#11 Support" on page 26
- "Extensions to PKCS#11" on page 32
- "Supported Mechanisms" on page 49
- "Using the SafeNet SDK" on page 294
- "Design Considerations" on page 309
- "Java Interfaces" on page 346
- "Microsoft Interfaces" on page 370

This preface also includes the following information about this document:

- "Customer Release Notes" below
- "Gemalto Rebranding" below
- "Audience" on the next page
- "Document Conventions" on the next page
- "Support Contacts" on page 17

For information regarding the document status and revision history, see "Document Information" on page 2.

Customer Release Notes

The customer release notes (CRN) provide important information about this release that is not included in the customer documentation. It is strongly recommended that you read the CRN to fully understand the capabilities, limitations, and known issues for this release. You can view or download the latest version of the CRN for this release at the following location:

- http://www.securedbysafenet.com/releasenotes/luna/crn_luna_hsm_6-2-1.pdf

Gemalto Rebranding

In early 2015, Gemalto completed its acquisition of SafeNet, Inc. As part of the process of rationalizing the product portfolios between the two organizations, the Luna name has been removed from the SafeNet HSM product line, with the SafeNet name being retained. As a result, the product names for SafeNet HSMs have changed as follows:

Old product name	New product name
Luna SA HSM	SafeNet Network HSM
Luna PCI-E HSM	SafeNet PCIe HSM
Luna G5 HSM	SafeNet USB HSM
Luna PED	SafeNet PED
Luna Client	SafeNet HSM Client
Luna Dock	SafeNet Dock
Luna Backup HSM	SafeNet Backup HSM
Luna CSP	SafeNet CSP
Luna JSP	SafeNet JSP
Luna KSP	SafeNet KSP

Note: These branding changes apply to the documentation only. The SafeNet HSM software and utilities continue to use the old names.

Audience

This document is intended for personnel responsible for maintaining your organization's security infrastructure. This includes SafeNet HSM users and security officers, key manager administrators, and network administrators.

All products manufactured and distributed by Gemalto are designed to be installed, operated, and maintained by personnel who have the knowledge, training, and qualifications required to safely perform the tasks assigned to them. The information, processes, and procedures contained in this document are intended for use by trained and qualified personnel only.

It is assumed that the users of this document are proficient with security concepts.

Document Conventions

This document uses standard conventions for describing the user interface and for alerting you to important information.

Notes

Notes are used to alert you to important or helpful information. They use the following format:

Note: Take note. Contains important or helpful information.

Cautions

Cautions are used to alert you to important information that may help prevent unexpected results or data loss. They use the following format:

> **CAUTION:** Exercise caution. Contains important information that may help prevent unexpected results or data loss.

Warnings

Warnings are used to alert you to the potential for catastrophic data loss or personal injury. They use the following format:

> **WARNING! Be extremely careful and obey all safety and security measures. In this situation you might do something that could result in catastrophic data loss or personal injury.**

Command Syntax and Typeface Conventions

Format	Convention
bold	The bold attribute is used to indicate the following: • Command-line commands and options (Type dir /p.) • Button names (Click Save As.) • Check box and radio button names (Select the Print Duplex check box.) • Dialog box titles (On the Protect Document dialog box, click Yes.) • Field names (User Name: Enter the name of the user.) • Menu names (On the File menu, click Save.) (Click Menu > Go To > Folders.) • User input (In the Date box, type April 1.)
italics	In type, the italic attribute is used for emphasis or to indicate a related document. (See the *Installation Guide* for more information.)
<variable>	In command descriptions, angle brackets represent variables. You must substitute a value for command line arguments that are enclosed in angle brackets.
[**optional**] [<optional>]	Represent optional **keywords** or <variables> in a command line description. Optionally enter the keyword or <variable> that is enclosed in square brackets, if it is necessary or desirable to complete the task.
{**a**|**b**|**c**} {<a>|<b>|<c>}	Represent required alternate **keywords** or <variables> in a command line description. You must choose one command line argument enclosed within the braces. Choices are separated by vertical (OR) bars.
[**a**|**b**|**c**] [<a>|<b>|<c>]	Represent optional alternate keywords or variables in a command line description. Choose one command line argument enclosed within the braces, if desired. Choices are separated by vertical (OR) bars.

Support Contacts

Contact method	Contact	
Address	Gemalto 4690 Millennium Drive Belcamp, Maryland 21017 USA	
Phone	Global	+1 410-931-7520
	Australia	1800.020.183
	China	(86) 10 8851 9191
	France	0825 341000
	Germany	01803 7246269
	India	000.800.100.4290
	Netherlands	0800.022.2996
	New Zealand	0800.440.359
	Portugal	800.1302.029
	Singapore	800.863.499
	Spain	900.938.717
	Sweden	020.791.028
	Switzerland	0800.564.849
	United Kingdom	0800.056.3158
	United States	(800) 545-6608
Web	www.safenet-inc.com	
Support and Downloads	www.safenet-inc.com/support Provides access to the Gemalto Knowledge Base and quick downloads for various products.	
Technical Support Customer Portal	https://serviceportal.safenet-inc.com Existing customers with a Technical Support Customer Portal account can log in to manage incidents, get the latest software upgrades, and access the Gemalto Knowledge Base.	

1
SafeNet SDK Overview

This chapter provides an overview of the SafeNet Software Development Kit (SDK), a development platform you can use to integrate a SafeNet HSM into your application or system. It contains the following topics:

Supported Cryptographic Algorithms

The K6 Cryptographic engine supports cryptographic algorithms that include:

- RSA
- DSA
- Diffie-Hellman
- DES and triple DES
- MD2 and MD5
- SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
- RC2, RC4 and RC5
- AES
- PBE
- ECC
- ECIES
- ARIA, SEED

Application Programming Interface

The major API provided with SafeNet Product Software Development Kit conforms to RSA Laboratories' Public-Key Cryptography Standards #11 (PKCS #11) v2.20. A set of API services (called PKCS #11 Extensions) designed by SafeNet, augments the services provided by PKCS#11. The API is a library – a DLL in Windows, a shared object in Solaris, AIX and Linux, a shared library in HP-UX – called Chrystoki. Applications wanting to use token services must connect with Chrystoki.

In addition, support is provided for Microsoft's cryptographic APIs (CAPI/CNG) and Oracle's Java Security API.

The extensions to each API enable optimum use of SafeNet hardware for commonly used calls and functions, where the unaugmented API would tend to use software, or to make generic, non-optimized use of available HSMs.

Table 1: SafeNet libraries by platform

Platform	Key name	Libraries
Windows	LibNT	X:\Program Files\SafeNet\LunaClient\cryptoki.dll
		X:\Program Files\SafeNet\LunaClient\cklog201.dll
		X:\Program Files\SafeNet\LunaClient\shim.dll
		X:\Program Files\SafeNet\LunaClient\LunaCSP\LunaCSP.dll
		C:\WINDOWS\system32\SafeNetKSP.dll
Solaris (32-bit)	LibUNIX	/opt/safenet/lunaclient/lib/libCryptoki2.so
		/opt/safenet/lunaclient/lib/libcklog2.so
		/opt/safenet/lunaclient/lib/libshim.so
Solaris (64-bit)	LibUNIX64	/opt/safenet/lunaclient/lib/libCryptoki2_64.so
		/opt/safenet/lunaclient/lib/libcklog2.so
		/opt/safenet/lunaclient/lib/libshim_64.so
Linux (64-bit)	LibUNIX	/usr/safenet/lunaclient/lib/libCryptoki2.so
		/usr/safenet/lunaclient/lib/libcklog2.so
		/usr/safenet/lunaclient/lib/libshim.so
Linux (64-bit)	LibUNIX64	/usr/safenet/lunaclient/lib/libCryptoki2_64.so
		/usr/safenet/lunaclient/lib/libcklog2.so
		/usr/safenet/lunaclient/lib/libshim_64.so
HP-UX (32-bit and 64-bit)	LibHPUX	/opt/safenet/lunaclient/lib/libCryptoki2.sl
		/opt/safenet/lunaclient/lib/libCryptoki2_64.sl
		/opt/safenet/lunaclient//lib/libcklog2.sl
		/opt/safenet/lunaclient/lib/libshim.sl
AIX (32-bit and 64-bit)	LibAIX	/usr/safenet/lunaclient/lib/libCryptoki2.so
		/usr/safenet/lunaclient/lib/libCryptoki2_64.so
		/usr/safenet/lunaclient/lib/libcklog2.so
		/usr/safenet/lunaclient/lib/libshim.so

Included with SafeNet Product Software Development Kit is a sample application – and the source code – to accelerate integration of SafeNet's cryptographic engine into your system.

Note: To reduce development or adaptation time, you may re-distribute the salogin program to customers who use SafeNet Network HSM, in accordance with the terms of the End User License Agreement. However, you may not re-distribute the SafeNet Software Development Kit itself.

Application Programming Interface (API) Overview

The major API provided with SafeNet Product Software Development Kit conforms to RSA Laboratories' Public-Key Cryptography Standards #11 (PKCS #11) v2.20, as described in "PKCS#11 Support" on page 26. A set of API services (called PKCS #11 Extensions) designed by SafeNet, augments the services provided by PKCS#11, as described in "Extensions to PKCS#11" on page 32. The extensions to each API enable optimum use of SafeNet SafeNet hardware for commonly used calls and functions, where the unaugmented API would tend to use software, or to make generic, non-optimized use of available HSMs.

In addition, support is provided for Microsoft's cryptographic APIs (CAPI/CNG) (see "Microsoft Interfaces" on page 370 and Oracle's Java Security API (see "Java Interfaces" on page 346).

The API is a library – a DLL in Windows, a shared object in Solaris, AIX and Linux, a shared library in HP-UX – called Chrystoki. Applications wanting to use token services must connect with Chrystoki.

Table 1: SafeNet libraries by platform

Platform	Key name	Libraries
Windows	LibNT	X:\Program Files\SafeNet\LunaClient\cryptoki.dll
		X:\Program Files\SafeNet\LunaClient\cklog201.dll
		X:\Program Files\SafeNet\LunaClient\shim.dll
		X:\Program Files\SafeNet\LunaClient\LunaCSP\LunaCSP.dll
		C:\WINDOWS\system32\SafeNetKSP.dll
Solaris (32-bit)	LibUNIX	/opt/safenet/lunaclient/lib/libCryptoki2.so
		/opt/safenet/lunaclient/lib/libcklog2.so
		/opt/safenet/lunaclient/lib/libshim.so
Solaris (64-bit)	LibUNIX64	/opt/safenet/lunaclient/lib/libCryptoki2_64.so
		/opt/safenet/lunaclient/lib/libcklog2.so
		/opt/safenet/lunaclient/lib/libshim_64.so
Linux (64-bit)	LibUNIX	/usr/safenet/lunaclient/lib/libCryptoki2.so
		/usr/safenet/lunaclient/lib/libcklog2.so
		/usr/safenet/lunaclient/lib/libshim.so

Platform	Key name	Libraries
Linux (64-bit)	LibUNIX64	/usr/safenet/lunaclient/lib/libCryptoki2_64.so
		/usr/safenet/lunaclient/lib/libcklog2.so
		/usr/safenet/lunaclient/lib/libshim_64.so
HP-UX (32-bit and 64-bit)	LibHPUX	/opt/safenet/lunaclient/lib/libCryptoki2.sl
		/opt/safenet/lunaclient/lib/libCryptoki2_64.sl
		/opt/safenet/lunaclient//lib/libcklog2.sl
		/opt/safenet/lunaclient/lib/libshim.sl
AIX (32-bit and 64-bit)	LibAIX	/usr/safenet/lunaclient/lib/libCryptoki2.so
		/usr/safenet/lunaclient/lib/libCryptoki2_64.so
		/usr/safenet/lunaclient/lib/libcklog2.so
		/usr/safenet/lunaclient/lib/libshim.so

Sample Application

Included with SafeNet Product Software Development Kit is a sample application – and the source code – to accelerate integration of SafeNet's cryptographic engine into your system.

Note: To reduce development or adaptation time, you may re-distribute the salogin program to customers who use SafeNet Network HSM, in accordance with the terms of the End User License Agreement. However, you may not re-distribute the SafeNet Software Development Kit itself.

A Note About RSA Key Attributes 'p' and 'q'

When RSA keys are generated, 'p' and 'q' components are generated which, theoretically, could be of considerably different sizes.

Unwrapping

The SafeNet Network HSM allows RSA private keys to be unwrapped onto the HSM where the lengths of the 'p' and 'q' components are unequal. Because the effective strength of an RSA key pair is determined by the length of the shorter component, choosing 'p' and 'q' to be of equal length provides the maximum strength from the generated key pair. If your application is designed to generate key pairs that will be unwrapped onto the HSM, care should be taken in choosing the lengths of the 'p' and 'q' components such that they differ by no more than 15%.

Generation

Where you are generating RSA private keys within the HSM, the HSM enforces that 'p' and 'q' be equal in size, to the byte level.

A Note About the Shim

The Client install includes a shim library to support PKCS#11 integration with various third-party products. You should have no need for this shim library in your development. If for some reason you determine that you need the shim, Chrystoki supports it.

What Does 'Supported' Mean?

With the exception of some generic items that (for example) might need to be set in Windows when installing CSP, KSP, or Java, we do not include a list of integrations in the main product documentation.

Instead, you can check with the www.safenet-inc.com website for third-party applications that have been integrated and tested with SafeNet HSMs by our Integrations group. That group is constantly testing and updating third-party integrations and publishing notes and instructions to help you integrate our HSMs with your applications.

As a general rule, if a specific version of an application and a specific version of a SafeNet HSM product are mentioned in an Integration document, then those items will definitely work together. A newer version of the SafeNet HSM or its attendant software is most likely to work with the indicated application without problem. We take care, for several generations of a given HSM product, to not break working relationships, though eventually it might happen that very old versions of third-party software and systems can no longer be supported. One thing that can sometimes happen is that we update HSM firmware to include newer algorithms, and to exclude older algorithms or key sizes that no longer meet industry-accepted standards (like NIST, Common Criteria, etc.).

A newer version of a third-party software might, or might not work with SafeNet HSMs that were tested to work with a specific earlier version of the same software. This is because some vendors make changes in their products that require new adaptation or at least new configuration instructions. If this happens to you, SafeNet Customer Support or Sales Engineering is usually happy to work with you to find a solution - both to support you as one of our customers and to have a revised/new integration that can be added to our portfolio.

Check the website or contact SafeNet Customer Support for the latest list of third-party applications that are tested and supported with SafeNet HSMs.

Why Is an Integration Not Listed Here Or On the Website?

In many cases, third-party application vendors see a need to integrate their application with SafeNet products. In those cases, the third-party company performs the integration and testing, and also provides the support for the integrated solution to their customers (including you). For integrations not listed by SafeNet, please contact the application vendor for current information.

Similarly some value-added resellers and custom/third-party integrators or consultants might have performed specific integrations of SafeNet HSMs for the benefit of their specific customers. If you have purchased services or product from such a supplier, you will need to contact them for support of such integrations.

Third-party-tested integrations are not listed here or on the SafeNet website library of integration documents because we have not verified them in our own labs. If you call SafeNet Support regarding use of our product with an application that we have not integrated, you will be asked to contact the third party that performed the integration.

Frequently Asked Questions

This section provides additional information by answering questions that are frequently asked by our customers.

How can we use a SafeNet HSM with a Key Manager?

A SafeNet HSM could be a Certificate Authority (CA) within your organization, and would operate in parallel with a Key Manager. It is normally the Key Manager that requests service from a CA, and not the other way around. For example, the Key Manager might generate an RSA key pair for an endpoint to use for authentication. The KM would then go to its associated CA and request a certificate for the public key.

The other typical use case for a KM looking to a CA for service is for confirming certificate validity, either through CRLs or OCSP.

In general, the HSM keeps keys safe within its confines, and exports only metadata about the contained objects. The metadata allows the KM or an integrated application to refer to the keys and objects within the HSM, when invoking cryptographic operations by the HSM, but not to touch the actual keys or objects themselves.

A CA's private key(s) are extremely valuable and often are used only by a CA application operating on a stand-alone server or one on a very minimally-connected subnet. Backup is normally done to a small form factor HSM that can then be locked away in a safe.

We need to encrypt PANs on MS SQL Server 2008 (Extensible Key Management). We have a problem with the encrypted PAN, as the length is greater than the original PAN (16 digits).

The issue is a common one and it arises because the CBC padding scheme requires an extra padding block (8 bytes), with all bytes having the hex value 8, to be appended if the length of the original plaintext is a multiple of the cipher's block length. Another format issue often comes up as well since encrypted data does not generally represent well as decimal digits.

We suggest one of two options:

1. You can set up a shadow table to hold the encrypted PANs. The shadow table schema can then be set up for a sufficient number of hex numerals to hold the padded data or just make that field a binary blob. This takes some coding on your part, and the plaintext PANs would be retrieved into a dynamic view, rather than back into the "real" table, to protect their confidentiality. You should do this only if there is a hard requirement to use SafeNet HSM, such as certification.
2. Alternatively, you can switch to DataSecure. It has tokenization support and is, in general, designed for DB security.

"Makecert" fails when using SafeNet Network HSM with MS Authenticode, because the MD5 algorithm is not available when the HSM is in FIPS mode. Error: CryptHashPublicKeyInfo failed => 0x80090005 (-2146893819) Failed, and FINIDigest_Init ***CKR_MECHANISM_INVALID***(296ms) {}

The certificate always has an MD5 hash in it. Configure LunaCSP algorithm registration such that MD5 hashing is performed in software. For example:

register.exe /algorithms

We are developing our application(s) in C#, and we want to integrate with SafeNet HSMs

If you want to integrate your C# application with SafeNet HSM 6.x using PKCS#11 calls, rather than using Microsoft CAPI or CNG, then you might consider using "ncryptoki". At the time this note is being written, we have not created anything formal, but we have worked with some customers who are successfully using "ncryptoki" for that purpose.

Keep an eye on the Safenet C3 website, or ask your SafeNet technical representatives if anything new has been added. Or, you could engage SafeNet Professional Services for formal assistance with your project.

We intend to use PKCS#11 data objects - is this supported in the API for your HSMs?

Yes, it's a basic requirement.
If you have concerns, you might wish to verify if SafeNet HSMs' (and our API's) handling of data objects are conducive to the operation of your intended application(s). SafeNet API generally places no restrictions on whether data objects can be private or not. We understand that, in the past, some competitors' modules might have allowed only public data objects, if that was the basis of your question.

However, one concern that might arise is Java.
Java offers no support for data objects, and so we do not support them with the LunaProvider. Unexpected results can occur with SafeNet JCA if a data object is present in a partition. This might be the case if you attempt to use an application that uses the CSP, and then the JSP accesses the same partition. CSP inherently creates a data object for its own purposes.

Therefore, keep CSP and JSP clients tied to separate partitions. Generally do not allow JSP to connect to a partition that contains a data object, regardless of the source - Java (and therefore JSP) doesn't know what to do with it.

If your application scenario really does demand the use of both the Microsoft Cryptographic Provider and Java against a common partition, then consider upgrading/updating to Microsoft CNG and use our KSP, which does not inherently create a data object, and so would not cause conflict of that sort.

In our application, both for PKCS#11 and for the JCA/JCE SafeNet Provider, we need to use CKM_SHAxxx_RSA_PKCS mechanism for Signing. Does Hashing occur at the Client or in the HSM?

CKM_SHAxxx_RSA_PKCS is a PKCS#11 mechanism, not a Java method.

For PKCS#11 the digest operation is done within the HSM if that mechanism is called.
For Java, digests are done in software.

We were using another vendor's HSM - or are evaluating HSM products - to host an online sub- or issuing CA with MSCA. With the other vendor we must check "Allow administrator interaction when the private key is accessed by the CA" in the "Configure Cryptography" setup dialog. SafeNet HSMs seem to work regardless of whether that selection is checked or not.

So, for that other vendor's product, you need to enter the additional credentials every time you need to issue a certificate? That seems a bit restrictive.

"Allow administrator interaction..." actually means "Allow administrator interaction if the underlying KSP requires it".

The Windows operating system passes a Windows handle that the KSP can use to render any GUI designed by a vendor (SafeNet or some other vendor).

Somewhere in the process a KSP reports that it can (or cannot) interact with the GUI so the application will (or will not) request GUI interaction; that is, pass a window handle to the KSP.

So, the <competitor product> KSP expects a window handle - implying hands-on action by an administrator, each time - whereas SafeNetKsp ignores the handle (if one was provided).

SafeNet's KSP was designed to register partitions ahead of time. SafeNet HSMs can be Activated, which caches the administrative and enabling credentials, such that only the partition challenge (text string) is needed, which can be passed by your application without need for GUI interaction. Furthermore, SafeNet Network HSM can "AutoActivate" partitions, which allows cached ("Activated") partition credentials to be retained through power interruptions as long as 2 hours in duration.

For SafeNet HSMs, as long as the user is registered in the KSP utility, and the partition is activated, the "Allow administrator interaction..." check box (checked or not checked) does not impose any additional, ongoing, authentication requirements -- no additional prompts for credentials from the GUI. After initial setup and Activation, the SafeNet HSM knows what to do, and doesn't need to pester you.

For root CAs, on the other hand, you always have the option of not activating the partition, so PED interaction would always be required to ensure close supervision for each use of the private key.

2
PKCS#11 Support

This chapter describes the PKCS#11 support provided by the SafeNet SDK. It contains the following topics:

PKCS#11 Compliance

This section shows the compliance of SafeNet Software Development Kit HSM products to the PKCS#11 standard, with reference to particular versions of the standard. The text of the standard is not reproduced here.

Supported PKCS#11 Services

The table below identifies which PKCS#11 services this version of SafeNet Software Development Kit supports. The table following lists other features of PKCS#11 and identifies the compliance of this version of the SafeNet Software Development Kit to these features.

Table 1: PKCS#11 function support

Category	Function	Supported SafeNet ver 2.20
General purpose functions	C_Initialize	Yes
	C_Finalize	Yes
	C_GetInfo	Yes
	C_GetFunctionList	Yes
	C_Terminate	Yes

Category	Function	Supported SafeNet ver 2.20
Slot and token management functions	C_GetSlotList	Yes
	C_GetSlotInfo	Yes
	C_GetTokenInfo	Yes
	C_WaitForSlotEvent	No
	C_GetMechanismList	Yes
	C_GetMechanismInfo	Yes
	C_InitToken	Yes
	C_InitPIN	Yes
	C_SetPIN	Yes
Session management functions	C_OpenSession	Yes
	C_CloseSession	Yes
	C_CloseAllSessions	Yes
	C_GetSessionInfo	Yes
	C_GetOperationState	Yes
	C_SetOperationState	Yes
	C_Login	Yes
	C_Logout	Yes
Object management functions	C_CreateObject	Yes
	C_CopyObject	Yes
	C_DestroyObject	Yes
	C_GetObjectSize	Yes
	C_GetAttributeValue	Yes
	C_SetAttributeValue	Yes
	C_FindObjectsInit	Yes
	C_FindObjects	Yes
	C_FindObjectsFinal	Yes

Category	Function	Supported SafeNet ver 2.20
Encryption functions	C_EncryptInit	Yes
	C_Encrypt	Yes
	C_EncryptUpdate	Yes
	C_EncryptFinal	Yes
Decryption functions	C_DecryptInit	Yes
	C_Decrypt	Yes
	C_DecryptUpdate	Yes
	C_DecryptFinal	Yes
Message digesting functions	C_DigestInit	Yes
	C_Digest	Yes
	C_DigestUpdate	Yes
	C_DigestKey	Yes
	C_DigestFinal	Yes
Signing and MACing functions	C_SignInit	Yes
	C_Sign	Yes
	C_SignUpdate	Yes
	C_SignFinal	Yes
	C_SignRecoverInit	No
	C_SignRecover	No
Functions for verifying signatures and MACs	C_VerifyInit	Yes
	C_Verify	Yes
	C_VerifyUpdate	Yes
	C_VerifyFinal	Yes
	C_VerifyRecoverInit	No
	C_VerifyRecover	No

Category	Function	Supported SafeNet ver 2.20
Dual-purpose cryptographic functions	C_DigestEncryptUpdate	No
	C_DecryptDigestUpdate	No
	C_SignEncryptUpdate	No
	C_DecryptVerifyUpdate	No
Key management functions	C_GenerateKey	Yes
	C_GenerateKeyPair	Yes
	C_WrapKey	Yes
	C_UnwrapKey*	Yes
	C_DeriveKey	Yes
Random number generation functions	C_SeedRandom	Yes
	C_GenerateRandom	Yes
Parallel function management functions	C_GetFunctionStatus	No
	C_CancelFunction	No
Callback function		No

*C_UnwrapKey has support for the CKA_Unwrap_Template object. All mechanisms that perform the unwrap function support an unwrap template. Nested templates are not supported.
The ability to affect key attributes is controlled by partition policy *11: Allow changing key attributes*.

Note: UNWRAP TEMPLATE attribute - Your Backup HSM must have firmware version 6.24.0 or newer, as well. If a key is cloned or backed-up to an HSM with older firmware, the newer attribute will not be recognized and will be dropped from the object. So when the object is restored, it will no longer have a CKA_UNWRAP_TEMPLATE attribute.

Table 2: PKCS#11 feature support

Feature	Supported?
Exclusive sessions	Yes
Parallel sessions	No

Additional Functions

Please note that certain additional functions have been implemented by SafeNet as extensions to the standard. These include aspects of object cloning, and are described in detail in "SafeNet Extensions to PKCS#11" on page 32

Using the PKCS#11 Sample

The SafeNet SDK includes a simple "C" language cross platform source example, **p11Sample**, that demonstrates the following:

- how to dynamically load the SafeNet cryptoki library
- how to obtain the function pointers to the exported PKCS11 standard functions and the SafeNet extension functions.

The sample demonstrates how to invoke some, but not all of the API functions.

The SfntLibPath Environment Variable

The sample depends on an environment variable created and exported prior to execution. This variable specifies the location of **cryptoki.dll** (Windows) or **libCryptoki2.so** on Linux/UNIX. The variable is called **SfntLibPath**. You are free to provide your own means for locating the library.

What p11Sample Does

The p11Sample program performs the following actions:

1. The sample first attempts to load the dynamic library in the function called **LoadP11Functions**. This calls **LoadLibrary** (Windows) or **dlopen** (Linux/UNIX).
2. The function then attempts to get a function pointer to the PKCS11 API **C_GetFunctionList** using **GetProcAddress** (Windows) or **dlsym** (Linux/UNIX).
3. Once the function pointer is obtained, use the API to obtain a pointer called **P11Functions** that points to the static CK_FUNCTION_LIST structure in the library. This structure holds pointers to all the other PKCS11 API functions supported by the library.

   At this point, if successful, PKCS11 APIs may be invoked like the following:

   ```
   P11Functions->C_Initialize(...);
   P11Functions->C_GetSlotList(...);
   P11Functions->C_OpenSession(...);
   P11Functions->C_Login(...);
   P11Functions->C_GenerateKey(...);
   P11Functions->C_Encrypt(...);
   :
   :
   etc
   ```

4. The sample next attempts to get a function pointer to the SafeNet extension **API CA_GetFunctionList** using **GetProcAddress** (Windows) or **dlsym** (Linux/UNIX).
5. Once the function pointer is obtained, use the API to obtain a pointer called **SfntFunctions** that points to the static CK_SFNT_CA_FUNCTION_LIST structure in the library. This structure holds pointers to some but not all of the other SafeNet extension API functions supported by the library.
6. At this point, if successful, SafeNet extension APIs may be invoked like the following:

   ```
   SfntFunctions->CA_GetHAState(...);
   :
   :
   etc.
   ```

7. Three sample makefiles are provided:
   - one for 32-bit Windows,
   - one for 32-bit Linux, and
   - one for 64-bit AIX.

   You can easily port to another platform with minor changes.
8. To build:

Windows	nmake -f Makefile.win32
Linux	make -f Makefile.linux.32
Aix	make -f Makefile.aix.64

Note: Please note that this simple example loads the cryptoki library directly. If your application requires integration with cklog or ckshim, you will need to load the required library (see SDK General for naming on your platform) in lieu of cryptoki. cklog and ckshim will then use the Chrystoki configuration file to locate and load cryptoki. You also have the option of locating the cryptoki library by parsing the Chrystoki2 section of the Chrystoki config file. If you do this, then the initial library (cryptoki, cklog, or ckshim) can be changed by simply updating the configuration file.

3
Extensions to PKCS#11

This chapter describes the SafeNet extensions to the PKCS#11 standard. It contains the following topics:

SafeNet Extensions to PKCS#11

This section presents a set of extensions which have been added to PKCS#11 by SafeNet. They cover several areas of cryptographic protocol/standard support and system information, as follows:

- FIPS 140-2 validation including a secure PIN and data port
- Key CloningTM support
- secret sharing activation support
- support for sharing login state across applications
- support for an alternate login scheme, referred to as "Indirect Login"
- support for manipulating token state vectors
- support for synchronization of multiple SafeNet XL tokens for enhanced cryptographic acceleration.

Other APIs

These commands and functions can also be used as extensions to other Application Programming Interfaces (for example, OpenSSL).

Summary of New Functions

The several functions defined in this extension to PKCS#11 are introduced in the following table. These functions are described in more detail in later sections of this document.

Table 1: Summary of new Cryptoki Functions

Category	Function	Description
Key cloning	CA_SetCloningDomain	Sets the domain string used during token initialization.
	CA_ClonePrivateKey	Permits the secure transfer a private key (RSA) between a source token and a target token.
	CA_ GenerateTokenKeys	Generate the private keys used for secure key cloning operations.
	CA_ GetTokenCertificateInfo	Obtain the cloning certificate.
	CA_SetTokenCertificate Signature	Sign the cloning certificate with the private keys generated for key cloning operations
Secret Sharing Activation (commonly referred to as MofN)	CA_SetMofN	Sets the security policy for the token to use the secret sharing feature.
	CA_GenerateMofN	Generates the secret informa-tion on a token.
	CA_ActivateMofN	Activates a token that has the secret sharing feature enabled.
	CA_ GenerateCloneableMofN	Creates a clonable secret-splitting vector on a token.
	CA_CloneMofN	Copy a clonable secret-splitting vector from one token to another.
	CA_DuplicateMofN	Creates duplicates (copies) of all MofN secret splits.
	CA_ModifyMofN	Modifies the secret-splitting vector on a token.
	CA_GetMofNStatus	Retrieves the MofN structure of the specified token.
Share login state across applications	CA_SetApplicationID	Sets the application's identifier.
	CA_OpenApplicationID	Activates an application identifier, independent of any open sessions.
	CA_CloseApplicationID	Deactivates an application identifier.
Indirect Login	CA_InitIndirectPIN	Initializes a user PIN so that it may be used normally or indirectly.
	CA_IndirectLogin	Performs an indirect login operation.

Category	Function	Description
Token State Vector Manipulation	CA_GetFPV	Retrieves the token's Fixed Policy Vector (FPV).
	CA_GetTPV	Retrieves the token's Token Policy Vector (TPV).
	CA_GetExtendedTPV	Retrieves the token's TPV and extended TPV.
	CA_SetTPV	Sets the token's TPV.
	CA_SetExtendedTPV	Sets the token's TPV and extended TPV.
XL	CA_ GetNumberOfSSLSlots	Determine the number of accelerator slots (distinct from authentication slots, when SafeNet XL is also used with SafeNet CA *3* .)
	CA_ SSLSynchronizeObjects	Designates a master slot in an XL installation (you pass the slot number with the call) then clones the content of the master slot to all other accelerator slots
Derive and Wrap	CA_DeriveKeyAndWrap	This is an optimization of C_DeriveKey with C_Wrap, merging the two functions into one (the in and out constraints are the same as for the individual functions). A further optimization is applied when mechanism CKM_ECDH1_DERIVE is used with CA_ DeriveKeyAndWrap.

Cryptoki Version Supported

The current release of SafeNet SafeNet Toolkit provides the Chrystoki library supporting version 2.20 of the Cryptoki standard.

HSM Configuration Settings

SafeNet HSMs implement configuration settings that can be used to modify the behavior of the HSM, or can be read to determine how the HSM will behave. There are multiple settings that may be manipulated. Other than the "allow non-FIPS algorithms", most customers have no need to either query or change HSM settings. If you believe that your application needs more control over the HSM, please contact SafeNet for guidance.

SafeNet Network HSM-Specific Commands

SafeNet Network HSM, both the HSM Server and the client, use PKCS#11 and the SafeNet Extensions, with some exceptions that differ from other SafeNet products. This SDK document is meant to support all SafeNet products that use PKCS#11 and the other supported interfaces, in addition to SafeNet Network HSM.

Commands Not Available Through Libraries

Several commands, both standard PKCS#11 commands and our Extensions are not enabled in the Client, because their functions are addressed on SafeNet Network HSM via the lunash interface. These are:

- C_InitToken
- C_SetPin
- CA_ResetPin
- CA_SetCloningDomain
- all of the CCM commands
- CA_ClonePrivateKey
- C_GetOperationState
- C_SetOperationState

Configuration Settings

Other SafeNet tokens implement configuration settings that can be used to modify the behavior of the token, or can be read to determine how the token will behave.

In SafeNet Network HSM, this configuration and modification of HSM and behavior is controlled in lunash via HSM Policies, using the following commands:

- "hsm showpolicies" on page 1
- "hsm changepolicy" on page 1

Control of HSM Partition behavior is accomplished through the HSM Partition Policies, using the following lunash commands:

- "partition showpolicies" on page 1
- "partition changepolicy" on page 1

Secure PIN Port Authentication

Generally, an application collects an authentication code or PIN from a user and/or other source controlled by the host computer. With Gemalto's FIPS 140-2 level 3-validated products (such as SafeNet Network HSM), the PIN must come from a device connected to the secure port of the physical interface (or connected via a secure Remote PED protocol connection). The SafeNet PED (PIN Entry Device) is used for secure entry of PINs.

A bit setting in the device's capabilities settings determines whether the HSM requires that PINs be entered through the secure port. If the appropriate configuration bit is set, PINs must be entered through the secure port.

If the device`s configuration bit is off, the application must provide the PIN through the existing mechanism. Through setting the PIN parameters, the application tells the token where to look for PINs. A similar programming approach applies to define the key cloning domain identifier.

Applications wanting PINs to be collected via the secure port must pass a NULL pointer for the pPin parameter and a value of zero for the ulPinLen parameter in function calls with PIN parameters. This restriction applies everywhere PINs are used. The following functions are affected:

- C_InitToken
- C_InitIndirectToken
- C_InitPIN
- C_SetPIN
- CA_InitIndirectPIN

- C_Login
- CA_IndirectLogin

When domains are generated/collected through the secure port during a C_InitToken call, the application must pass a NULL pointer for the pbDomainString parameter and a value of zero for the ulDomainStringLen parameter in the CA_SetCloningDomain function.

Shared Login State and Application IDs

The PKCS#11 specification states that sessions within an application share a login state. An application is defined as a single address space and all threads that execute within it. Thus, if process A spawns multiple threads, and all of those threads open sessions on token #1, then all of those sessions share a login state. When one is logged in, they all are, and when one is logged out, they all are. However, if process B also has sessions open on token #1, they are independent from the sessions of process A. The login state of process B sessions is irrelevant to process A sessions (except where they conflict, such as process A logging in as USER when process B is already logged in as SO).

The Chrystoki library provides additional functionality that allows separate applications to share a login state. Within Chrystoki, each application has an application ID. An application ID is a 64-bit integer, normally specified in two 32-bit parts. A default application ID for the application is generated automatically by the Chrystoki library, when the application invokes **C_Initialize**. The default value is based upon the process ID of the application, so different applications will always have different application IDs.

Each session also has an application ID associated with it. This is the application ID of the application that created the session. Within Chrystoki and SafeNet tokens, login states are shared by sessions which have identical application IDs. Since there is usually a one-to-one mapping between applications and application IDs, this means that login states are normally shared between sessions within an application but not between applications. In order to allow separate Chrystoki applications to share session state, Chrystoki provides functionality that allows applications to alter their application IDs.

Why Share Session State Between Applications?

For many applications, the functionality described here serves no purpose. If an application consists of a single process that exists perpetually, unshared session states are sufficient. If the application supports multiple processes, but the application designer wants each process to validate (login) separately, unshared session states are sufficient.

However, if

- the application consists of multiple processes each with its own sessions and
- the application designer wants to require only one login action by the user and
- the system uses SafeNet CA3 tokens (where PINs cannot be cached and used multiple times by the application),

then, it is necessary to share login state between processes.

The SafeNet CA3 token provides FIPS 140-1 level 3 security through use of a separate port for password entry (with the SafeNet CA3 token, PINs take the form of special data keys). Use of these keys prevents an application from caching a password and using it to log in with multiple sessions. If you want to log in once only, and you use separate processes, you must somehow share login state between processes.

[UPDATE: Applies to newer SafeNet HSMs as well, in some integrations, for ease of use particularly against PED-authenticated HSMs.]

Login State Sharing Overview

The simplest form the extra Chrystoki functionality takes is the **CA_SetApplicationID** function. This function should be invoked after **C_Initialize** is invoked, but before any sessions are opened. Two separate applications can use this function to set their application IDs to the same value, and thus allow them to share login states between their sessions.

Alternately, the **AppIdMajor** and **AppIdMinor** fields in the **Misc** section of the Chrystoki configuration file can be set. This causes the default application ID of all applications to be set to the value given in the configuration file, rather than being generated from the application's process ID. This means that unless applications use the **CA_SetApplicationID** function, all applications on a host system will share login state between their sessions.

Example

A sample configuration file (**crystoki.ini** for Windows) using explicit application IDs is duplicated here:

```
[Chrystoki2]
LibNT=D:\Program Files\SafeNet\LunaClient\cryptoki.dl
[Luna]
DefaultTimeOut=500000
PEDTimeout1=100000
PEDTimeout2=200000
[CardReader]
RemoteCommand=1
[Misc]
AppIdMajor=2
AppIdMinor=4
```

One effect that can still cause problems is that when all sessions of a particular application ID are closed, that application ID reverts to a dormant state. When another session for that application ID is created, the application ID is recreated, but always in the logged-out state, regardless of the state it was in when it went dormant.

For example, consider an application where a parent process sets its application ID, opens a session, logs the session in, then closes the session and terminates. Several child pro-cesses then set their application IDs, open sessions and try to use them. However, since the application ID went dormant when the parent process closed its session, the child processes find their sessions logged out. The logged-in state of the parent process' session was lost when it closed its session.

The CA_OpenApplicationID function can be used to ensure that the login state of an application ID is maintained, even when no sessions exist which belong to that application ID. When CA_OpenApplicationID is invoked, the application ID is tagged so that it never goes dormant, even if no open ses-sions exist.

Login State Sharing Functions

Use the following functions to configure and manage login state sharing:

CA_SetApplicationID

```
CK_RV CK_ENTRY CA_SetApplicationID(
CK_ULONG ulHigh,
CK_ULONG ulLow
);
```

The **CA_SetApplicationID** function allows an application to set its own application ID, rather than letting the application ID be generated automatically from the application's process ID. **CA_SetApplicationID** should be invoked

after **C_Initialize** but before any session manipulation functions are invoked. If **CA_SetApplicationID** is invoked after sessions have been opened, results will be unpredictable.

CA_SetApplicationID always returns CKR_OK.

CA_OpenApplicationID

```
CK_RV CK_ENTRY CA_OpenApplicationID(
CK_SLOT_ID slotID,
CK_ULONG ulHigh,
CK_ULONG ulLow
);
```

The **CA_OpenApplicationID** function forces a given application ID on a given token to remain active, even when all sessions belonging to the application ID have been closed. Normally an application ID on a token goes dormant when the last session that belongs to the application ID is closed. When an application ID goes dormant login state is lost, so when a new session is created within the application ID, it starts in the logged-out state. However, if **CA_OpenApplicationID** is used the application ID is prevented from going dormant, so login state is main-tained even when all sessions for an application ID are closed.

CA_OpenApplicationID can return CKR_SLOT_ID_INVALID or CKR_TOKEN_NOT_PRESENT.

CA_CloseApplicationID

```
CK_RV CK_ENTRY CA_CloseApplicationID(
CK_SLOT_ID slotID,
CK_ULONG ulHigh,
CK_ULONG ulLow
);
```

The **CA_CloseApplicationID** function removes the property of an application ID that prevents it from going dormant. **CA_CloseApplicationID** also closes any open sessions owned by the given application ID. Thus, when **CA_CloseApplicationID** returns, all open sessions owned by the given application ID have been closed and the application ID has gone dormant.

CA_CloseApplicationID can return CKR_SLOT_ID_INVALID or CKR_TOKEN_NOT_PRESENT.

Application ID Examples

The following code fragments show how two separate applications might share a single application ID:

```
app 1:          app 2:
C_Initialize()
CA_SetApplicationID(3,4)
C_OpenSession()
C_Login()
                C_Initialize()
                CA_SetApplicationID(3,4)
                C_OpenSession()
                C_GetSessionInfo()
                // Session info shows session
                // already logged in.
                <perform work, no login
                necessary>

C_Logout()
                C_GetSessionInfo()
                // Session info shows session
```



```
                // logged out.

C_CloseSession()
                C_CloseSession()
C_Finalize()
                C_Finalize()
```

The following code fragments show how one process might login for others:

Setup app:

```
C_Initialize()
CA_SetApplicationID(7,9)
CA_OpenApplicationID(slot,7,9)
C_OpenSession(slot)
C_Login()
C_CloseSession()
```

Spawn many child applications:

```
C_Finalize()
```

Terminate each child app:

```
                C_Initialize()
                CA_SetApplicationID(7,9)
                C_OpenSession(slot)
                <perform work, no login necessary>
```

Takedown app:

Terminate child applications:

```
                C_CloseSession()
                C_Finalize()
C_Initialize()
CA_CloseApplicationID(slot,7,9)
C_Finalize()
```

High Availability Indirect Login Functions

> **Note:** In order to implement High Availability Recovery, the primary and secondary tokens must exist on separate systems.

The following enhancements securely extend the indirect login capability to SafeNet CA3 tokens. SafeNet CA3 tokens to store sensitive information (encrypted) in flash memory, and must therefore be protected against attack by a man-in-the-middle who physically attacks the target token to expose the contents of flash memory, and employs that information against intercepted (or spuriously-generated) message traffic.

The SafeNet CA3 to SafeNet CA3 indirect login protocol also supports old-style MofN authentication between tokens that share an MofN secret.

Initialization functions

Initialization of tokens in a high-availability environment involves three steps:

1. The generation of an RSA login key pair (the public key of the pair may be discarded),

2. Cloning of the private key member to the User (and optionally to the SO) spaces of all tokens within that environment and,
3. Calling the **CA_HAInit** function on all tokens within that environment, in the context of the session owned by the User or SO.

The first two steps are performed using ordinary key generate and cloning Cryptoki function calls. The **CA_HAInit** function is implemented as follows:

CA_HAInit()

```
CK_RV CK_ENTRY CA_HAInit(
CK_SESSION_HANDLE hSession, // Logged-in session of user
// who owns the Login key pair
CK_OBJECT_HANDLE hLoginPrivateKey // Handle to Login private key
);
```

Recovery Functions

The HA recovery mechanism requires the following commands and interface functions:

CA_HAGetMasterPublic()

Called on the primary token, CA_HAGetMasterPublic() retrieves the primary token's TWC (Token Wrapping Certificate) and returns it as a blob (octet string and length). The format of this function is as follows:

```
CK_RV CK_ENTRY CA_HAGetMasterPublic(
CK_SLOT_ID slotId, // Slot number of the primary
// token
CK_BYTE_PTR pCertificate, // pointer to buffer to hold
//TWC certificate
CK_ULONG_PTR pulCertificateLen // pointer to value to hold
//TWC certificate length
);
```

CA_HAGetLoginChallenge()

Called on the secondary token, **CA_HAGetLoginChallenge()** accepts the TWC blob and returns the secondary token's login challenge blob. The format of this command is as follows:

```
CK_RV CK_ENTRY CA_HAGetLoginChallenge(
CK_SESSION_HANDLE hSession, // Public session
CK_USER_TYPE userType, // User type - SO or USER
CK_BYTE_PTR pCertificate, // TWC certificate retrieved
// from primary
CK_ULONG ulCertificateLen, // TWC certificate length
CK_BYTE_PTR pChallengeBlob, // pointer to buffer to hold
// challenge blob
CK_ULONG_PTR pulChallengeBlobLen // pointer to value to hold
// challenge blob length
);
```

CA_HAAnswerLoginChallenge()

Called on the primary token, **CA_HAAnswerLoginChallenge()** accepts the login challenge blob and returns the encrypted SO or User PIN, as appropriate.

```
CK_RV CK_ENTRY CA_HAAnswerLoginChallenge(
CK_SESSION_HANDLE hSession, // Session of the Login Private
```



```
// key owner
CK_OBJECT_HANDLE hLoginPrivateKey, // object handle to login key
CK_BYTE_PTR pChallengeBlob, // pointer to buffer containing
// challenge blob
CK_ULONG ulChallengeBlobLen, // length of challenge blob
CK_BYTE_PTR pEncryptedPin, // pointer to buffer holding
// encrypted PIN
CK_ULONG_PTR pulEncryptedPinLen // pointer to value holding
// encrypted PIN length
);
```

CA_HALogin()

Called on the secondary token, **CA_HALogin()** accepts the encrypted PIN and logs the secondary token in. If the second-ary token requires MofN authentication, an MofN challenge blob is returned. If no MofN authentication is required, a zero-length blob is returned. The format of this function is as follows:

```
CK_RV CK_ENTRY CA_HALogin(
CK_SESSION_HANDLE hSession, // Same public session opened
// in CA_HAGetLoginChallenge,
//above
CK_BYTE_PTR pEncryptedPin, // pointer to buffer holding
// encrypted PIN
CK_ULONG ulEncryptedPinLen,  //  length of encrypted PIN
CK_BYTE_PTR pMofNBlob, // pointer to buffer to hold
// MofN blob
CK_ULONG_PTR pulMofNBlobLen // pointer to value to hold the
// length of MofN blob
);
```

If the call is successful, then the session now becomes a pri-vate session owned by the User or SO (as appropriate).

CA_AnswerMofNChallenge()

Called on the primary token, **CA_AnswerMofNChallenge()** accepts the MofN challenge blob and returns the primary token's masked MofN secret. The format of this function is as follows:

```
CK_RV CK_ENTRY CA_HAAnswerMofNChallenge(
CK_SESSION_HANDLE hSession, // Private session
CK_BYTE_PTR pMofNBlob, // passed in MofN blob
CK_ULONG ulMofNBlobLen, // length of MofN blob
CK_BYTE_PTR pMofNSecretBlob, // pointer to buffer to hold
// MofN secret blob
CK_ULONG_PTR pulMofNSecretBlobLen//pointer to value that holds
// the MofN secret blob len
);
```

CA_HAActivateMofN()

Called on the secondary token, **CA_HAActivateMofN()** accepts the masked MofN secret and performs MofN authentication. The resulting MofN secret is checked against the CRC stored in the MofN PARAM structure.

```
CK_RV CK_ENTRY CA_HAActivateMofN(
CK_SESSION_HANDLE hSession, // The now-private session from
// successful CA_HALogin call
CK_BYTE_PTR pMofNSecretBlob, // pointer to MofN secret
// blob that is passed in
CK_ULONG ulMofNSecretBlobLen // length of MofN secret blob
);
```


It is expected that the recovery functions will be executed in the proper sequence and as part of an atomic operation. Nonetheless, the recovery operation may be restarted at any time due to an error. Restarting the recovery operation resets the state condition of the secondary token, and any data that has been stored or generated on the token is discarded.

Login Key Attributes

The login keys must possess the following attributes to function properly in a HA recovery scenario:

```
// Object
CKA_CLASS = CKO_PRIVATE_KEY,
// StorageClass
CKA_TOKEN = True,
CKA_PRIVATE = True,
CKA_MODIFIABLE = False,
// Key
CKA_KEY_TYPE = CKK_RSA,
CKA_DERIVE = False,
CKA_LOCAL = True,
// Private
CKA_SENSITIVE = True,
CKA_DECRYPT = False,
CKA_SIGN = False,
CKA_SIGN_RECOVER = False,
CKA_UNWRAP = False,
CKA_EXTRACTABLE = False
```

Control of HA Functionality

Refer to for the mechanisms by which the SO can control availability of the HA functionality.

MofN Secret Sharing

In previous SafeNet HSM releases, this page described library and firmware aspects of MofN secret sharing.

Current implementation (since HSM firmware 5) no longer implements MofN via the HSM.

Instead, MofN is entirely mediated via SafeNet PED 2.4 and later. The HSM is unaware of secret sharing. Multi-person access control for any of the authentication secrets (SO, User, Cloning domains, Remote PED Vector, Secure Recovery Vector) is a PED function, and the HSM sees only the fully reconstituted MofN secrets as they are presented to it by the PED.

Green PED Keys are no longer used.

This implementation is both cleaner and more flexible than the legacy implementation.If you have used, or are still using legacy SafeNet HSMs, be aware that the legacy implementation of MofN split-secret, multi-person access control is not compatible with the modern implementation. For migration instructions, contact SafeNet Technical Support -- e-mail: support@safenet-inc.com or phone 800-545-6608 (+1 410-931-7520 International)

Key Export Features

The SafeNet Key Export HSM provides the feature(s) detailed in this section.

RSA Key Component Wrapping

The RSA Key Component Wrapping is a feature that allows an application to wrap any subset of attributes from an RSA private key with 3-DES. Access to the feature is through the PKCS #11 function C_WrapKey with the CKM_DES3_ECB mechanism. The wrapping key must be a CKK_DES2 or CKK_DES3 key with its CKA_WRAP attribute set to TRUE. The key to wrap must be an RSA private key with CKA_EXTRACTABLE set to TRUE and the FPV must have FPV_WRAPPING_TOKEN turned on.

The details of the wrapping format are specified with a format descriptor. The format descriptor is provided as the mechanism parameter to the CKM_DES3_ECB mechanism. This descriptor consists of a 32-bit format version, followed by a set of field element descriptors. Each field element descriptor consists of a 32-bit Field Type Identifier and optionally some additional data. The SafeNet firmware parses the set of field element descriptors and builds the custom layout of the RSA private key in an internal buffer. Once all field element descriptors are processed, the buffer is wrapped with 3-DES and passed out to the calling application. It is the responsibility of the calling application to ensure that the buffer is a multiple of 8 bytes.

The format descriptor version (the first 32-bit value in the format data) must always be set to zero.

The set of supported field element descriptor constants is as follows:

- #define KM_APPEND_STRING 0x00000000
- #define KM_APPEND_ATTRIBUTE 0x00000001
- #define KM_APPEND_REVERSED_ATTRIBUTE 0x00000002
- #define KM_APPEND_RFC1423_PADDING 0x00000010
- #define KM_APPEND_ZERO_PADDING 0x00000011
- #define KM_APPEND_ZERO_WORD_PADDING 0x00000012
- #define KM_APPEND_INV_XOR_CHECKSUM 0x00000020
- #define KM_DEFINE_IV_FOR_CBC 0x00000030

The meanings of the field element descriptors is as follows:

Field element descriptor	Description
KM_APPEND_STRING	Appends an arbitrary string of bytes to the custom layout buffer. The field type identifier is followed by a 32-bit length field defining the number of bytes to append. The length field is followed by the bytes to append. There is no restriction of the length of data that may be appended, as long as the total buffer length does not exceed 3072 bytes.
KM_APPEND_ATTRIBUTE	Appends an RSA private key component into the buffer in big endian representation. The field type identifier is followed by a 32-bit CK_ATTRIBUTE_TYPE value set to one of the following: CKA_PRIVATE_EXPONENT, CKA_PRIME_1, CKA_PRIME_2, CKA_EXPONENT_1, CKA_EXPONENT_2, or CKA_COEFFICIENT.. The key component is padded with leading zeros such that the length is equal to the modulus length in the case of the private exponent, or equal to half of the modulus length in the case of the other 5 components.
KM_APPEND_REVERSED_	Appends an RSA private key component into the buffer in little endian representation. The field type identifier is followed by a 32-bit CK_ATTRIBUTE_TYPE value set to one of the

Field element descriptor	Description
ATTRIBUTE	following: CKA_PRIVATE_EXPONENT, CKA_PRIME_1, CKA_PRIME_2, CKA_EXPONENT_1, CKA_EXPONENT_2, or CKA_COEFFICIENT. The key component is padded with trailing zeros such that the length is equal to the modulus length in the case of the private exponent, or equal to half of the modulus length in the case of the other 5 components.
KM_APPEND_RFC1423_PADDING	Applies RFC 1423 padding to the buffer (appends 1 to 8 bytes with values equal to the number of bytes, such that the total buffer length becomes a multiple of 8). This would typically be the last formatting element in a set, but this is not enforced.
KM_APPEND_ZERO_PADDING	Applies Zero padding to the buffer (appends 0 to 7 bytes with values equal to Zero, such that the total buffer length becomes a multiple of 8). This would typically be the last formatting element in a set, but this is not enforced.
KM_APPEND_ZERO_WORD_PADDING	Zero pads the buffer to the next 32-bit word boundary.
KM_APPEND_INV_XOR_CHECKSUM	Calculates and adds a checksum byte to the buffer. The checksum is calculated as the complement of the bytewise XOR of the buffer being built.
KM_DEFINE_IV_FOR_CBC	Allows definition of an IV so that 3DES_CBC wrapping can be performed even though the functionality is invoked with the CKM_3DES_ECB mechanism. The field type identifier is followed by a 32-bit length field, which must be set to 8. The length is followed by exactly 8 bytes of data which are used as the IV for the wrapping operation.

Examples

To wrap just the private exponent of an RSA key in big endian representation, the parameters would appear as follows:

> **Note:** Ensure that the packing alignment for your structures uses one (1) byte boundaries.

```
struct
{
UInt32 version = 0;
UInt32 elementType = KM_APPEND_ATTRIBUTE;
CK_ATTRIBUTE_TYPE attribute = CKA_PRIVATE_EXPONENT;
}
```

To wrap the set of RSA key components Prime1, Prime2, Coefficient, Exponent1, Exponent2 in little endian representation with a leading byte of 0x05 and ending with a CRC byte and then zero padding, the parameters would appear in a packed structure as follows:

```
struct
{
UInt32 version = 0;
UInt32 elementType1 = KM_APPEND_STRING;
UInt32 length = 1;
```



```
UInt8 byteValue = 5;
UInt32 elementType2 = KM_APPEND_REVERSED_ATTRIBUTE;
CK_ATTRIBUTE_TYPE attribute1 = CKA_PRIME_1;
UInt32 elementType3 = KM_APPEND_REVERSED_ATTRIBUTE;
CK_ATTRIBUTE_TYPE attribute2 = CKA_PRIME_2;
UInt32 elementType4 = KM_APPEND_REVERSED_ATTRIBUTE;
CK_ATTRIBUTE_TYPE attribute3 = CKA_COEFFICIENT;
UInt32 elementType5 = KM_APPEND_REVERSED_ATTRIBUTE;
CK_ATTRIBUTE_TYPE attribute4 = CKA_EXPONENT_1;
UInt32 elementType6 = KM_APPEND_REVERSED_ATTRIBUTE;
CK_ATTRIBUTE_TYPE attribute5 = CKA_EXPONENT_2;
UInt32 elementType7 = KM_APPEND_INV_XOR_CHECKSUM;
UInt32 elementType8 = KM_APPEND_ZERO_PADDING;
}
```

Derivation of Symmetric Keys with 3DES_ECB

SafeNet supports derivation of symmetric keys by the encryption of "diversification data" with a base key. Access to the derivation functionality is through the PKCS #11 C_DeriveKey function with the CKM_DES3_ECB and CKM_DES_ECB mechanism. Diversification data is provided as the mechanism parameter. The derived key can be any type of symmetric key. The encrypted data forms the CKA_VALUE attribute of the derived key. A template provided as a parameter to the C_DeriveKey function defines all other attributes.

Rules for the derivation are as follows:

- The Base Key must be of type CKK_DES2 or CKK_DES3 when using CKM_DES3_ECB. It must be of type CKK_DES when using CKM_DES_ECB.
- The base key must have its CKA_DERIVE attribute set to TRUE.
- The template for the derived key must identify the key type (CKA_KEY_TYPE) and length (CKA_VALUE_LEN). The type and length must be compatible. The length can be omitted if the key type supports only one length. (E.g., If key type is CKK_DES2, the length must either be explicitly defined as 16, or be omitted to allow the value to default to 16). Other attributes in the template must be consistent with the security policy settings of the SafeNet HSM.
- The derivation mechanism must be set to CKM_DES3_ECB or CKM_DES_ECB, the mechanism parameter pointer must point to the diversification data, and the mechanism parameter length must be set to the diversification data length.
- The diversification data must be the same length as the key to be derived, with one exception. If the key to be derived is16 bytes, the base key is CKK_DES2 and the diversification data is only 8 bytes, then the data is encrypted twice - once with the base key and once with the base key with its halves reversed. Joining the two encrypted pieces forms the derived key.
- If the derived key is of type CKK_DES, CKK_DES2 or CKK_DES3, odd key parity is applied to the new key value immediately following the encryption of the diversification data. The encrypted data is taken as-is for the formation of all other types of symmetric keys.

PKCS # 11 Extension HA Status Call

A SafeNet extension to the PKCS#11 standard allows query of the HA group state.

Function Definition

```
CK_RV CK_ENTRY CA_GetHAState( CK_SLOT_ID slotId, CK_HA_STATE_PTR pState );
```


The structure definitions for a CK_HA_STATE_PTR and CK_HA_MEMBER are:

```
typedef struct CK_HA_MEMBER{
CK_ULONG memberSerial;
CK_RV memberStatus;
}CK_HA_MEMBER;

typedef struct CK_HA_STATUS{
CK_ULONG groupSerial;
CK_HA_MEMBER memberList[CK_HA_MAX_MEMBERS];
CK_USHORT listSize;
}CK_HA_STATUS;
```

See the JavaDocs included with the software for a description of the Java methods derived from this cryptoki function.

Pseudorandom Function KDF Mechanisms

The SafeNet HSMs support the following two vendor defined mechanisms. They can be used to perform Counter Mode KDF (key derivation functions) using various CMAC algorithms (DES3, AES, ARIA, SEED) as the PRF (pseudo-random function). See NIST SP 800-108. These mechanisms are available in firmware 6.2.1 and later.

```
#define CKM_NIST_PRF_KDF                        (CKM_VENDOR_DEFINED + 0xA02)
#define CKM_PRF_KDF                             (CKM_VENDOR_DEFINED + 0xA03)

/* Parameter and values used with CKM_PRF_KDF and * CKM_NIST_PRF_KDF. */

typedef CK_ULONG CK_KDF_PRF_TYPE;
typedef CK_ULONG CK_KDF_PRF_ENCODING_SCHEME;

/** PRF KDF types */
#define CK_NIST_PRF_KDF_DES3_CMAC       0x00000001
#define CK_NIST_PRF_KDF_AES_CMAC        0x00000002
#define CK_PRF_KDF_ARIA_CMAC            0x00000003
#define CK_PRF_KDF_SEED_CMAC            0x00000004

#define LUNA_PRF_KDF_ENCODING_SCHEME_1      0x00000000
#define LUNA_PRF_KDF_ENCODING_SCHEME_2      0x00000001

typedef struct CK_KDF_PRF_PARAMS {
   CK_KDF_PRF_TYPE            prfType;
   CK_BYTE_PTR                pLabel;
   CK_ULONG                   ulLabelLen;
   CK_BYTE_PTR                pContext;
   CK_ULONG                   ulContextLen;
   CK_ULONG                   ulCounter;
   CK_KDF_PRF_ENCODING_SCHEME ulEncodingScheme;
} CK_PRF_KDF_PARAMS;

typedef CK_PRF_KDF_PARAMS CK_PTR CK_KDF_PRF_PARAMS_PTR;
```

Derive Template

The CKA_DERIVE_TEMPLATE attribute is an optional extension to the C_DeriveKey function. This attribute points to an array template which provides additional security by restricting important attributes in the resulting derived key. This

derive template, along with the user-supplied application template (called pTemplate in the PKCS#11 specification), determine the attributes of the derived key.

To invoke a derive template, the base key must have the CKA_DERIVE_TEMPLATE attribute set, pointing to a user-supplied derive template. When you specify this attribute on the base key and then attempt to derive a key, the derive operation adds the attributes of the application template to the attributes in the derive template. If there are any mismatches between attribute values specified in the two templates, the derive operation fails. Otherwise, the operation succeeds, producing a derived key with the combined attributes of the two templates.

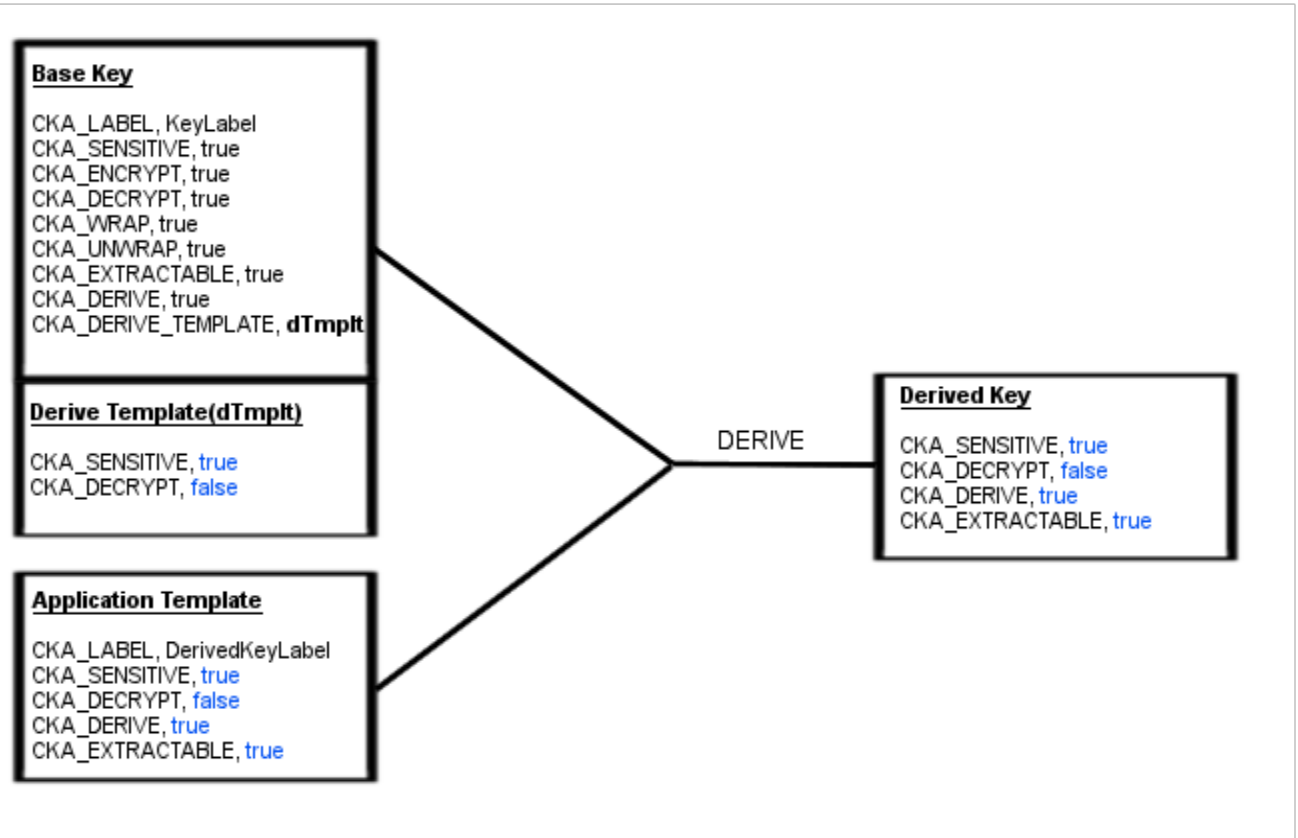
Any and all attributes which are valid for application template of a particular mechanism are also valid for the derive template. For security, the most effective attributes to restrict are those which might allow the derived key to be misused or expose secret information. Broadly these include but are not limited to encryption/decryption capabilities, extractability, the CKA_SENSITIVE attribute and the CKA_MODIFIABLE attribute. All mechanisms which support key derivation also support derive templates.

Examples

The following examples show a successful derivation with a derive template, and a failed derivation.

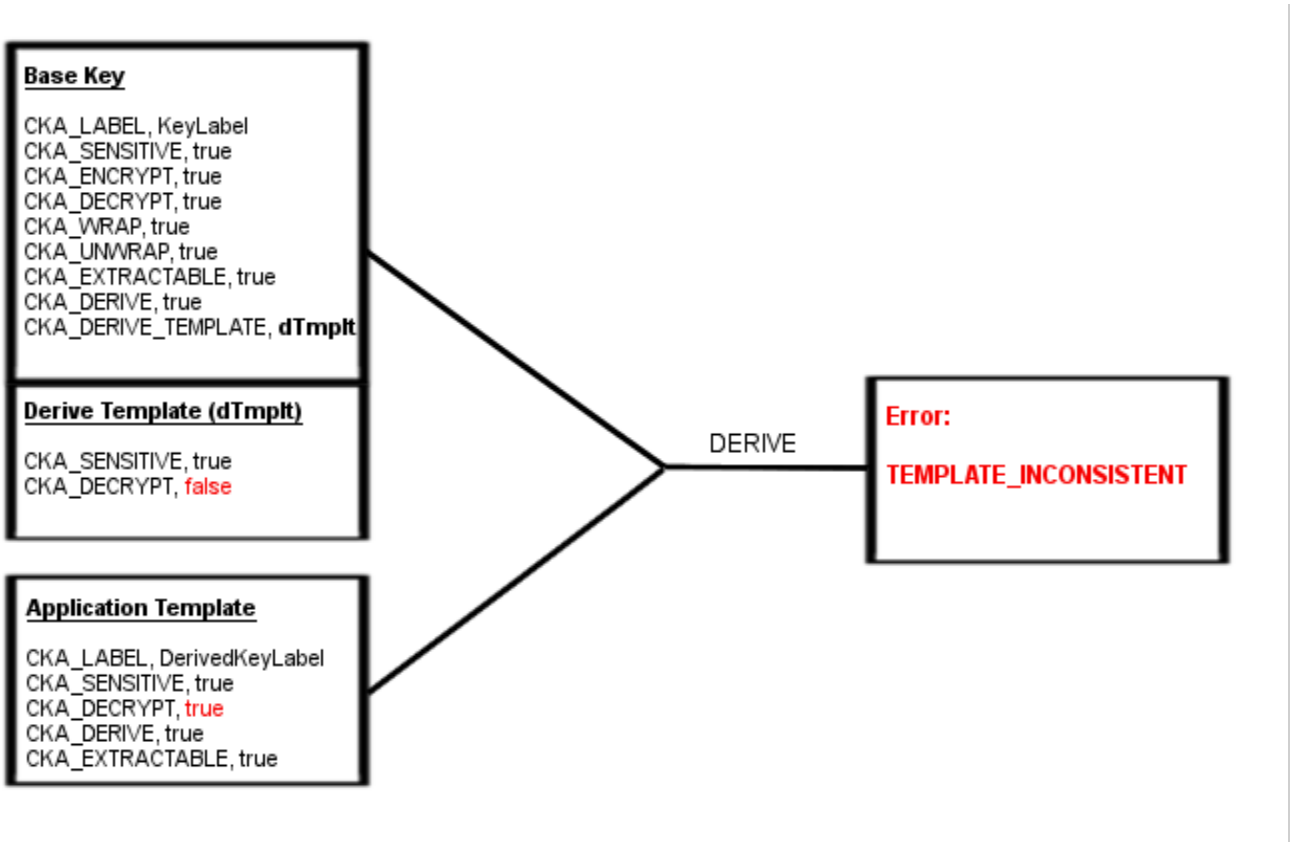
Successful Derivation

Here, the base key has the CKA_DERIVE_TEMPLATE attribute pointing to the derive template dTmplt. There are no conflicts between dTmplt and the application template. The application template's extra attributes are added to dTmplt's attributes, and the derivation operation produces a derived key containing the attributes in the two templates.



Failed Derivation

Here, the base key has the CKA_DERIVE_TEMPLATE attribute pointing to the derive template dTmplt. Notice that dTmplt has the CKA_DECRYPT attribute set to false, where the application template has the CKA_DECRYPT attribute set to true. This conflict causes the derivation operation to fail with the error TEMPLATE_INCONSISTENT.



4
Supported Mechanisms

This chapter provides an alphabetical listing of the supported PKCS #11 standard mechanisms and SafeNet proprietary mechanisms.

Mechanism Remap for FIPS Compliance

Under FIPS 186-3/4, the only RSA methods permitted for generating keys are 186-3 with primes and 186-3 with aux primes. This means that RSA PKCS and X9.31 key generation is no longer approved for operation in a FIPS-compliant HSM. Firmware version 6.2.1 and older supported only PKCS and X9.31, and these were allowed in FIPS mode. Firmware versions 6.10 through 6.21 provide the newer mechanisms, and allow both older and newer mechanisms in FIPS mode. Firmware versions 6.22.0 and newer do not allow PKCS and X9.31 in FIPS mode.

Firmware Version	Supported Mechanisms	FIPS-mode Allowed Mechanisms
fw <= 6.2.1	PKCS, X9.31	PKCS, X9.31
6.10 <= fw <= 6.21	PKCS, X9.31, 186-3 with primes, 186-3 with aux primes	PKCS, X9.31, 186-3 with primes, 186-3 with aux primes
fw >= 6.22.0	PKCS, X9.31, 186-3 with primes, 186-3 with aux primes	186-3 with primes, 186-3 with aux primes

Mechanism Remap Configuration Settings

Two configuration settings are available in the Chrystoki.conf (Linux/UNIX) or Crystoki.ini (Windows) configuration file installed with SafeNet HSM Client, to deal with calls to newer-firmware HSMs for outdated mechanisms, or calls to older-firmware HSMs for newer mechanisms that they do not support. The configuration settings control redirecting or mapping of mechanism calls.

Redirect Old to New

Under the configuration file's [Misc] section, RSAKeyGenMechRemap can be set to 0 or 1.

- When RSAKeyGenMechRemap is set to 0 (the default) and firmware version is 6.10.x or greater, no re-mapping is performed.
- When RSAKeyGenMechRemap is set to 1 and firmware version is 6.10.x or greater, the following re-mapping occurs:
  - PKCS Key Gen --> 186-3 Prime key gen
  - X9.31 Key Gen --> 186-3 Aux Prime key gen

Note: This setting is intended for older applications, allowing them to continue to call outdated mechanisms, but have the calls redirected to newer, equivalent, FIPS-acceptable mechanisms, while your software development or integration catches up.

The following table summarizes the possible combinations, for firmware versions that are supported in SafeNet HSM 6.0 and later.

Firmware version	State of RSAKeyGen MechRemap	Action in your application	Result
6.2.x	N/A	N/A	• RSAKeyGenMechRemap has no effect
6.10- through-6.21	0	Call PKCS Key Gen or X9.31 Key Gen	• PKCS Key Gen or X9.31 Key Gen is called and runs as requested • redirect is not set, and does not occur
	1		• call is redirected and 186-3 Prime key gen or 186-3 Aux Prime key gen is run
	0	Call 186-3 Prime key gen or 186-3 Aux Prime key gen	• either set of mechanisms is available • 186-3 Prime key gen or 186-3 Aux Prime key gen is run as requested
	1		• either set of mechanisms is available • 186-3 Prime key gen or 186-3 Aux Prime key gen is run as requested
6.22.0 or newer	0	Call PKCS Key Gen or X9.31 Key Gen	• Error message; old mechanism does not exist and no redirect is indicated [see Note 1]
	1		• old mechanisms do not exist in FIPS mode; new ones exist • call is redirected and 186-3 Prime key gen or 186-3 Aux Prime key gen is run
Note 1: Calling an unsupported mechanism, where no redirect is in place, yields error CKR_MECHANISM_ INVALID			
Note 2: If RSA-PKCS keys or X9.31 keys were previously created by an older firmware version, and firmware is updated to version 6.22.0, then : • keys of size 2048 or 3072 bits can still be used for sign and verify operations • keys of size 1024-up-to-4096 bits can be used to verify existing signatures, only. • when FIPS186-4 with SP800-131A is applied, it disallows RSA 4096-bit keys for signing			

In FIPS mode

When RSAKeyGenMechRemap is enabled,

1. CKM_RSA_PKCS_KEY_PAIR_GEN is inserted into the C_GetMechanismList output by the client library, as the

HSM does not return it in FIPS mode.

2. C_GetMechanismInfo for CKM_RSA_PKCS_KEY_PAIR_GEN returns the default Mechanism information from the client library. In FIPS mode, the HSM does not return it.

When RSAKeyGenMechRemap is disabled

1. CKM_RSA_PKCS_KEY_PAIR_GEN is not returned by C_GetMachanismList.
2. C_GetMachanismInfo for CKM_RSA_PKCS_KEY_PAIR_GEN results in an Invalid Mechanism Attribute error.

Redirect New to Old

Under the configuration file's [Misc] section, RSAPre1863KeyGenMechRemap can be set to 0 or 1.

- When RSAPre1863KeyGenMechRemap is set to 0 (the default) and firmware is version 6.2.x, no re-mapping is performed.
- When RSAPre1863KeyGenMechRemap is set to 1 and firmware is version 6.2.x, the following re-mapping occurs:
  - 186-3 Prime key gen --> PKCS Key Gen
  - 186-3 Aux Prime key gen --> X9.31 Key Gen

> ⚠ **CAUTION:** This setting is intended for evaluation purposes, such as with existing integrations that require newer mechanisms, before you update to firmware that actually supports the more secure mechanisms. Be careful with this setting, which makes it appear you are getting a new, secure mechanism, when really you are getting an outdated, insecure mechanism.

The following table summarizes the possible combinations, for firmware versions that are supported in SafeNet HSM 6.0 and later.

Firmware version	State of RSAPre1863 KeyGen MechRemap	Action in your application	Result
6.2.x	0	Call PKCS Key Gen or X9.31 Key Gen	• PKCS Key Gen or X9.31 Key Gen is called and runs
	1		• PKCS Key Gen or X9.31 Key Gen is called and runs
	0	Call 186-3 Prime key gen or 186-3 Aux Prime key gen	• Call fails; new mechanism does not exist
	1		• PKCS Key Gen or X9.31 Key Gen is called and runs • new mechanism does not exist; redirect to old [see Note 1]
6.10- through-6.22	N/A	N/A	• RSAPre1863KeyGenMechRemap has no effect

Note 1: The inclusion of redirection to the outdated mechanisms, where the firmware does not support the newer

Firmware version	State of RSAPre1863 KeyGen MechRemap	Action in your application	Result
mechanisms, allows you to [re-]write your implementation to call the newer, FIPS-approved mechanisms, yet allows you to use that application with older-firmware HSMs - perhaps in a mixed or transitioning environment.			

CKM_2DES_DERIVE

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	192
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	Not Listed

CKM_AES_CBC

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	AES
Algorithms	AES
Modes	CBC
Flags	Extractable

CKM_AES_CBC_ENCRYPT_DATA

Summary

FIPS approved?	Yes
Supported functions	Derive
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	AES
Algorithms	None
Modes	None
Flags	None

CKM_AES_CBC_PAD

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	AES
Algorithms	AES
Modes	CBC_PAD
Flags	Extractable

CKM_AES_CBC_PAD_EXTRACT

Summary

FIPS approved?	No
Supported functions	None
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	AES
Algorithms	None
Modes	None
Flags	Not Listed

CKM_AES_CBC_PAD_EXTRACT_DOMAIN_CTRL

Summary

FIPS approved?	No
Supported functions	None
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	AES
Algorithms	None
Modes	None
Flags	Not Listed

CKM_AES_CBC_PAD_EXTRACT_FLATTENED

Summary

FIPS approved?	No
Supported functions	None
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	AES
Algorithms	None
Modes	None
Flags	Not Listed

CKM_AES_CBC_PAD_EXTRACT_PUBLIC

Summary

FIPS approved?	No
Supported functions	None
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	AES
Algorithms	None
Modes	None
Flags	Not Listed

CKM_AES_CBC_PAD_EXTRACT_PUBLIC_FLATTENED

Summary

FIPS approved?	No
Supported functions	None
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	AES
Algorithms	None
Modes	None
Flags	Not Listed

CKM_AES_CBC_PAD_INSERT

Summary

FIPS approved?	No
Supported functions	None
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	AES
Algorithms	None
Modes	None
Flags	Not Listed

CKM_AES_CBC_PAD_INSERT_DOMAIN_CTRL

Summary

FIPS approved?	No
Supported functions	None
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	AES
Algorithms	None
Modes	None
Flags	Not Listed

CKM_AES_CBC_PAD_INSERT_FLATTENED

Summary

FIPS approved?	No
Supported functions	None
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	AES
Algorithms	None
Modes	None
Flags	Not Listed

CKM_AES_CBC_PAD_INSERT_PUBLIC

Summary

FIPS approved?	No
Supported functions	None
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	AES
Algorithms	None
Modes	None
Flags	Not Listed

CKM_AES_CBC_PAD_INSERT_PUBLIC_FLATTENED

Summary

FIPS approved?	No
Supported functions	None
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	AES
Algorithms	None
Modes	None
Flags	Not Listed

CKM_AES_CBC_PAD_IPSEC

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	AES
Algorithms	AES
Modes	CBC_PAD_IPSEC
Flags	Extractable

CKM_AES_CFB8

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	1
Key types	AES
Algorithms	AES
Modes	CFB
Flags	Extractable

CKM_AES_CFB128

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	16
Key types	AES
Algorithms	AES
Modes	CFB
Flags	Extractable

CKM_AES_CMAC

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	AES
Algorithms	AES
Modes	MAC
Flags	Extractable | CMAC

CKM_AES_CTR

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	AES
Algorithms	AES
Modes	CTR
Flags	Extractable

CKM_AES_ECB

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	AES
Algorithms	AES
Modes	ECB
Flags	Extractable

CKM_AES_ECB_ENCRYPT_DATA

Summary

FIPS approved?	Yes
Supported functions	Derive
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	AES
Algorithms	None
Modes	None
Flags	None

CKM_AES_GCM

GCM is the Galois/Counter Mode of operation of the AES algorithm for symmetric key encryption.

Note: If GCM input is confined to data that will not be encrypted, then you can use GMAC (see "CKM_AES_GMAC" on the next page) instead for much better performance in authentication-only mode on the input data.

Note: Do not attempt to use mechanism CKM_AES_GCM for data bigger than 16kilobytes.

Note: Random initialization vector (IV) is supported and recommended for GCM and for GMAC. In FIPS mode, the HSM firmware does not accept the IV parameter, and instead returns a generated IV.

If you are using CKDEMO to try CKM_AES_GCM, be aware that CKDEMO generally defaults to an external IV, so for an HSM in FIPS mode, be sure to set CKDEMO menu item 98 Options, item 11 - GCM IV Source to "internal" instead of "external", otherwise CKM_AES_GCM would return CKR_MECHANISM_PARAM_INVALID)

Note: Our GMAC and GCM are single part operations, so even if they are called using multi-part API, we accumulate the data (up to a maximum) and return data only on the “final” operation. That is the meaning of "Accumulating" in the table, below.

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	AES
Algorithms	AES
Modes	GCM
Flags	Extractable | Accumulating

CKM_AES_GMAC

GCM is the Galois/Counter Mode of operation of the AES algorithm for symmetric key encryption. GMAC is a variant of GCM for sign/verify operation. If GCM input is confined to data that will not be encrypted, then GMAC is purely an authentication mode on the input data. The SafeNet HSM GMAC implementation, formerly invoked only via PKCS#11 interface, can now be accessed via JCPROV and via our Java Provider (see Notes, below).

The GMAC implementation follows NIST SP-800-38D. It supports AES symmetric key sizes of 128, 192, and 256 bits.

If the HSM is in FIPS mode (see HSM policy 12 at HSM Capabilities and Policies), the initialization vector (IV) is generated in the HSM and returned to the PKCS#11 function call. The buffer must be large enough to store the GMAC tag plus the generated IV (which has a length of 16 bytes).

If the HSM is **not** in FIPS mode, then the developer is responsible to specify an IV. Random IV is supported and recommended for GCM and GMAC. If you are not using random IV, then the most efficient IV value length is 12 bytes; any other size reduces performance and requires more work (per NIST SP-800-38D).

Note: For PKCS#11, to achieve highest performance, use the Gemalto-SafeNet defined CK_AES_GMAC_PARAMS to define the GMAC operation parameters (additional authenticated data, tag size, IV and the IV size). To initialize the sign operation, use the CKM_AES_GMAC mechanism.

For authentication, it is possible to use CKM_AES_GCM mechanism, when passing no data to encrypt (for strict compliance with NIST specification), and performance in that mode is better than in previous SafeNet releases. However, expect lower performance than would be obtained from CKM_AES_GMAC for the same purpose.

Note: JCPROV - at time of writing (August 2015) GMAC is supported, but GCM is not. Use CK_AES_CMAC_PARAMS.java to define the GMAC operation. Implementation is the same as for PKCS#11.

Note: Java Provider (JSP) - both GMC and GMAC are supported. "GmacAesDemo.java" provides a sample for using GMAC with Java.

Java Parameter Specification class LunaGmacParameterSpec.java defines default values recommended by the NIST specification.

Note: The correlation is not exact but, as a general rule for a given mechanism, invocation by PKCS#11 API always provides the best performance, JSP performance is close but lower due to Java architecture, and JCPROV performance is somewhat lower still than PKCS#11 and JSP performance levels.

Note: Our GMAC and GCM are single part operations, so even if they are called using multi-part API, we accumulate the data (up to a maximum) and return data only on the “final” operation. That is the meaning of "Accumulating" in the table, below.

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	AES
Algorithms	AES
Modes	GCM
Flags	Extractable | Accumulating

CKM_AES_KEY_GEN

Summary

FIPS approved?	Yes
Supported functions	Generate Key
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	AES
Algorithms	None
Modes	None
Flags	None

CKM_AES_KW

NIST Special Publication 800-38F describes cryptographic methods that are approved for "key wrapping," that is, the protection of the confidentiality and integrity of cryptographic keys. In addition to describing existing methods, that publication specifies two new, deterministic authenticated-encryption modes of operation of the Advanced Encryption Standard (AES) algorithm: the AES Key Wrap (KW) mode and the AES Key Wrap With Padding (KWP) mode. Gemalto's SafeNet HSMs implement the AES Key Wrap (KW) mode at this time, which SP800-38F recommends as more secure than CKM_AES_CBC. Your HSM must have firmware 6.24 installed, in order to make use of the new mechanism.

Note: NIST Special Publication 800-38F recommends this method as more secure than CKM_AES_CBC.

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	8
Digest size	0
Key types	AES
Algorithms	AES
Modes	KW
Flags	Extractable | Accumulating

CKM_AES_MAC

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	AES
Algorithms	AES
Modes	MAC
Flags	Extractable

CKM_AES_OFB

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	AES
Algorithms	AES
Modes	OFB
Flags	Extractable

CKM_ARIA_CBC

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	ARIA
Algorithms	ARIA
Modes	CBC
Flags	Extractable

CKM_ARIA_CBC_ENCRYPT_DATA

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	ARIA
Algorithms	None
Modes	None
Flags	None

CKM_ARIA_CBC_PAD

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	ARIA
Algorithms	ARIA
Modes	CBC_PAD
Flags	Extractable

CKM_ARIA_CFB8

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	1
Key types	ARIA
Algorithms	ARIA
Modes	CFB
Flags	Extractable

CKM_ARIA_CFB128

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	16
Key types	ARIA
Algorithms	ARIA
Modes	CFB
Flags	Extractable

CKM_ARIA_CMAC

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	ARIA
Algorithms	ARIA
Modes	MAC
Flags	Extractable | CMAC

CKM_ARIA_CTR

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	ARIA
Algorithms	ARIA
Modes	CTR
Flags	Extractable

CKM_ARIA_ECB

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	ARIA
Algorithms	ARIA
Modes	ECB
Flags	Extractable

CKM_ARIA_ECB_ENCRYPT_DATA

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	ARIA
Algorithms	None
Modes	None
Flags	None

CKM_ARIA_GCM

Note: Our GCM is a single part operation, so even if it is called using multi-part API, we accumulate the data (up to a maximum) and return data only on the “final” operation. That is the meaning of "Accumulating" in the table, below.

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	ARIA
Algorithms	ARIA
Modes	GCM
Flags	Extractable | Accumulating

CKM_ARIA_KEY_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	0
Digest size	0
Key types	ARIA
Algorithms	None
Modes	None
Flags	None

CKM_ARIA_L_CBC

Summary

FIPS approved?	No
Supported functions	Decrypt | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	ARIA
Algorithms	ARIA
Modes	CBC
Flags	Extractable

CKM_ARIA_L_CBC_PAD

Summary

FIPS approved?	No
Supported functions	Decrypt | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	ARIA
Algorithms	ARIA
Modes	CBC_PAD
Flags	Extractable

CKM_ARIA_L_ECB

Summary

FIPS approved?	No
Supported functions	Decrypt | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	ARIA
Algorithms	ARIA
Modes	ECB
Flags	Extractable

CKM_ARIA_L_MAC

Summary

FIPS approved?	No
Supported functions	Verify
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	ARIA
Algorithms	ARIA
Modes	MAC
Flags	Extractable

CKM_ARIA_MAC

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	ARIA
Algorithms	ARIA
Modes	MAC
Flags	Extractable

CKM_ARIA_OFB

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	16
Digest size	0
Key types	ARIA
Algorithms	ARIA
Modes	OFB
Flags	Extractable

CKM_CAST3_CBC

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	8
Digest size	0
Key types	CAST3
Algorithms	CAST3
Modes	CBC
Flags	Extractable

CKM_CAST3_CBC_PAD

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	8
Digest size	0
Key types	CAST3
Algorithms	CAST3
Modes	CBC_PAD
Flags	Extractable

CKM_CAST3_ECB

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	8
Digest size	0
Key types	CAST3
Algorithms	CAST3
Modes	ECB
Flags	Extractable

CKM_CAST3_KEY_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	0
Digest size	0
Key types	CAST3
Algorithms	None
Modes	None
Flags	None

CKM_CAST3_MAC

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	8
Digest size	0
Key types	CAST3
Algorithms	CAST3
Modes	MAC
Flags	Extractable

CKM_CAST5_CBC

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	8
Digest size	0
Key types	CAST5
Algorithms	CAST5
Modes	CBC
Flags	Extractable

CKM_CAST5_CBC_PAD

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	8
Digest size	0
Key types	CAST5
Algorithms	CAST5
Modes	CBC_PAD
Flags	Extractable

CKM_CAST5_ECB

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	8
Digest size	0
Key types	CAST5
Algorithms	CAST5
Modes	ECB
Flags	Extractable

CKM_CAST5_KEY_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	0
Digest size	0
Key types	CAST5
Algorithms	None
Modes	None
Flags	None

CKM_CAST5_MAC

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	8
Digest size	0
Key types	CAST5
Algorithms	CAST5
Modes	MAC
Flags	Extractable

CKM_CONCATENATE_BASE_AND_DATA

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	Internal

CKM_CONCATENATE_BASE_AND_KEY

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	Internal

CKM_CONCATENATE_DATA_AND_BASE

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	Internal

CKM_CONCATENATE_KEY_AND_BASE

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	Internal

CKM_DES_CBC

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	8
Digest size	0
Key types	DES
Algorithms	DES
Modes	CBC
Flags	Extractable

CKM_DES_CBC_ENCRYPT_DATA

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

CKM_DES_CBC_PAD

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	8
Digest size	0
Key types	DES
Algorithms	DES
Modes	CBC_PAD
Flags	Extractable

CKM_DES_ECB

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	8
Digest size	0
Key types	DES
Algorithms	DES
Modes	ECB
Flags	Extractable

CKM_DES_ECB_ENCRYPT_DATA

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

CKM_DES_KEY_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	0
Digest size	0
Key types	DES
Algorithms	None
Modes	None
Flags	None

CKM_DES_MAC

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	8
Digest size	0
Key types	DES
Algorithms	DES
Modes	MAC
Flags	Extractable

CKM_DES2_DUKPT_DATA

The CKM_DES2_DUKPT family of key derive mechanisms create keys used to protect EFTPOS terminal sessions. The mechanisms implement the algorithm for server side DUKPT derivation as defined by ANSI X9.24 part 1.

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	192
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

Usage

This mechanism has the following attributes:

- Only CKK_DES2 keys can be derived. The mechanism will force the CKA_KEY_TYPE attribute of the derived object to equal CKK_DES2. If the template does specify a CKA_KEY_TYPE attribute then it must be CKK_DES2.
- The mechanism takes a CK_KEY_DERIVATION_STRING_DATA structure as a parameter.
- The pData field of the parameter must point to a 10 byte array which holds the 80 bit Key Sequence Number (KSN).
- This mechanism contributes the CKA_CLASS and CKA_KEY_TYPE and CKA_VALUE to the resulting object.

The DUKPT MAC and DATA versions will default to the appropriate usage mechanism as described in the following table:

Mechanism	CKA_SIGN	CKA_VERIFY	CKA_DECRYPT	CKA_ENCRYPT
CKM_DES2_DUKPT_MAC	True	True		
CKM_DES2_DUKPT_MAC_RESP	True			
CKM_DES2_DUKPT_DATA			True	True
CKM_DES2_DUKPT_DATA_RESP				True

Example

```
#define CKM_DES2_DUKPT_PIN              (CKM_VENDOR_DEFINED + 0x611)
#define CKM_DES2_DUKPT_MAC              (CKM_VENDOR_DEFINED + 0x612)
#define CKM_DES2_DUKPT_MAC_RESP         (CKM_VENDOR_DEFINED + 0x613)
#define CKM_DES2_DUKPT_DATA             (CKM_VENDOR_DEFINED + 0x614)
#define CKM_DES2_DUKPT_DATA_RESP        (CKM_VENDOR_DEFINED + 0x615)

CK_OBJECT_HANDLE hBDKey; // handle of CKK_DES2 or CKK_DES2 Base Derive Key
CK_OBJECT_HANDLE hMKey;   // handle of CKK_DES2 MAC session Key
CK_MECHANISM svMech = { CKM_DES3_X919_MAC , NULL, 0};

CK_KEY_DERIVATION_STRING_DATA param;
CK_MECHANISM kdMech = { CKM_DES2_DUKPT_MAC , NULL, 0};
CK_CHAR ksn[10];
CK_CHAR inp[any length];
CK_CHAR mac[4];
CK_SIZE len;

// Derive MAC verify session key
param.pData=ksn;
param.ulLen = 10;

kdMech.mechanism = CKM_DES2_DUKPT_MAC;
kdMech.pParameter = &param;
kdMech.ulParameterLen = sizeof parram;

C_DeriveKey(hSes, &kdMech, hBDKey , NULL, 0, &hMKey);

// Single part verify operation

C_VerifyInit(hSes, &svMech, hMKey);
len = sizeof mac;
C_Verify(hSes, inp, sizeof inp, mac, len);

// clean up

C_DestroyObject(hSes, hMKey);

// Test vectors
```

CKM_DES2_DUKPT_DATA_RESP

The CKM_DES2_DUKPT family of key derive mechanisms create keys used to protect EFTPOS terminal sessions. The mechanisms implement the algorithm for server side DUKPT derivation as defined by ANSI X9.24 part 1.

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	192
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

Usage

This mechanism has the following attributes:

- Only CKK_DES2 keys can be derived. The mechanism will force the CKA_KEY_TYPE attribute of the derived object to equal CKK_DES2. If the template does specify a CKA_KEY_TYPE attribute then it must be CKK_DES2.
- The mechanism takes a CK_KEY_DERIVATION_STRING_DATA structure as a parameter.
- The pData field of the parameter must point to a 10 byte array which holds the 80 bit Key Sequence Number (KSN).
- This mechanism contributes the CKA_CLASS and CKA_KEY_TYPE and CKA_VALUE to the resulting object.

The DUKPT MAC and DATA versions will default to the appropriate usage mechanism as described in the following table:

Mechanism	CKA_SIGN	CKA_VERIFY	CKA_DECRYPT	CKA_ENCRYPT
CKM_DES2_DUKPT_MAC	True	True		
CKM_DES2_DUKPT_MAC_RESP	True			
CKM_DES2_DUKPT_DATA			True	True
CKM_DES2_DUKPT_DATA_RESP				True

Example

```
#define CKM_DES2_DUKPT_PIN              (CKM_VENDOR_DEFINED + 0x611)
#define CKM_DES2_DUKPT_MAC              (CKM_VENDOR_DEFINED + 0x612)
#define CKM_DES2_DUKPT_MAC_RESP         (CKM_VENDOR_DEFINED + 0x613)
#define CKM_DES2_DUKPT_DATA             (CKM_VENDOR_DEFINED + 0x614)
#define CKM_DES2_DUKPT_DATA_RESP        (CKM_VENDOR_DEFINED + 0x615)

CK_OBJECT_HANDLE hBDKey; // handle of CKK_DES2 or CKK_DES2 Base Derive Key
CK_OBJECT_HANDLE hMKey;   // handle of CKK_DES2 MAC session Key
CK_MECHANISM svMech = { CKM_DES3_X919_MAC , NULL, 0};

CK_KEY_DERIVATION_STRING_DATA param;
CK_MECHANISM kdMech = { CKM_DES2_DUKPT_MAC , NULL, 0};
CK_CHAR ksn[10];
CK_CHAR inp[any length];
CK_CHAR mac[4];
CK_SIZE len;

// Derive MAC verify session key
param.pData=ksn;
param.ulLen = 10;

kdMech.mechanism = CKM_DES2_DUKPT_MAC;
kdMech.pParameter = &param;
kdMech.ulParameterLen = sizeof parram;

C_DeriveKey(hSes, &kdMech, hBDKey , NULL, 0, &hMKey);

// Single part verify operation

C_VerifyInit(hSes, &svMech, hMKey);
len = sizeof mac;
C_Verify(hSes, inp, sizeof inp, mac, len);

// clean up

C_DestroyObject(hSes, hMKey);

// Test vectors
```

CKM_DES2_DUKPT_MAC

The CKM_DES2_DUKPT family of key derive mechanisms create keys used to protect EFTPOS terminal sessions. The mechanisms implement the algorithm for server side DUKPT derivation as defined by ANSI X9.24 part 1.

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	192
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

Usage

This mechanism has the following attributes:

- Only CKK_DES2 keys can be derived. The mechanism will force the CKA_KEY_TYPE attribute of the derived object to equal CKK_DES2. If the template does specify a CKA_KEY_TYPE attribute then it must be CKK_DES2.
- The mechanism takes a CK_KEY_DERIVATION_STRING_DATA structure as a parameter.
- The pData field of the parameter must point to a 10 byte array which holds the 80 bit Key Sequence Number (KSN).
- This mechanism contributes the CKA_CLASS and CKA_KEY_TYPE and CKA_VALUE to the resulting object.

The DUKPT MAC and DATA versions will default to the appropriate usage mechanism as described in the following table:

Mechanism	CKA_SIGN	CKA_VERIFY	CKA_DECRYPT	CKA_ENCRYPT
CKM_DES2_DUKPT_MAC	True	True		
CKM_DES2_DUKPT_MAC_RESP	True			
CKM_DES2_DUKPT_DATA			True	True
CKM_DES2_DUKPT_DATA_RESP				True

Example

```
#define CKM_DES2_DUKPT_PIN              (CKM_VENDOR_DEFINED + 0x611)
#define CKM_DES2_DUKPT_MAC              (CKM_VENDOR_DEFINED + 0x612)
#define CKM_DES2_DUKPT_MAC_RESP         (CKM_VENDOR_DEFINED + 0x613)
#define CKM_DES2_DUKPT_DATA             (CKM_VENDOR_DEFINED + 0x614)
#define CKM_DES2_DUKPT_DATA_RESP        (CKM_VENDOR_DEFINED + 0x615)

CK_OBJECT_HANDLE hBDKey; // handle of CKK_DES2 or CKK_DES2 Base Derive Key
CK_OBJECT_HANDLE hMKey;   // handle of CKK_DES2 MAC session Key
CK_MECHANISM svMech = { CKM_DES3_X919_MAC , NULL, 0};

CK_KEY_DERIVATION_STRING_DATA param;
CK_MECHANISM kdMech = { CKM_DES2_DUKPT_MAC , NULL, 0};
CK_CHAR ksn[10];
CK_CHAR inp[any length];
CK_CHAR mac[4];
CK_SIZE len;

// Derive MAC verify session key
param.pData=ksn;
param.ulLen = 10;

kdMech.mechanism = CKM_DES2_DUKPT_MAC;
kdMech.pParameter = &param;
kdMech.ulParameterLen = sizeof parram;

C_DeriveKey(hSes, &kdMech, hBDKey , NULL, 0, &hMKey);

// Single part verify operation

C_VerifyInit(hSes, &svMech, hMKey);
len = sizeof mac;
C_Verify(hSes, inp, sizeof inp, mac, len);

// clean up

C_DestroyObject(hSes, hMKey);

// Test vectors
```

CKM_DES2_DUKPT_MAC_RESP

The CKM_DES2_DUKPT family of key derive mechanisms create keys used to protect EFTPOS terminal sessions. The mechanisms implement the algorithm for server side DUKPT derivation as defined by ANSI X9.24 part 1.

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	192
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

Usage

This mechanism has the following attributes:

- Only CKK_DES2 keys can be derived. The mechanism will force the CKA_KEY_TYPE attribute of the derived object to equal CKK_DES2. If the template does specify a CKA_KEY_TYPE attribute then it must be CKK_DES2.
- The mechanism takes a CK_KEY_DERIVATION_STRING_DATA structure as a parameter.
- The pData field of the parameter must point to a 10 byte array which holds the 80 bit Key Sequence Number (KSN).
- This mechanism contributes the CKA_CLASS and CKA_KEY_TYPE and CKA_VALUE to the resulting object.

The DUKPT MAC and DATA versions will default to the appropriate usage mechanism as described in the following table:

Mechanism	CKA_SIGN	CKA_VERIFY	CKA_DECRYPT	CKA_ENCRYPT
CKM_DES2_DUKPT_MAC	True	True		
CKM_DES2_DUKPT_MAC_RESP	True			
CKM_DES2_DUKPT_DATA			True	True
CKM_DES2_DUKPT_DATA_RESP				True

Example

```
#define CKM_DES2_DUKPT_PIN              (CKM_VENDOR_DEFINED + 0x611)
#define CKM_DES2_DUKPT_MAC              (CKM_VENDOR_DEFINED + 0x612)
#define CKM_DES2_DUKPT_MAC_RESP         (CKM_VENDOR_DEFINED + 0x613)
#define CKM_DES2_DUKPT_DATA             (CKM_VENDOR_DEFINED + 0x614)
#define CKM_DES2_DUKPT_DATA_RESP        (CKM_VENDOR_DEFINED + 0x615)

CK_OBJECT_HANDLE hBDKey; // handle of CKK_DES2 or CKK_DES2 Base Derive Key
CK_OBJECT_HANDLE hMKey;   // handle of CKK_DES2 MAC session Key
CK_MECHANISM svMech = { CKM_DES3_X919_MAC , NULL, 0};

CK_KEY_DERIVATION_STRING_DATA param;
CK_MECHANISM kdMech = { CKM_DES2_DUKPT_MAC , NULL, 0};
CK_CHAR ksn[10];
CK_CHAR inp[any length];
CK_CHAR mac[4];
CK_SIZE len;

// Derive MAC verify session key
param.pData=ksn;
param.ulLen = 10;

kdMech.mechanism = CKM_DES2_DUKPT_MAC;
kdMech.pParameter = &param;
kdMech.ulParameterLen = sizeof parram;

C_DeriveKey(hSes, &kdMech, hBDKey , NULL, 0, &hMKey);

// Single part verify operation

C_VerifyInit(hSes, &svMech, hMKey);
len = sizeof mac;
C_Verify(hSes, inp, sizeof inp, mac, len);

// clean up

C_DestroyObject(hSes, hMKey);

// Test vectors
```

CKM_DES2_DUKPT_PIN

The CKM_DES2_DUKPT family of key derive mechanisms create keys used to protect EFTPOS terminal sessions. The mechanisms implement the algorithm for server side DUKPT derivation as defined by ANSI X9.24 part 1.

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	192
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

Usage

This mechanism has the following attributes:

- Only CKK_DES2 keys can be derived. The mechanism will force the CKA_KEY_TYPE attribute of the derived object to equal CKK_DES2. If the template does specify a CKA_KEY_TYPE attribute then it must be CKK_DES2.
- The mechanism takes a CK_KEY_DERIVATION_STRING_DATA structure as a parameter.
- The pData field of the parameter must point to a 10 byte array which holds the 80 bit Key Sequence Number (KSN).
- This mechanism contributes the CKA_CLASS and CKA_KEY_TYPE and CKA_VALUE to the resulting object.

The DUKPT MAC and DATA versions will default to the appropriate usage mechanism as described in the following table:

Mechanism	CKA_SIGN	CKA_VERIFY	CKA_DECRYPT	CKA_ENCRYPT
CKM_DES2_DUKPT_MAC	True	True		
CKM_DES2_DUKPT_MAC_RESP	True			
CKM_DES2_DUKPT_DATA			True	True
CKM_DES2_DUKPT_DATA_RESP				True

Example

```
#define CKM_DES2_DUKPT_PIN              (CKM_VENDOR_DEFINED + 0x611)
#define CKM_DES2_DUKPT_MAC              (CKM_VENDOR_DEFINED + 0x612)
#define CKM_DES2_DUKPT_MAC_RESP         (CKM_VENDOR_DEFINED + 0x613)
#define CKM_DES2_DUKPT_DATA             (CKM_VENDOR_DEFINED + 0x614)
#define CKM_DES2_DUKPT_DATA_RESP        (CKM_VENDOR_DEFINED + 0x615)

CK_OBJECT_HANDLE hBDKey; // handle of CKK_DES2 or CKK_DES2 Base Derive Key
CK_OBJECT_HANDLE hMKey;   // handle of CKK_DES2 MAC session Key
CK_MECHANISM svMech = { CKM_DES3_X919_MAC , NULL, 0};

CK_KEY_DERIVATION_STRING_DATA param;
CK_MECHANISM kdMech = { CKM_DES2_DUKPT_MAC , NULL, 0};
CK_CHAR ksn[10];
CK_CHAR inp[any length];
CK_CHAR mac[4];
CK_SIZE len;

// Derive MAC verify session key
param.pData=ksn;
param.ulLen = 10;

kdMech.mechanism = CKM_DES2_DUKPT_MAC;
kdMech.pParameter = &param;
kdMech.ulParameterLen = sizeof parram;

C_DeriveKey(hSes, &kdMech, hBDKey , NULL, 0, &hMKey);

// Single part verify operation

C_VerifyInit(hSes, &svMech, hMKey);
len = sizeof mac;
C_Verify(hSes, inp, sizeof inp, mac, len);

// clean up

C_DestroyObject(hSes, hMKey);

// Test vectors
```

CKM_DES2_KEY_GEN

Summary

FIPS approved?	Yes
Supported functions	Generate Key
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	0
Digest size	0
Key types	DES2
Algorithms	None
Modes	None
Flags	None

CKM_DES3_CBC

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	128
Maximum key length (bits)	192
Block size	8
Digest size	0
Key types	DES3
Algorithms	DES3
Modes	CBC
Flags	Extractable

CKM_DES3_CBC_ENCRYPT_DATA

Summary

FIPS approved?	Yes
Supported functions	Derive
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	128
Maximum key length (bits)	192
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

CKM_DES3_CBC_PAD

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	128
Maximum key length (bits)	192
Block size	8
Digest size	0
Key types	DES3
Algorithms	DES3
Modes	CBC_PAD
Flags	Extractable

CKM_DES3_CBC_PAD_IPSEC

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	192
Block size	8
Digest size	0
Key types	DES3
Algorithms	DES3
Modes	CBC_PAD_IPSEC
Flags	Extractable

CKM_DES3_CFB8

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	128
Maximum key length (bits)	192
Block size	8
Digest size	1
Key types	DES3
Algorithms	DES3
Modes	CFB
Flags	Extractable

CKM_DES3_CFB64

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	128
Maximum key length (bits)	192
Block size	8
Digest size	8
Key types	DES3
Algorithms	DES3
Modes	CFB
Flags	Extractable

CKM_DES3_CMAC

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	128
Maximum key length (bits)	192
Block size	8
Digest size	0
Key types	DES3
Algorithms	DES3
Modes	MAC
Flags	Extractable | CMAC

CKM_DES3_CTR

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	128
Maximum key length (bits)	192
Block size	8
Digest size	0
Key types	DES3
Algorithms	DES3
Modes	CTR
Flags	Extractable

CKM_DES3_ECB

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	128
Maximum key length (bits)	192
Block size	8
Digest size	0
Key types	DES3
Algorithms	DES3
Modes	ECB
Flags	Extractable

CKM_DES3_ECB_ENCRYPT_DATA

Summary

FIPS approved?	Yes
Supported functions	Derive
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	128
Maximum key length (bits)	192
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

CKM_DES3_GCM

Note: Our GCM is a single part operation, so even if it is called using multi-part API, we accumulate the data (up to a maximum) and return data only on the “final” operation. That is the meaning of "Accumulating" in the table, below.

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	128
Maximum key length (bits)	192
Block size	8
Digest size	0
Key types	DES3
Algorithms	DES3
Modes	GCM
Flags	Extractable | Accumulating

CKM_DES3_KEY_GEN

Summary

FIPS approved?	Yes
Supported functions	Generate Key
Minimum key length (bits)	192
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	192
Block size	0
Digest size	0
Key types	DES3
Algorithms	None
Modes	None
Flags	None

CKM_DES3_MAC

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	128
Maximum key length (bits)	192
Block size	8
Digest size	0
Key types	DES3
Algorithms	DES3
Modes	MAC
Flags	Extractable

CKM_DES3_OFB

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	128
Maximum key length (bits)	192
Block size	8
Digest size	0
Key types	DES3
Algorithms	DES3
Modes	OFB
Flags	Extractable

CKM_DES3_X919_MAC

The CKM_DES3_X919_MAC is a signature generation and verification mechanism, as defined ANSI X9.19-1996 Financial Institution Retail Message Authentication annex 1 Cipher Block Chaining Procedure.

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	192
Block size	8
Digest size	0
Key types	DES3
Algorithms	DES3
Modes	MAC
Flags	Extractable

Usage

The CKM_DES3_X919_MAC mechanism is used with the **C_VerifyInit** and **C_SignInit** functions. It has the following attriobutes:

- Only CKK_DES2 and CKK_DES3 keys are supported.
- The mechanism takes no parameter.
- Multi-part operation is supported.
- The total input data length must be at least one byte.
- The length of result is half the size of the DES block (i.e. 4 bytes).

Example

```
#define CKM_DES3_X919_MAC (CKM_VENDOR_DEFINED + 0x150)

CK_OBJECT_HANDLE hKey; // handle of CKK_DES2 or CKK_DES3 key
CK_MECHANISM mech = { CKM_DES3_X919_MAC , NULL, 0};
CK_CHAR inp[any length];
CK_CHAR mac[4];
CK_SIZE len;

// Single-part operation
```



```
C_SignInit(hSes, &mech, hKey);
len = sizeof mac;
C_Sign(hSes, inp, sizeof inp, mac, &len);

// Multi-part operation

C_SignInit(hSes, &mech, hKey);
C_SignUpdate(hSes, inp, sizeof inp/2);
C_SignUpdate(hSes, inp+ (sizeof inp)/2, sizeof inp/2);
len = sizeof mac;
C_SignFinal(hSes, mac, &len);

// Test vectors

static const UInt8 retailKey[16] =
{
   0x58, 0x91, 0x25, 0x86, 0x3D, 0x46, 0x10, 0x7F,
   0x46, 0x3E, 0x52, 0x3B, 0xF7, 0x46, 0x9D, 0x52
};

static const UInt8 retailInputAscii[19] =
{
   't','h','e',' ','q','u','i','c','k',' ','b','r','o','w','n',' ','f','o','x'
};

static const UInt8 retailMACAscii[4] =
{
   0x55, 0xA7, 0xBF, 0xBA
};

static const UInt8 retailInputEBCDIC[19] =
{
   // "the quick brown fox" in EBCDIC
   0xA3, 0x88, 0x85, 0x40, 0x98, 0xA4, 0x89, 0x83,
   0x92, 0x40, 0x82, 0x99, 0x96, 0xA6, 0x95, 0x40,
   0x86, 0x96, 0xA7
};

static const UInt8 retailMACEBCDIC[4] =
{
   0x60, 0xAE, 0x2C, 0xD7
};
```

CKM_DH_PKCS_DERIVE

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	512
Minimum key length for FIPS use (bits)	512
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	0
Digest size	0
Key types	DH
Algorithms	None
Modes	None
Flags	None

CKM_DH_PKCS_KEY_PAIR_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key Pair
Minimum key length (bits)	512
Minimum key length for FIPS use (bits)	512
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	0
Digest size	0
Key types	DH
Algorithms	None
Modes	None
Flags	None

CKM_DH_PKCS_PARAMETER_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	512
Minimum key length for FIPS use (bits)	0
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	0
Digest size	0
Key types	DH
Algorithms	None
Modes	None
Flags	None

CKM_DSA

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	3072
Block size	0
Digest size	0
Key types	DSA
Algorithms	DSA
Modes	None
Flags	None

CKM_DSA_KEY_PAIR_GEN

Summary

FIPS approved?	Yes
Supported functions	Generate Key Pair
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	3072
Block size	0
Digest size	0
Key types	DSA
Algorithms	None
Modes	None
Flags	None

CKM_DSA_PARAMETER_GEN

Summary

FIPS approved?	Yes
Supported functions	Generate Key
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	3072
Block size	0
Digest size	0
Key types	DSA
Algorithms	None
Modes	None
Flags	None

CKM_EC_KEY_PAIR_GEN

Summary

FIPS approved?	Yes
Supported functions	Generate Key Pair
Minimum key length (bits)	105
Minimum key length for FIPS use (bits)	224
Minimum legacy key length for FIPS use (bits)	160
Maximum key length (bits)	571
Block size	0
Digest size	0
Key types	ECDSA
Algorithms	None
Modes	None
Flags	None

CKM_EC_KEY_PAIR_GEN_W_EXTRA_BITS

Summary

FIPS approved?	Yes
Supported functions	Generate Key Pair
Minimum key length (bits)	105
Minimum key length for FIPS use (bits)	224
Minimum legacy key length for FIPS use (bits)	160
Maximum key length (bits)	571
Block size	0
Digest size	0
Key types	ECDSA
Algorithms	None
Modes	None
Flags	ECC_EXTRA_BITS

CKM_ECDH1_COFACTOR_DERIVE

Summary

FIPS approved?	Yes
Supported functions	Derive
Minimum key length (bits)	105
Minimum key length for FIPS use (bits)	224
Minimum legacy key length for FIPS use (bits)	160
Maximum key length (bits)	571
Block size	0
Digest size	0
Key types	ECDSA
Algorithms	None
Modes	None
Flags	None

CKM_ECDH1_DERIVE

Elliptic Curve Diffie-Hellman is an anonymous key-agreement protocol. CKM_ECDH1_DERIVE is the derive function for that protocol.

Note: To enhance performance, we have created a proprietary call CA_DeriveKeyAndWrap, which is an optimization of C_DeriveKey with C_Wrap, merging the two functions into one (the in and out constraints are the same as for the individual functions). A further optimization is applied when mechanism CKM_ECDH1_DERIVE is used with CA_DeriveKeyAndWrap.

If CA_DeriveKeyAndWrap is called with other mechanisms, those would not be optimized.

Summary

FIPS approved?	Yes
Supported functions	Derive
Minimum key length (bits)	105
Minimum key length for FIPS use (bits)	224
Minimum legacy key length for FIPS use (bits)	160
Maximum key length (bits)	571
Block size	0
Digest size	0
Key types	ECDSA
Algorithms	None
Modes	None
Flags	None

CKM_ECDSA

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	105
Minimum key length for FIPS use (bits)	224
Minimum legacy key length for FIPS use (bits)	160
Maximum key length (bits)	571
Block size	0
Digest size	0
Key types	ECDSA
Algorithms	ECDSA
Modes	None
Flags	None

CKM_ECIES

Note: This is a single part operation, so even if it is called using multi-part API, we accumulate the data (up to a maximum) and return data only on the “final” operation. That is the meaning of "Accumulating" in the table, below.

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	105
Minimum key length for FIPS use (bits)	224
Minimum legacy key length for FIPS use (bits)	160
Maximum key length (bits)	571
Block size	0
Digest size	0
Key types	ECDSA
Algorithms	None
Modes	None
Flags	Accumulating

CKM_ECMQV_DERIVE

Summary

FIPS approved?	Yes
Supported functions	Derive
Minimum key length (bits)	105
Minimum key length for FIPS use (bits)	224
Minimum legacy key length for FIPS use (bits)	160
Maximum key length (bits)	571
Block size	0
Digest size	0
Key types	ECDSA
Algorithms	None
Modes	None
Flags	None

CKM_EXTRACT_KEY_FROM_KEY

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	Internal

CKM_GENERIC_SECRET_KEY_GEN

Summary

FIPS approved?	Yes
Supported functions	Generate Key
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	0
Digest size	0
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_HAS160

Summary

FIPS approved?	No
Supported functions	Digest
Minimum key length (bits)	0
Minimum key length for FIPS use (bits)	0
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	0
Block size	64
Digest size	20
Key types	None
Algorithms	HAS160
Modes	None
Flags	Extractable | Korean

CKM_HAS160_KCDSA

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	1024
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	64
Digest size	20
Key types	KCDSA
Algorithms	HAS160
Modes	None
Flags	Korean

CKM_HAS160_KCDSA_NO_PAD

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	1024
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	64
Digest size	20
Key types	KCDSA
Algorithms	HAS160
Modes	None
Flags	Korean

CKM_HMAC_HAS160

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	64
Digest size	20
Key types	Symmetric
Algorithms	HAS160
Modes	HMAC
Flags	Extractable | Korean | Internal

CKM_HMAC_MD5

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	64
Digest size	16
Key types	Symmetric
Algorithms	MD5
Modes	HMAC
Flags	Extractable

CKM_HMAC_MD5_80

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	64
Digest size	16
Key types	Symmetric
Algorithms	MD5
Modes	HMAC
Flags	Extractable | Internal

CKM_HMAC_RIPEMD160

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	64
Digest size	20
Key types	Symmetric
Algorithms	RIPEMD160
Modes	HMAC
Flags	Extractable | Internal

CKM_HMAC_SHA1

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	80
Maximum key length (bits)	4096
Block size	64
Digest size	20
Key types	Symmetric
Algorithms	SHA
Modes	HMAC
Flags	Extractable

CKM_HMAC_SHA1_80

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	80
Maximum key length (bits)	4096
Block size	64
Digest size	20
Key types	Symmetric
Algorithms	SHA
Modes	HMAC
Flags	Extractable

CKM_HMAC_SHA224

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	80
Maximum key length (bits)	4096
Block size	64
Digest size	28
Key types	Symmetric
Algorithms	SHA224
Modes	HMAC
Flags	Extractable

CKM_HMAC_SHA256

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	80
Maximum key length (bits)	4096
Block size	64
Digest size	32
Key types	Symmetric
Algorithms	SHA256
Modes	HMAC
Flags	Extractable

CKM_HMAC_SHA384

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	80
Maximum key length (bits)	4096
Block size	128
Digest size	48
Key types	Symmetric
Algorithms	SHA384
Modes	HMAC
Flags	Extractable

CKM_HMAC_SHA512

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	80
Maximum key length (bits)	4096
Block size	128
Digest size	64
Key types	Symmetric
Algorithms	SHA512
Modes	HMAC
Flags	Extractable

CKM_HMAC_SM3

SM3 is a hash function published by the Chinese Commercial Cryptography Administration Office for the use of electronic authentication service system. The design of SM3 builds upon the design of the SHA-2 hash function, but introduces additional strengthening features. For SafeNet HSMs, the available mechanisms are CKM_SM3, the hash function, and CKM_SM3_KEY_DERIVATION, and CKM_HMAC_SM3.

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	8
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	64
Digest size	32
Key types	Symmetric
Algorithms	SM3
Modes	HMAC
Flags	None

CKM_KCDSA_KEY_PAIR_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key Pair
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	1024
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	0
Digest size	0
Key types	KCDSA
Algorithms	None
Modes	None
Flags	Korean

CKM_KCDSA_PARAMETER_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	1024
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	0
Digest size	0
Key types	KCDSA
Algorithms	None
Modes	None
Flags	Korean

CKM_KEY_WRAP_SET_OAEP

Summary

FIPS approved?	No
Supported functions	Wrap | Unwrap
Minimum key length (bits)	256
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	8192
Block size	0
Digest size	0
Key types	RSA
Algorithms	None
Modes	None
Flags	None

CKM_LOOP_BACK

Summary

FIPS approved?	No
Supported functions	None
Minimum key length (bits)	0
Minimum key length for FIPS use (bits)	0
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	0
Block size	0
Digest size	0
Key types	None
Algorithms	None
Modes	None
Flags	Not Listed

CKM_LZS

Summary

FIPS approved?	No
Supported functions	None
Minimum key length (bits)	0
Minimum key length for FIPS use (bits)	0
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	0
Block size	0
Digest size	0
Key types	None
Algorithms	None
Modes	None
Flags	Not Listed

CKM_MD2

Summary

FIPS approved?	No
Supported functions	Digest
Minimum key length (bits)	0
Minimum key length for FIPS use (bits)	0
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	0
Block size	16
Digest size	16
Key types	None
Algorithms	MD2
Modes	None
Flags	Extractable

CKM_MD2_DES_CBC

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	16
Digest size	16
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_MD2_KEY_DERIVATION

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	16
Digest size	16
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

CKM_MD5

Summary

FIPS approved?	No
Supported functions	Digest
Minimum key length (bits)	0
Minimum key length for FIPS use (bits)	0
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	0
Block size	64
Digest size	16
Key types	None
Algorithms	MD5
Modes	None
Flags	Extractable | Internal

CKM_MD5_CAST_CBC

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	64
Digest size	16
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_MD5_CAST3_CBC

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	64
Digest size	16
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_MD5_DES_CBC

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	64
Block size	64
Digest size	16
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_MD5_KEY_DERIVATION

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	64
Digest size	16
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

CKM_MD5_RSA_PKCS

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	256
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	8192
Block size	64
Digest size	16
Key types	RSA
Algorithms	MD5
Modes	None
Flags	Extractable | Internal

CKM_NIST_PRF_KDF

Summary

FIPS approved?	Yes
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

Usage

The CKM_NIST_PRF_KDF mechanism only supports counter mode. CKM_NIST_PRF_KDF is always allowed. It does not matter if the “allow non-FIPS approved algorithms” HSM policy is on or off. This mechanism can only be used with DES3_CMAC or AES_CMAC as the PRF.

The SP 800-108 allows for some variation on what/how the information is encoded and describes some fields as optional. To accommodate that, there are two encoding schemes you can specify:

- LUNA_PRF_KDF_ENCODING_SCHEME_2: the separator byte and the length of the derived key are not encoded in the input data for the PRF.
- LUNA_PRF_KDF_ENCODING_SCHEME_1: both fields are included.

Example

```
/* Parameter and values used with CKM_PRF_KDF and CKM_NIST_PRF_KDF. */
typedef CK_ULONG CK_KDF_PRF_TYPE;
typedef CK_ULONG CK_KDF_PRF_ENCODING_SCHEME;
/** PRF KDF schemes */
#define CK_NIST_PRF_KDF_DES3_CMAC      0x00000001
#define CK_NIST_PRF_KDF_AES_CMAC       0x00000002
#define CK_PRF_KDF_ARIA_CMAC           0x00000003
#define CK_PRF_KDF_SEED_CMAC           0x00000004
#define LUNA_PRF_KDF_ENCODING_SCHEME_1     0x00000000
#define LUNA_PRF_KDF_ENCODING_SCHEME_2     0x00000001
typedef struct CK_KDF_PRF_PARAMS {
CK_KDF_PRF_TYPE            prfType;
```



```
CK_BYTE_PTR                pLabel;
CK_ULONG                   ulLabelLen;
CK_BYTE_PTR                pContext;
CK_ULONG                   ulContextLen;
CK_ULONG                   ulCounter;
CK_KDF_PRF_ENCODING_SCHEME ulEncodingScheme;
} CK_PRF_KDF_PARAMS;
typedef CK_PRF_KDF_PARAMS CK_PTR CK_KDF_PRF_PARAMS_PTR;
```


CKM_PKCS5_PBKD2

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	8
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	8
Block size	0
Digest size	0
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_PRF_KDF

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

Usage

The CKM_NIST_PRF mechanism only supports counter mode. CKM_PRF_KDF is only allowed when the “allow non-FIPS approved algorithms” HSM policy is on. This mechanism can be used with DES3_CMAC, AES_CMAC, ARIA_CMAC or SEED_CMAC as the PRF.

The SP 800-108 allows for some variation on what/how the information is encoded and describes some fields as optional. To accommodate that, there are two encoding schemes you can specify:

- LUNA_PRF_KDF_ENCODING_SCHEME_2: the separator byte and the length of the derived key are not encoded in the input data for the PRF.
- LUNA_PRF_KDF_ENCODING_SCHEME_1: both fields are included.

Example

```
/* Parameter and values used with CKM_PRF_KDF and CKM_NIST_PRF_KDF. */
typedef CK_ULONG CK_KDF_PRF_TYPE;
typedef CK_ULONG CK_KDF_PRF_ENCODING_SCHEME;
/** PRF KDF schemes */
#define CK_NIST_PRF_KDF_DES3_CMAC      0x00000001
#define CK_NIST_PRF_KDF_AES_CMAC       0x00000002
#define CK_PRF_KDF_ARIA_CMAC           0x00000003
#define CK_PRF_KDF_SEED_CMAC           0x00000004
#define LUNA_PRF_KDF_ENCODING_SCHEME_1     0x00000000
#define LUNA_PRF_KDF_ENCODING_SCHEME_2     0x00000001
typedef struct CK_KDF_PRF_PARAMS {
CK_KDF_PRF_TYPE            prfType;
```



```
CK_BYTE_PTR                pLabel;
CK_ULONG                   ulLabelLen;
CK_BYTE_PTR                pContext;
CK_ULONG                   ulContextLen;
CK_ULONG                   ulCounter;
CK_KDF_PRF_ENCODING_SCHEME ulEncodingScheme;
} CK_PRF_KDF_PARAMS;
typedef CK_PRF_KDF_PARAMS CK_PTR CK_KDF_PRF_PARAMS_PTR;
```


CKM_RC2_CBC

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	1024
Block size	8
Digest size	0
Key types	RC2
Algorithms	RC2
Modes	CBC
Flags	Extractable

CKM_RC2_CBC_PAD

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	1024
Block size	8
Digest size	0
Key types	RC2
Algorithms	RC2
Modes	CBC_PAD
Flags	Extractable

CKM_RC2_ECB

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	1024
Block size	8
Digest size	0
Key types	RC2
Algorithms	RC2
Modes	ECB
Flags	Extractable

CKM_RC2_KEY_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	1024
Block size	0
Digest size	0
Key types	RC2
Algorithms	None
Modes	None
Flags	None

CKM_RC2_MAC

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	1024
Block size	8
Digest size	0
Key types	RC2
Algorithms	RC2
Modes	MAC
Flags	Extractable

CKM_RC4

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	0
Digest size	0
Key types	RC4
Algorithms	RC4
Modes	STREAM
Flags	Extractable

CKM_RC4_KEY_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	0
Digest size	0
Key types	RC4
Algorithms	None
Modes	None
Flags	None

CKM_RC5_CBC

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2040
Block size	8
Digest size	0
Key types	RC5
Algorithms	RC5
Modes	CBC
Flags	Extractable

CKM_RC5_CBC_PAD

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2040
Block size	8
Digest size	0
Key types	RC5
Algorithms	RC5
Modes	CBC_PAD
Flags	Extractable

CKM_RC5_ECB

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2040
Block size	8
Digest size	0
Key types	RC5
Algorithms	RC5
Modes	ECB
Flags	Extractable

CKM_RC5_KEY_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2040
Block size	0
Digest size	0
Key types	RC5
Algorithms	None
Modes	None
Flags	None

CKM_RC5_MAC

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2040
Block size	8
Digest size	0
Key types	RC5
Algorithms	RC5
Modes	MAC
Flags	Extractable

CKM_RIPEMD160

Summary

FIPS approved?	No
Supported functions	Digest
Minimum key length (bits)	0
Minimum key length for FIPS use (bits)	0
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	0
Block size	64
Digest size	20
Key types	None
Algorithms	RIPEMD160
Modes	None
Flags	Extractable | Internal

CKM_RSA_FIPS_186_3_AUX_PRIME_KEY_PAIR_GEN

Summary

FIPS approved?	Yes
Supported functions	Generate Key Pair
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	4096
Block size	0
Digest size	0
Key types	RSA
Algorithms	None
Modes	None
Flags	None

CKM_RSA_FIPS_186_3_PRIME_KEY_PAIR_GEN

Summary

FIPS approved?	Yes
Supported functions	Generate Key Pair
Minimum key length (bits)	2048
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	0
Digest size	0
Key types	RSA
Algorithms	None
Modes	None
Flags	None

CKM_RSA_PKCS

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify | Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	256
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	0
Digest size	0
Key types	RSA
Algorithms	None
Modes	None
Flags	None

CKM_RSA_PKCS_KEY_PAIR_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key Pair
Minimum key length (bits)	256
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	0
Digest size	0
Key types	RSA
Algorithms	None
Modes	None
Flags	None

CKM_RSA_PKCS_OAEP

The RSA PKCS OAEP mechanism can now use a supplied hashing mechanism. Previously RSA OAEP would always use SHA1 and returned an error if another was attempted.

With current firmware, PKCS#11 API and ckdemo now accept a new mechanism.

Allowed mechanisms are:

CKM_SHA1

CKM_SHA224

CKM_SHA256

CKM_SHA384

CKM_SHA512

0 (use the firmware's default engine, which is currently SHA1)

In ckdemo menu option 98 has a new value 17 - OAEP Hash Params, which can be set to use either default (CKM_SHA1) or selectable. When it is set to selectable the user is prompted for a hash mechanism when using the OAEP mechanism.

Summary

FIPS approved?	Yes
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	256
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	0
Digest size	0
Key types	RSA
Algorithms	None
Modes	None
Flags	None

CKM_RSA_PKCS_PSS

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	256
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	0
Digest size	0
Key types	RSA
Algorithms	None
Modes	None
Flags	None | PSS

CKM_RSA_X_509

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	256
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	8192
Block size	0
Digest size	0
Key types	RSA
Algorithms	None
Modes	None
Flags	None

CKM_RSA_X9_31

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	0
Digest size	0
Key types	RSA
Algorithms	None
Modes	None
Flags	Extractable | X9.31

CKM_RSA_X9_31_KEY_PAIR_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key Pair
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	0
Digest size	0
Key types	RSA
Algorithms	None
Modes	None
Flags	X9.31

CKM_RSA_X9_31_NON_FIPS

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	8192
Block size	0
Digest size	0
Key types	RSA
Algorithms	None
Modes	None
Flags	Extractable | X9.31 | Non-FIPS X9.31

CKM_SEED_CBC

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	16
Digest size	0
Key types	SEED
Algorithms	SEED
Modes	CBC
Flags	Extractable | Korean

CKM_SEED_CBC_PAD

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	16
Digest size	0
Key types	SEED
Algorithms	SEED
Modes	CBC_PAD
Flags	Extractable | Korean

CKM_SEED_CMAC

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	16
Digest size	0
Key types	SEED
Algorithms	SEED
Modes	MAC
Flags	Extractable | Korean | CMAC

CKM_SEED_CTR

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	16
Digest size	0
Key types	SEED
Algorithms	SEED
Modes	CTR
Flags	Extractable | Korean

CKM_SEED_ECB

Summary

FIPS approved?	No
Supported functions	Encrypt | Decrypt | Wrap | Unwrap
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	16
Digest size	0
Key types	SEED
Algorithms	SEED
Modes	ECB
Flags	Extractable | Korean

CKM_SEED_KEY_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	0
Digest size	0
Key types	SEED
Algorithms	None
Modes	None
Flags	Korean

CKM_SEED_MAC

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	16
Digest size	0
Key types	SEED
Algorithms	SEED
Modes	MAC
Flags	Extractable | Korean

CKM_SHA_1

Summary

FIPS approved?	Yes
Supported functions	Digest
Minimum key length (bits)	0
Minimum key length for FIPS use (bits)	0
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	0
Block size	64
Digest size	20
Key types	None
Algorithms	SHA
Modes	None
Flags	Extractable

CKM_SHA1_CAST5_CBC

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	64
Minimum key length for FIPS use (bits)	64
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	64
Digest size	20
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_SHA1_DES2_CBC

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	64
Digest size	20
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_SHA1_DES2_CBC_OLD

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	64
Digest size	20
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_SHA1_DES3_CBC

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	192
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	192
Block size	64
Digest size	20
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_SHA1_DES3_CBC_OLD

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	192
Minimum key length for FIPS use (bits)	192
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	192
Block size	64
Digest size	20
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_SHA1_DSA

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	3072
Block size	64
Digest size	20
Key types	DSA
Algorithms	SHA
Modes	None
Flags	Extractable

CKM_SHA1_ECDSA

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	105
Minimum key length for FIPS use (bits)	224
Minimum legacy key length for FIPS use (bits)	160
Maximum key length (bits)	571
Block size	64
Digest size	20
Key types	ECDSA
Algorithms	SHA
Modes	None
Flags	Extractable

CKM_SHA1_KCDSA

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	1024
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	64
Digest size	20
Key types	KCDSA
Algorithms	SHA
Modes	None
Flags	Korean

CKM_SHA1_KCDSA_NO_PAD

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	1024
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	64
Digest size	20
Key types	KCDSA
Algorithms	SHA
Modes	None
Flags	Korean

CKM_SHA1_KEY_DERIVATION

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	64
Digest size	20
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

CKM_SHA1_RC2_40_CBC

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	40
Minimum key length for FIPS use (bits)	40
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	40
Block size	64
Digest size	20
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_SHA1_RC2_128_CBC

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	64
Digest size	20
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_SHA1_RC4_40

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	40
Minimum key length for FIPS use (bits)	40
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	40
Block size	64
Digest size	20
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_SHA1_RC4_128

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	64
Digest size	20
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_SHA1_RSA_PKCS

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	256
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	64
Digest size	20
Key types	RSA
Algorithms	SHA
Modes	None
Flags	Extractable

CKM_SHA1_RSA_PKCS_PSS

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	256
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	64
Digest size	20
Key types	RSA
Algorithms	SHA
Modes	None
Flags	Extractable | PSS

CKM_SHA1_RSA_X9_31

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	64
Digest size	20
Key types	RSA
Algorithms	SHA
Modes	None
Flags	Extractable | X9.31

CKM_SHA1_RSA_X9_31_NON_FIPS

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	8192
Block size	64
Digest size	20
Key types	RSA
Algorithms	SHA
Modes	None
Flags	Extractable | X9.31 | Non-FIPS X9.31

CKM_SHA224

Summary

FIPS approved?	Yes
Supported functions	Digest
Minimum key length (bits)	0
Minimum key length for FIPS use (bits)	0
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	0
Block size	64
Digest size	28
Key types	None
Algorithms	SHA224
Modes	None
Flags	Extractable

CKM_SHA224_DSA

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	3072
Block size	64
Digest size	28
Key types	DSA
Algorithms	SHA224
Modes	None
Flags	Extractable

CKM_SHA224_ECDSA

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	105
Minimum key length for FIPS use (bits)	224
Minimum legacy key length for FIPS use (bits)	160
Maximum key length (bits)	571
Block size	64
Digest size	28
Key types	ECDSA
Algorithms	SHA224
Modes	None
Flags	Extractable

CKM_SHA224_KCDSA

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	1024
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	64
Digest size	28
Key types	KCDSA
Algorithms	SHA224
Modes	None
Flags	Korean

CKM_SHA224_KCDSA_NO_PAD

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	1024
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	64
Digest size	28
Key types	KCDSA
Algorithms	SHA224
Modes	None
Flags	Korean

CKM_SHA224_KEY_DERIVATION

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	64
Digest size	28
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

CKM_SHA224_RSA_PKCS

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	256
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	64
Digest size	28
Key types	RSA
Algorithms	SHA224
Modes	None
Flags	Extractable

CKM_SHA224_RSA_PKCS_PSS

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	512
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	64
Digest size	28
Key types	RSA
Algorithms	SHA224
Modes	None
Flags	Extractable | PSS

CKM_SHA224_RSA_X9_31

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	64
Digest size	28
Key types	RSA
Algorithms	SHA224
Modes	None
Flags	Extractable | X9.31

CKM_SHA224_RSA_X9_31_NON_FIPS

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	8192
Block size	64
Digest size	28
Key types	RSA
Algorithms	SHA224
Modes	None
Flags	Extractable | X9.31 | Non-FIPS X9.31

CKM_SHA256

Summary

FIPS approved?	Yes
Supported functions	Digest
Minimum key length (bits)	0
Minimum key length for FIPS use (bits)	0
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	0
Block size	64
Digest size	32
Key types	None
Algorithms	SHA256
Modes	None
Flags	Extractable

CKM_SHA256_DSA

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	3072
Block size	64
Digest size	32
Key types	DSA
Algorithms	SHA256
Modes	None
Flags	Extractable

CKM_SHA256_ECDSA

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	105
Minimum key length for FIPS use (bits)	224
Minimum legacy key length for FIPS use (bits)	160
Maximum key length (bits)	571
Block size	64
Digest size	32
Key types	ECDSA
Algorithms	SHA256
Modes	None
Flags	Extractable

CKM_SHA256_ECDSA_GBCS

GBCS is the Great Britain Companion Specification, a component of the Smart Metering Equipment Technical Specification, in support of the Smart Metering Programme. SHA256withECDSAGBCS is a proprietary ECDSA signature algorithm defined by the GBCS standard. It does not appear to be congruent with any of the other Deterministic ECDSA algorithms available in the various published RFCs (at time of writing this comment). As well (at time of writing) this algorithm is currently not FIPS compliant. SHA256withECDSAGBCS was implemented specifically for GBCS integration. If you need to be compliant with GBCS then you must use SHA256withECDSAGBCS.

Otherwise, SHA256withECDSA is the standard ECDSA algorithm defined by most other standards (for example FIPS, X9).

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	256
Minimum key length for FIPS use (bits)	256
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	256
Block size	64
Digest size	32
Key types	ECDSA
Algorithms	SHA256
Modes	None
Flags	Extractable

CKM_SHA256_KCDSA

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	1024
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	64
Digest size	32
Key types	KCDSA
Algorithms	SHA256
Modes	None
Flags	Korean

CKM_SHA256_KCDSA_NO_PAD

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	1024
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	64
Digest size	32
Key types	KCDSA
Algorithms	SHA256
Modes	None
Flags	Korean

CKM_SHA256_KEY_DERIVATION

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	64
Digest size	32
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

CKM_SHA256_RSA_PKCS

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	256
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	64
Digest size	32
Key types	RSA
Algorithms	SHA256
Modes	None
Flags	Extractable

CKM_SHA256_RSA_PKCS_PSS

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	512
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	64
Digest size	32
Key types	RSA
Algorithms	SHA256
Modes	None
Flags	Extractable | PSS

CKM_SHA256_RSA_X9_31

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	64
Digest size	32
Key types	RSA
Algorithms	SHA256
Modes	None
Flags	Extractable | X9.31

CKM_SHA256_RSA_X9_31_NON_FIPS

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	8192
Block size	64
Digest size	32
Key types	RSA
Algorithms	SHA256
Modes	None
Flags	Extractable | X9.31 | Non-FIPS X9.31

CKM_SHA384

Summary

FIPS approved?	Yes
Supported functions	Digest
Minimum key length (bits)	0
Minimum key length for FIPS use (bits)	0
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	0
Block size	128
Digest size	48
Key types	None
Algorithms	SHA384
Modes	None
Flags	Extractable

CKM_SHA384_ECDSA

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	105
Minimum key length for FIPS use (bits)	224
Minimum legacy key length for FIPS use (bits)	160
Maximum key length (bits)	571
Block size	128
Digest size	48
Key types	ECDSA
Algorithms	SHA384
Modes	None
Flags	Extractable

CKM_SHA384_KCDSA

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	1024
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	128
Digest size	48
Key types	KCDSA
Algorithms	SHA384
Modes	None
Flags	Korean

CKM_SHA384_KCDSA_NO_PAD

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	1024
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	128
Digest size	48
Key types	KCDSA
Algorithms	SHA384
Modes	None
Flags	Korean

CKM_SHA384_KEY_DERIVATION

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	128
Digest size	48
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

CKM_SHA384_RSA_PKCS

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	256
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	128
Digest size	48
Key types	RSA
Algorithms	SHA384
Modes	None
Flags	Extractable

CKM_SHA384_RSA_PKCS_PSS

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	512
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	128
Digest size	48
Key types	RSA
Algorithms	SHA384
Modes	None
Flags	Extractable | PSS

CKM_SHA384_RSA_X9_31

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	128
Digest size	48
Key types	RSA
Algorithms	SHA384
Modes	None
Flags	Extractable | X9.31

CKM_SHA384_RSA_X9_31_NON_FIPS

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	8192
Block size	128
Digest size	48
Key types	RSA
Algorithms	SHA384
Modes	None
Flags	Extractable | X9.31 | Non-FIPS X9.31

CKM_SHA512

Summary

FIPS approved?	Yes
Supported functions	Digest
Minimum key length (bits)	0
Minimum key length for FIPS use (bits)	0
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	0
Block size	128
Digest size	64
Key types	None
Algorithms	SHA512
Modes	None
Flags	Extractable

CKM_SHA512_ECDSA

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	105
Minimum key length for FIPS use (bits)	224
Minimum legacy key length for FIPS use (bits)	160
Maximum key length (bits)	571
Block size	128
Digest size	64
Key types	ECDSA
Algorithms	SHA512
Modes	None
Flags	Extractable

CKM_SHA512_KCDSA

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	1024
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	128
Digest size	64
Key types	KCDSA
Algorithms	SHA512
Modes	None
Flags	Korean

CKM_SHA512_KCDSA_NO_PAD

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	1024
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	2048
Block size	128
Digest size	64
Key types	KCDSA
Algorithms	SHA512
Modes	None
Flags	Korean

CKM_SHA512_KEY_DERIVATION

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	112
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	128
Digest size	64
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

CKM_SHA512_RSA_PKCS

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	256
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	128
Digest size	64
Key types	RSA
Algorithms	SHA512
Modes	None
Flags	Extractable

CKM_SHA512_RSA_PKCS_PSS

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	128
Digest size	64
Key types	RSA
Algorithms	SHA512
Modes	None
Flags	Extractable | PSS

CKM_SHA512_RSA_X9_31

Summary

FIPS approved?	Yes
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	1024
Maximum key length (bits)	8192
Block size	128
Digest size	64
Key types	RSA
Algorithms	SHA512
Modes	None
Flags	Extractable | X9.31

CKM_SHA512_RSA_X9_31_NON_FIPS

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	8192
Block size	128
Digest size	64
Key types	RSA
Algorithms	SHA512
Modes	None
Flags	Extractable | X9.31 | Non-FIPS X9.31

CKM_SM3

SM3 is a hash function published by the Chinese Commercial Cryptography Administration Office for the use of electronic authentication service system. The design of SM3 builds upon the design of the SHA-2 hash function, but introduces additional strengthening features. For SafeNet HSMs, the available mechanisms are CKM_SM3, the hash function, and CKM_SM3_KEY_DERIVATION, and CKM_HMAC_SM3.

Summary

FIPS approved?	No
Supported functions	Digest
Minimum key length (bits)	0
Minimum key length for FIPS use (bits)	0
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	0
Block size	64
Digest size	32
Key types	None
Algorithms	SM3
Modes	None
Flags	None

CKM_SM3_KEY_DERIVATION

SM3 is a hash function published by the Chinese Commercial Cryptography Administration Office for the use of electronic authentication service system. The design of SM3 builds upon the design of the SHA-2 hash function, but introduces additional strengthening features. For SafeNet HSMs, the available mechanisms are CKM_SM3, the hash function, and CKM_SM3_KEY_DERIVATION, and CKM_HMAC_SM3.

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	8
Minimum key length for FIPS use (bits)	8
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	64
Digest size	32
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

CKM_SSL3_KEY_AND_MAC_DERIVE

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	384
Minimum key length for FIPS use (bits)	384
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	384
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

CKM_SSL3_MASTER_KEY_DERIVE

Summary

FIPS approved?	No
Supported functions	Derive
Minimum key length (bits)	384
Minimum key length for FIPS use (bits)	384
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	384
Block size	0
Digest size	0
Key types	Symmetric
Algorithms	None
Modes	None
Flags	None

CKM_SSL3_MD5_MAC

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	128
Minimum key length for FIPS use (bits)	128
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	128
Block size	64
Digest size	16
Key types	Symmetric
Algorithms	MD5
Modes	HMAC
Flags	Extractable

CKM_SSL3_PRE_MASTER_KEY_GEN

Summary

FIPS approved?	No
Supported functions	Generate Key
Minimum key length (bits)	384
Minimum key length for FIPS use (bits)	384
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	384
Block size	0
Digest size	0
Key types	None
Algorithms	None
Modes	None
Flags	None

CKM_SSL3_SHA1_MAC

Summary

FIPS approved?	No
Supported functions	Sign | Verify
Minimum key length (bits)	160
Minimum key length for FIPS use (bits)	160
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	160
Block size	64
Digest size	20
Key types	Symmetric
Algorithms	SHA
Modes	HMAC
Flags	Extractable

CKM_UNKNOWN

Summary

FIPS approved?	No
Supported functions	None
Minimum key length (bits)	0
Minimum key length for FIPS use (bits)	0
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	0
Block size	0
Digest size	0
Key types	None
Algorithms	None
Modes	None
Flags	Not Listed

CKM_X9_42_DH_DERIVE

Summary

FIPS approved?	Yes
Supported functions	Derive
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	0
Digest size	0
Key types	X9_42_DH
Algorithms	None
Modes	None
Flags	None

CKM_X9_42_DH_HYBRID_DERIVE

Summary

FIPS approved?	Yes
Supported functions	Derive
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	0
Digest size	0
Key types	X9_42_DH
Algorithms	None
Modes	None
Flags	None

CKM_X9_42_DH_KEY_PAIR_GEN

Summary

FIPS approved?	Yes
Supported functions	Generate Key Pair
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	0
Digest size	0
Key types	X9_42_DH
Algorithms	None
Modes	None
Flags	None

CKM_X9_42_DH_PARAMETER_GEN

Summary

FIPS approved?	Yes
Supported functions	Generate Key
Minimum key length (bits)	1024
Minimum key length for FIPS use (bits)	2048
Minimum legacy key length for FIPS use (bits)	N/A
Maximum key length (bits)	4096
Block size	0
Digest size	0
Key types	X9_42_DH
Algorithms	None
Modes	None
Flags	None

5
Using the SafeNet SDK

This chapter describes how to use the SDK to develop applications that exercise the HSM. It contains the following topics:

Libraries and Applications

This section explains how to make the Chrystoki library available to the other components of the SafeNet Software Development Kit.

An application has no knowledge of which library is to be loaded nor does the application know the library's location. For these reasons, a special scheme must be employed to tell the application, while it is running, where to find the library. The next paragraphs describe how applications connect to Chrystoki.

SafeNet SDK Applications General Information

All applications provided in SafeNet Network HSM Software Development Kit have been compiled with a component called CkBridge, which uses a configuration file to find the library it is intended to load. Ckbridge first uses the environment variable "ChrystokiConfigurationPath" to locate the corresponding configuration file. If this environment variable is not set, it attempts to locate the configuration file in a default location depending on the product platform (/etc on Unix, and c:\Program Files\SafeNet\LunaClient on Windows).

Configuration files differ from one platform to the next - refer to the appropriate sub-section for the operating system and syntax applicable to your development platform.

Windows

In Windows, an initialization file called **crystoki.ini** specifies which library is to be loaded. The syntax of this file is standard to Windows.

The following example shows proper configuration files for Windows:

```
[Chrystoki2]
LibNT=C:\Program Files\SafeNet\LunaClient\cryptoki.dll
[LunaSA Client]
SSLConfigFile=C:\Program Files\SafeNet\LunaClient\openssl.cnf
ReceiveTimeout=20000
NetClient=1
ServerCAFile=C:\Program Files\SafeNet\LunaClient\cert\server\CAFile.pem
ClientCertFile=C:\Program Files\SafeNet\LunaClient\cert\client\ClientNameCert.pem
```

```
ClientPrivKeyFile=C:\Program Files\SafeNet\LunaClient\cert\client\ClientNameKey.pem
[Luna]
DefaultTimeOut=500000
PEDTimeout1=100000
PEDTimeout2=200000
PEDTimeout3=10000
[CardReader]
RemoteCommand=1
```

CAUTION: NEVER insert TAB characters into the crystoki.ini (Windows) or chrystoki.conf (UNIX) file.

UNIX

In UNIX, a configuration file called "Chrystoki.conf" is used to guide CkBridge in finding the appropriate library.

The configuration file is a regular text file with a special format. It is made up of a number of sections, each section containing one or multiple entries. The following example shows a typical UNIX configuration file:

```
Chrystoki2 = {
LibUNIX=/usr/lib/libCryptoki2.so;
}
Luna = {
DefaultTimeOut=500000;
PEDTimeout1=100000;
PEDTimeout2=200000;
PEDTimeout3=10000;
KeypairGenTimeOut=2700000;
CloningCommandTimeOut=300000;
}
CardReader = {
RemoteCommand=1;
}
LunaSA Client = {
NetClient = 1;
ServerCAFile = /usr/safenet/lunaclient/cert/server/CAFile.pem;
ClientCertFile = /usr/safenet/lunaclient/cert/client/ClientNameCert.pem;
ClientPrivKeyFile = /usr/safenet/lunaclient/cert/client/ClientNameKey.pem;
SSLConfigFile = /usr/safenet/lunaclient/bin/openssl.cnf;
ReceiveTimeout = 20000;
}
```

The shared object "libcrystoki2.so" is a library supporting version 2.2.0 of the PKCS#11 standard.

CAUTION: NEVER insert TAB characters into the chrystoki.ini (Windows) or crystoki.conf (UNIX) file.

Compiler Tools

Tools used for SafeNet development are platform specific tools/development environments, where applicable (e.g., Visual C++ on Windows 2008 and Windows 2012, or Workshop on Solaris, or aCC on HP-UX). Current version information is provided in the Customer Release Notes.

Note: Contact SafeNet for information about the availability of newer versions of compilers.

The Applications

See the "About the Utilities Reference Guide" on page 1 for information about individual tools and utilities, provided for use with SafeNet HSMs.

Named Curves and User-Defined Parameters

The SafeNet HSM is a PKCS#11 oriented device. Prior to firmware 4.6.7, the HSM firmware statically defined the NIST named curve OIDs and curve parameters. To expand on that capability and add flexibility, firmware 4.6.7 (SafeNet Network HSM 4.3) and later added support for Brainpool curve OIDs and curve parameters. Additional support was added to decode the ecParameters structure and use that data in the generation of keys as well as in signing and verification.

Curve Validation Limitations

The HSM can validate the curve parameters, however domain parameter validation guarantees only the consistency/sanity of the parameters and the most basic, well-known security properties. The HSM has no way of judging the quality of a user-specified curve.

It is therefore important that you perform Known Answer Tests to verify the operation of the HSM for any new Domain Parameter.set. To maintain NIST-FIPS compatibility the feature is selectively enabled with the feature being disabled by default. Therefore the Administrator must 'opt-in' by actively choosing to enable the appropriate HSM policy. Among other effects, this causes the HSM to display a shell/console message to the effect that the HSM is not operating in FIPS mode.

Storing Domain Parameters

Under PKCS#11 v2.20, Domain Parameters are stored in object attribute CKA_EC_PARAMS. The value of this parameter is the DER encoding of an ANSI X9.62 Parameters value.

```
Parameters ::= CHOICE {
ecParameters ECParameters,
namedCurve CURVES.&id({CurveNames}),
implicitlyCA NULL
}
```

Because PKCS#11 states that the implicitlyCA is not supported by cryptoki, therefore the CKA_EC_PARAMS attribute must contain the encoding of an ecParameters or a namedCurve. Cryptoki holds ECC key pairs in separate Private and Public key objects. Each object has its own CKA_EC_PARAMS attribute which must be provided when the object is created and cannot be subsequently changed.

Cryptoki also supports CKO_DOMAIN_PARAMETERS objects. These store Domain Parameters but perform no cryptographic operations. A Domain Parameters object is really only for storage. To generate a key pair, you must extract the attributes from the Domain Parameters object and insert them in the CKA_EC_PARAMS attribute of the Public key template. Cryptoki can create new ECC Public and Private key objects and Domain Parameters objects in the following ways:

- Objects can be directly stored using the C_CreateObject command
- Public and private key objects can be generated internally with the C_GenerateKeyPair command and the CKM_EC_KEY_PAIR_GEN mechanism.
- Objects can be imported in encrypted form using C_UnwrapKey command.

Using Domain Parameters

ECC keys may be used for Signature Generation and Verification with the CKM_ECDSA and CKM_ECDSA_SHA1 mechanism and Encryption and Decryption with the CKM_ECIES mechanism. These three mechanism are enhanced so that they fetch the Domain Parameters from the CKA_EC_PARAMS attribute for both ecParameters and namedCurve choice and not just namedCurve choice.

User Friendly Encoder

Using ECC with Cryptoki to create or generate ECC keys requires that the CKA_EC_PARAMS attribute be specified. This is a DER encoded binary array. Usually in public documents describing ECC curves the Domain Parameters are specified as a series of printable strings so the programmer faces the problem of converting these to the correct format for Cryptoki use.

The cryptoki library is extended to support functions called CA_EncodeECPrimeParams and CA_ EncodeECChar2Params which allow an application to specify the parameter details of a new curve. These functions implement DER encoders to build the CKA_EC_PARAMS attribute from large integer presentations of the Domain Parameter values.

Refer to "Sample Domain Parameter Files " on page 299 for some sample Domain Parameter files.

Application Interfaces

CA_EncodeECPrimeParams

#include “cryptoki.h”

CK_RV CA_ EncodeECPrimeParams (

CK_BYTE_PTR DerECParams, CK_ULONG_PTR DerECParams Len

CK_BYTE_PTR prime, CK_USHORT primelen,

CK_BYTE_PTR a, CK_USHORT alen,

CK_BYTE_PTR b, CK_USHORT blen,

CK_BYTE_PTR seed, CK_USHORT seedlen,

CK_BYTE_PTR x, CK_USHORT xlen,

CK_BYTE_PTR y, CK_USHORT ylen,

CK_BYTE_PTR order, CK_USHORT orderlen,

CK_BYTE_PTR cofactor, CK_USHORT cofactorlen,

);

Do DER enc of ECC Domain Parameters Prime

Parameters

DerECParams	Resultant Encoding (length prediction supported if NULL).
DerECParamsLen	Buffer len/Length of resultant encoding
prime	Prime modulus

primelen	Prime modulus len
a	Elliptic Curve coefficient a
alen	Elliptic Curve coefficient a length
b	Elliptic Curve coefficient b
blen	Elliptic Curve coefficient b length
seed	Seed (optional may be NULL)
seedlen	Seed length
x	Elliptic Curve point X coord
xlen	Elliptic Curve point X coord length
y	Elliptic Curve point Y coord
ylen	Elliptic Curve point Y coord length
order	Order n of the Base Point
orderlen	Order n of the Base Point length
cofactor	The integer h = #E(Fq)/n (optional)
cofactorlen	h length
Return	Status of operation. CKR_OK if ok.

CA_EncodeECChar2Params

```
#include “cryptoki.h”
CK_RV CA_EncodeECChar2Params(
        CK_BYTE_PTR DerECParams, CK_ULONG_PTR DerECParams Len
        CK_USHORT m,
         CK_USHORT k1,
         CK_USHORT k2,
        CK_USHORT k3,
        CK_BYTE_PTR a, CK_USHORT alen,
         CK_BYTE_PTR b, CK_USHORT blen,
        CK_BYTE_PTR seed, CK_USHORT seedlen,
         CK_BYTE_PTR x, CK_USHORT xlen,
        CK_BYTE_PTR y, CK_USHORT ylen,
        CK_BYTE_PTR order, CK_USHORT orderlen,
        CK_BYTE_PTR cofactor, CK_USHORT cofactorlen,
        );
```


Do DER enc of ECC Domain Parameters 2^M

Parameters

DerECParams	Resultant Encoding (length prediction supported if NULL).
DerECParamsLen	Buffer len/Length of resultant encoding
M	Degree of field
k1	parameter in trinomial or pentanomial basis polynomial
k2	parameter in pentanomial basis polynomial
k3	parameter in pentanomial basis polynomial
a	Elliptic Curve coefficient a
alen	Elliptic Curve coefficient a length
b	Elliptic Curve coefficient b
blen	Elliptic Curve coefficient b length
seed	Seed (optional may be NULL)
seedlen	Seed length
x	Elliptic Curve point X coord
xlen	Elliptic Curve point X coord length
y	Elliptic Curve point Y coord
ylen	Elliptic Curve point Y coord length
order	Order n of the Base Point
orderlen	Order n of the Base Point length
cofactor	The integer h = #E(Fq)/n (optional)
cofactorlen	h length
Return	Status of operation. CKR_OK if ok.

Sample Domain Parameter Files

The following examples show some sample domain parameter files.

prime192v1

```
#
#This file describes the domain parameters of an EC curve
#
```



```
#File contains lines of text. All lines not of the form key=value are ignored.
#All values must be Hexidecimal numbers except m, k, k1, k2 and k3 which are decimal.
#Lines starting with '#' are comments.
#
#Keys recognised for fieldID values are -
#p          - only if the Curve is based on a prime field
#m          - only if the curve is based on a 2^M field
#k1, k2, k3 - these three only if 2^M field
#
#You should have these combinations of fieldID values -
#p          - if Curve is based on a prime field
#m,k1,k2,k3 - if curve is based on 2^M
#
#These are the values common to prime fields and polynomial fields.
#a          - field element A
#b          - field element B
#s          - this one is optional
#x          - field element Xg of the point G
#y          - field element Yg of the point G
#q          - order n of the point G
#h          - (optional) cofactor h
#
#
# Curve name prime192v1
p    = FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF
a    = FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFC
b    = 64210519E59C80E70FA7E9AB72243049FEB8DEECC146B9B1
s    = 3045AE6FC8422F64ED579528D38120EAE12196D5
x    = 188DA80EB03090F67CBF20EB43A18800F4FF0AFD82FF1012
y    = 07192B95FFC8DA78631011ED6B24CDD573F977A11E794811
q    = FFFFFFFFFFFFFFFFFFFFFFFF99DEF836146BC9B1B4D22831
h    = 1
```

C2tnB191v1

```
#
#This file describes the domain parameters of an EC curve
#
#File contains lines of text. All lines not of the form key=value are ignored.
#All values must be Hexidecimal numbers except m, k, k1, k2 and k3 which are decimal.
#Lines starting with '#' are comments.
#
#Keys recognised for fieldID values are -
#p          - only if the Curve is based on a prime field
#m          - only if the curve is based on a 2^M field
#k1, k2, k3 - these three only if 2^M field
#
#You should have these combinations of fieldID values -
#p          - if Curve is based on a prime field
#m,k1,k2,k3 - if curve is based on 2^M
#
#
These are the values common to prime fields and polynomial fields.
#a          - field element A
#b          - field element B
#s          - this one is optional
#x          - field element Xg of the point G
```



```
#y          - field element Yg of the point G
#q          - order n of the point G
#h          - (optional) cofactor h
#
#
# Curve name C2tnB191v1
m     = 191
k1    = 9
k2    = 0
k3    = 0
a     = 2866537B676752636A68F56554E12640276B649EF7526267
b     = 2E45EF571F00786F67B0081B9495A3D95462F5DE0AA185EC
x     = 36B3DAF8A23206F9C4F299D7B21A9C369137F2C84AE1AA0D
y     = 765BE73433B3F95E332932E70EA245CA2418EA0EF98018FB
q     = 40000000000000000000000004A20E90C39067C893BBB9A5
```

brainpoolP160r1

```
#
#This file describes the domain parameters of an EC curve
#
#File contains lines of text. All lines not of the form key=value are ignored.
#All values must be Hexidecimal numbers except m, k, k1, k2 and k3 which are decimal.
#Lines starting with '#' are comments.
#
#Keys recognised for fieldID values are -
#p          - only if the Curve is based on a prime field
#m          - only if the curve is based on a 2^M field
#k1, k2, k3 - these three only if 2^M field
#
#You should have these combinations of fieldID values -
#p          - if Curve is based on a prime field
#m,k1,k2,k3 - if curve is based on 2^M
#
#These are the values common to prime fields and polynomial fields.
#a          - field element A
#b          - field element B
#s          - this one is optional
#x          - field element Xg of the point G
#y          - field element Yg of the point G
#q          - order n of the point G
#h          - (optional) cofactor h
#
#
# Curve name brainpoolP160r1

p      = E95E4A5F737059DC60DFC7AD95B3D8139515620F
a      = 340E7BE2A280EB74E2BE61BADA745D97E8F7C300
b      = 1E589A8595423412134FAA2DBDEC95C8D8675E58
x      = BED5AF16EA3F6A4F62938C4631EB5AF7BDBCDBC3
y      = 1667CB477A1A8EC338F94741669C976316DA6321
q      = E95E4A5F737059DC60DF5991D45029409E60FC09
h      = 1
```

brainpoolP512r1

```
#
```

```
#This file describes the domain parameters of an EC curve
#
#File contains lines of text. All lines not of the form key=value are ignored.
#All values must be Hexidecimal numbers except m, k, k1, k2 and k3 which are decimal.
#Lines starting with '#' are comments.
#
#Keys recognised for fieldID values are -
#p          - only if the Curve is based on a prime field
#m          - only if the curve is based on a 2^M field
#k1, k2, k3 - these three only if 2^M field
#
#You should have these combinations of fieldID values -
#p          - if Curve is based on a prime field
#m,k1,k2,k3 - if curve is based on 2^M
#
#These are the values common to prime fields and polynomial fields.
#a          - field element A
#b          - field element B
#s          - this one is optional
#x          - field element Xg of the point G
#y          - field element Yg of the point G
#q          - order n of the point G
#h          - (optional) cofactor h
#
#
# Curve name brainpoolP512r1

p=AADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA703308717D4D9B009BC66842AECDA12AE6A380-
E62881FF2F2D82C68528AA6056583A48F3
a=7830A3318B603B89E2327145AC234CC594CBDD8D3DF91610A83441CAEA9863BC2DED5D5AA8253AA10A2EF1C98B9AC8-
B57F1117A72BF2C7B9E7C1AC4D77FC94CA
b=3DF91610A83441CAEA9863BC2DED5D5AA8253AA10A2EF1C98B9AC8B57F1117A72BF2C7B9E7C1AC4D77FC94CADC083E-
67984050B75EBAE5DD2809BD638016F723
x=81AEE4BDD82ED9645A21322E9C4C6A9385ED9F70B5D916C1B43B62EEF4D0098EFF3B1F78E2D0D48D50D1687B93B97D-
5F7C6D5047406A5E688B352209BCB9F822
y=7DDE385D566332ECC0EABFA9CF7822FDF209F70024A57B1AA000C55B881F8111B2DCDE494A5F485E5BCA4BD88A2763-
AED1CA2B2FA8F0540678CD1E0F3AD80892
q=AADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA70330870553E5C414CA92619418661197FAC10-
471DB1D381085DDADDB58796829CA90069
h      = 1
```

Bad Parameter File

```
#
#This file describes the domain parameters of an EC curve
#
#File contains lines of text. All lines not of the form key=value are ignored.
#All values must be Hexidecimal numbers except m, k, k1, k2 and k3 which are decimal.
#Lines starting with '#' are comments.
#
#Keys recognised for fieldID values are -
#p          - only if the Curve is based on a prime field
#m          - only if the curve is based on a 2^M field
#k1, k2, k3 - these three only if 2^M field
#
#You should have these combinations of fieldID values -
#p          - if Curve is based on a prime field
#m,k1,k2,k3 - if curve is based on 2^M
```



```
#
#These are the values common to prime fields and polynomial fields.
#a          - field element A
#b          - field element B
#s          - this one is optional
#x          - field element Xg of the point G
#y          - field element Yg of the point G
#q          - order n of the point G
#h          - (optional) cofactor h
#
# Curve name prime192vx

p    = FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF
a    = FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFC
b    = 64210519E59C80E70FA7E9AB72243049FEB8DEECC146B9B13
s    = 3454556768574352345 7
x    = 188DA80EB03090F67CBF20EB43A18800F4FF0AFD82FF1012
y    = 07192B95FFC8DA78631011ED6B24CDD573F977A11E794811
q    = FFFFFFFFFFFFFFFFFFFFFFFF99DEF836146BC9B1B4D22831
h    = 12323435765786
```

Curve Names By Organization

Elliptic curves are widely used, despite being defined and described differently by different groups in the cryptographic world. The following table attempts to reconcile curve names across three different standards organizations, the SEC Group, ANSI, and NIST.

Table 1: Equivalent curves defined by standards organizations

SECG	ANSI X9.62	NIST
sect163k1		NIST K-163
sect163r1		
sect163r2		NIST B-163
sect193r1		
sect193r2		
sect233k1		NIST K-233
sect233r1		NIST B-233
sect239k1		
sect283k1		NIST K-283
sect283r1		NIST B-283
sect409k1		NIST K-409
sect409r1		NIST B-409
sect571k1		NIST K-571

SECG	ANSI X9.62	NIST
sect571r1		NIST B-571
secp160k1		
secp160r1		
secp160r2		
secp192k1		
secp192r1	prime192v1	NIST P-192
secp224k1		
secp224r1		NIST P-224
secp256k1		
secp256r1	prime256v1	NIST P-256
secp256r1		NIST P-384
secp521r1		NIST P-521

For additional information about the Elliptic Curve specification, see this article:

http://www.ietf.org/rfc/rfc4492.txt

Capability and Policy Configuration Control Using the SafeNet API

The configuration and control of the SafeNet HSM is provided by a set of capabilities and policies which you can query and set using the SafeNet API. See for more information.

HSM Capabilities and Policies

Each HSM has a set of capabilities. An HSM's capability set defines and controls the behaviour of the HSM.

HSM behaviour can be further modified through changing policies. The HSM Admin can refine the behaviour of an HSM by changing the policy settings.

HSM Partition Capabilities and Policies

Each HSM can support one-or-more virtual HSMs called HSM Partitions (may also be called “containers” in some areas of the API), which are used by properly authenticated remote clients to perform cryptographic operations.

Each HSM Partition has a set of capabilities. An HSM Partition's capability set defines and controls the behaviour of the HSM partition.

HSM Partition behaviour can be further modified through changing policies. The HSM Admin can refine the behaviour of an HSM Partition by changing the policy settings. Different Partitions can have different values for the configuration

elements which apply to specific HSM Partitions – in other words, if a Policy is set to a given value for one HSM Partition, the Policy can be set to a different value for another HSM Partitition on the same HSM.

In some cases, a Partition policy change is destructive.

Policy Refinement

For every policy set element, there is a corresponding capability set element (the reverse is not true – there can be some capability set elements that do not have corresponding policy set elements). The value of a policy set element can be modified by the HSM Admin, but only within the limitations imposed by the corresponding capability set element.

For example, there is a policy set element which determines how many failed login attempts may be made before a Partition is deleted or locked out. There is also a corresponding capability set element for the same purpose. The policy element may be modified by the HSM Admin, but may only be set to a value less than or equal to that of the capability set element. So if the capability set element has a value of 10, the HSM Admin can set the policy to a value less than or equal to 10.

In general, the HSM Admin may modify policy set elements to make the HSM or Partition policy more restrictive than that imposed by the capability set elements. The HSM Admin can not make the HSM or HSM Partition policy less restrictive or enable functionality that is disabled through capability settings.

Policy Types

There are three types of policy elements, as follows:

Normal policy elements	May be set at any time by the HSM Admin. The values which may be set are limited only by the corresponding capability element as described in the previous section (i.e. the policy element can be set only to a value less than or equal to the capability set element).
Destructive policy elements	May be set at any time, but setting them results in the erasure of any Partitions and their contents. Policy elements are destructive if changing them significantly affects the security policy of the HSM.
Write-restricted policy elements	Cannot be modified directly, but instead are affected by other actions that can be taken.

Querying and Modifying HSM Configuration

The following are the relevant functions (found in **sfnt_extensions.h**):

- CK_RV CK_ENTRY CA_GetConfigurationElementDescription(
- CK_SLOT_ID slotID,
- CK_ULONG ulIsContainerElement,
- CK_ULONG ulIsCapabilityElement,
- CK_ULONG ulElementId,
- CK_ULONG_PTR pulElementBitLength,
- CK_ULONG_PTR pulElementDestructive,

- CK_ULONG_PTR pulElementWriteRestricted,
- CK_CHAR_PTR pDescription);
- CK_RV CK_ENTRY CA_GetHSMCapabilitySet(
- CK_SLOT_ID uPhysicalSlot,
- CK_ULONG_PTR pulCapIdArray,
- CK_ULONG_PTR pulCapIdSize,
- CK_ULONG_PTR pulCapValArray,
- CK_ULONG_PTR pulCapValSize);
- CK_RV CK_ENTRY CA_GetHSMCapabilitySetting (
- CK_SLOT_ID slotID,
- CK_ULONG ulPolicyId,
- CK_ULONG_PTR pulPolicyValue);
- CK_RV CK_ENTRY CA_GetHSMPolicySet(
- CK_SLOT_ID uPhysicalSlot,
- CK_ULONG_PTR pulPolicyIdArray,
- CK_ULONG_PTR pulPolicyIdSize,
- CK_ULONG_PTR pulPolicyValArray,
- CK_ULONG_PTR pulPolicyValSize);
- CK_RV CK_ENTRY CA_GetHSMPolicySetting (
- CK_SLOT_ID slotID,
- CK_ULONG ulPolicyId,
- CK_ULONG_PTR pulPolicyValue);
- CK_RV CK_ENTRY CA_GetContainerCapabilitySet(
- CK_SLOT_ID uPhysicalSlot,
- CK_ULONG ulContainerNumber,
- CK_ULONG_PTR pulCapIdArray,
- CK_ULONG_PTR pulCapIdSize,
- CK_ULONG_PTR pulCapValArray,
- CK_ULONG_PTR pulCapValSize);
- CK_RV CK_ENTRY CA_GetContainerCapabilitySetting (
- CK_SLOT_ID slotID,
- CK_ULONG ulContainerNumber,
- CK_ULONG ulPolicyId,
- CK_ULONG_PTR pulPolicyValue);
- CK_RV CK_ENTRY CA_GetContainerPolicySet(
- CK_SLOT_ID uPhysicalSlot,

- CK_ULONG ulContainerNumber,
- CK_ULONG_PTR pulPolicyIdArray,
- CK_ULONG_PTR pulPolicyIdSize,
- CK_ULONG_PTR pulPolicyValArray,
- CK_ULONG_PTR pulPolicyValSize);
- CK_RV CK_ENTRY CA_GetContainerPolicySetting(
- CK_SLOT_ID uPhysicalSlot,
- CK_ULONG ulContainerNumber,
- CK_ULONG ulPolicyId,
- CK_ULONG_PTR pulPolicyValue);
- CK_RV CK_ENTRY CA_SetHSMPolicy (
- CK_SESSION_HANDLE hSession,
- CK_ULONG ulPolicyId,
- CK_ULONG ulPolicyValue);
- CK_RV CK_ENTRY CA_SetDestructiveHSMPolicy (
- CK_SESSION_HANDLE hSession,
- CK_ULONG ulPolicyId,
- CK_ULONG ulPolicyValue);
- CK_RV CK_ENTRY CA_SetContainerPolicy (
- CK_SESSION_HANDLE hSession,
- CK_ULONG ulContainer,
- CK_ULONG ulPolicyId,
- CK_ULONG ulPolicyValue);

The CA_GetConfigurationElementDescription() Function

The **CA_GetConfigurationElementDescription()** function requires that you pass in a zero or one value to indicate whether the element you are querying is an HSM Partition (container) element or an HSM element, and another zero/one value to define whether it is a capability or policy that you are interested in. You also pass in the id of the element and a character buffer of at least 60 characters. The function then returns the size of the element value (in bits), an indication of whether the element is destructive, an indication of whether the policy (if it is a policy) is write-restricted, and it also writes the description string into the buffer that you provided.

The CA_Get{HSM|Container}{Capability|Policy}Set() Functions

The various **CA_Get{HSM|Container}{Capability|Policy}Set()** functions all return (in the word arrays provided) a complete list of element id/value pairs for the set specified. For example, **CA_GetHSMCapabilitySet()** returns a list of all HSM capability elements and their values. The parameters for these functions include a list pointer and length pointer for each of the element ids and element values. On calling the function, you should provide a buffer or a null pointer for each of the lists, and the length value should be initialized to the size of the buffer. On return, the buffer (if

given) is populated, and the length is updated to the real length of the list. If the buffer is given but is not large enough, an error results.

Typically you would invoke the function twice: call the function the first time with null buffer pointers so that the real length necessary is returned, then allocate the necessary buffers and call the function a second time, giving the real buffers.

The various **CA_Get{HSM|Container}{Capability|Policy}Setting()** functions allow you to query a specific element value. Provide the element id and the function returns the value.

The CA_Set...() Functions

The various **CA_Set…()** functions allow you to set individual HSM and HSM Partition policies. There are two varieties for setting HSM policies, because changing the value of a destructive HSM policy results in the HSM being cleared of any Partitions and their contents. To make it clear when this is going to happen, the appropriate set function must be called based on whether the HSM policy is destructive or not (which you can determine with the **CA_GetConfigurationElementDescription()** function).

Connection Timeout

The connection timeout is not configurable.

Linux and Unix Connection Timeout

On Unix platforms, the client performs a "connect" on the socket. If the socket is busy or unavailable, the client performs a "select" on the socket with the timeout set to 10 seconds (hardcoded). If the "select" call returns before the timeout, then the client is able to connect. If not then it fails. This prevents the situation where some Unix operating systems can block for several minutes when SafeNet Network HSM is unavailable.

Windows Connection Timeout

On Windows platforms, "connect" is called without "select", relying upon the default Windows timeout of approximately 20 seconds.

6
Design Considerations

This chapter provides guidance for creating applications that use specific SafeNet HSM configurations or features. It contains the following topics:

PED-Authenticated HSMs

In systems or applications using SafeNet HSMs, SafeNet PED is required for FIPS 140-2 level 3 security. In normal use, SafeNet PED supplies PINs and certain other critical security parameters to the token/HSM, invisibly to the user. This prevents other persons from viewing PINs, etc. on a computer screen or watching them typed on a keyboard, which in turn prevents such persons from illicitly cloning token or HSM contents.

Two classes of users operate SafeNet PED: the ordinary HSM Partition Owner, and the HSM Administrator, (also called Security Officer or SO). The person handling new HSMs and using SafeNet PED is normally the HSM SO, who:

- initializes the HSM,
- conducts HSM maintenance, such as firmware and capability upgrades,
- initializes HSM Partitions and tokens,
- creates users (sets PINs),
- changes policy settings,
- changes passwords.

Following these initial activities, the SafeNet PED may be required to present the HSM Partition Owner's PED Key or keys (in case of MofN operations) to enable ordinary signing cryptographic operations carried out by your applications.

With the combination of Activation and AutoActivation, the black PED Key is required only upon initial authentication and then not again unless the authentication is interrupted by power failure or by deliberate action on the part of the PED Key holders.

About CKDemo with SafeNet PED

As its name suggests, CKDemo (CryptoKi Demonstration) is a demonstration program, allowing you to explore the capabilities and functions of several SafeNet products. The demo program breaks out a number of PKCS 11 functions,

as well as the SafeNet extensions to Cryptoki that allow the enhanced capabilities of our HSMs.However the flexibility, combined with the bare-bones nature of the program, can result in some confusion as to whether certain operations and combinations are permissible. Where these come up, in the explanation of CKDemo with SafeNet HSM with PED [Trusted Path] Authentication, and SafeNet PED, they are mentioned and explained if necessary.

The demo program appears to make it optional to permit several of the security operations via the keyboard and program interface, or to require that they be done only via the SafeNet PED keypad. In fact, the option is dictated by the SafeNet HSM, as it was configured and shipped from the factory, and cannot be changed by you. That is, you can use CKDemo to work/experiment with either type of SafeNet HSM – i.e., SafeNet HSM with Password Authentication or SafeNet HSM with PED Authentication, requiring SafeNet PED), but you cannot make one type behave like the other.

Security and design requirements, enforced by the SafeNet HSM with PED Authentication HSM, dictate that use of SafeNet PED be mandatory within the applications that you develop for it.

Interchangeability

As mentioned above, several secrets and security parameters related to HSMs are imprinted on PED Keys which provide "something you have" access control, as opposed to the "something you know" access control provided by password-authenticated HSMs. The HSM can create each type of secret, which is then also imprinted on a suitably labeled PED Key. Alternatively, the secret can be accepted from a PED Key (previously imprinted by another HSM) and imprinted on the current HSM. This is mandatory for the cloning domain, when HSMs (or HSM partitions) are to clone objects one to the other. It is optional for the other HSM secrets, as a matter of convenience or of your security policy, allowing more than one HSM to be accessed for administration by a single SO (blue PED Key holder) or more than one HSM Partition to be administered by a single Partition Owner/User. The exception is the SRK (purple PED Key) which carries a secret unique to its HSM and which cannot be imprinted on any other HSM.

PED Keys that have never been imprinted are completely interchangeable. They can be used with any modern SafeNet HSM, and can be imprinted with any of the various secrets. The self-stick labels are provided as a visual identifier of which type of secret has been imprinted on a PED Key, or is about to be imprinted .Imprinted PED Keys are tied to their associated HSMs and cannot be used to access HSMs or partitions that have been imprinted with different secrets.

Any SafeNet PED2 can be used with any SafeNet HSM - the PED itself contains no secrets; it simply provides the interface between you and your HSM(s). The exception is that only some SafeNet PEDs have the capability to be used remotely from the HSM. Any Remote-capable SafeNet PED2 is interchangeable with any other Remote-capable SafeNet PED2, and any SafeNet PED2 (remote-capable or not) is interchangeable with any other when locally connected to a SafeNet HSM.

HSM Partitions and Backup Tokens and PED Keys can be “re-cycled” for use in different combinations, but this reuse requires re-initializing the HSM(s) and re-imprinting the PED Keys with new secrets or security parameters. Re-initializing a token or HSM wipes previous information from it. Re-imprinting a PED Key overwrites any previous information it carried (PIN, domain, etc.).

Startup

SafeNet PED expects to be connected to a SafeNet HSM with Trusted Path Authentication. At power-up, it presents a message showing its firmware version. After a few seconds, the message changes to "Awaiting command.." The SafeNet PED is waiting for a command from the token/HSM.

The SafeNet PED screen remains in this status until the CKDemo program, or your own application, initiates a command through the token/HSM.

For the purposes of demonstration, you would now go ahead and create some objects and perform other transactions with the HSM.

Note: To perform most actions you must be logged in. CKDemo may not remind you before you perform actions out-of-order, but it generates error messages after such attempts. So, in general, if you receive an error message from the program, review your recent actions to determine if you have logged out or closed sessions and then not formally logged into a new session before attempting to create an object or perform other token/HSM actions.

When you do wish to end activities, be sure to formally log out and close sessions. With CKDemo, it would be merely an inconvenience to have old sessions still open when you attempt new activities. An orderly shutdown of your application, however, should include logging out any users and closing all sessions on HSMs.

Cloning of Tokens

To securely copy the contents of a SafeNet Network HSM Partition to another SafeNet Network HSM Partition (on the same SafeNet Network HSM or on another), you must perform a backup to a SafeNet Backup HSM from the source HSM Partition followed by a restore operation from the Backup HSM to the new destination HSM Partition. This is done via lunash command line, and cannot be accomplished via CKDemo.

High Availability (HA) Implementations

If you use the SafeNet Network HSM HA feature then the calls to the SafeNet Enterprise HSMs are load-balanced. The session handle that the application receives when it opens a session is a virtual one and is managed by the HA code in the library. The actual sessions with the HSM are established by the HA code in the library and hidden from the application and will come and go as necessary to fulfill application level requests.

Before the introduction of HA AutoInsert[1] , bringing a failed or lost group member back into the group (recovery) was a manual procedure.

The Administration & Maintenance section contains a general description of the how the HA AutoInsert function works, in practice.

For every PKCS11 call, the HA recover logic will check to see if we need to perform automatic recovery to a disconnected appliance. If there is a disconnected appliance then it will try to reconnect to that appliance before it proceeds with the current PKCS11 call.

The HA recovery logic is designed in such a way that it will try to reconnect to an appliance only every X secs and N number of times where X is pre-set to one minute, and N is configurable via the "VTL" utility.

For HA recovery attempts:

- The default retry interval is 60 seconds.
- The default number of retries is effectively infinite.
- The HA configuration section in the Chrystoki.conf/crystoki.ini file is created and populated when either the interval or the number of retries is specified in the lunacm hagroup retry commands.

The following is the pseudo code of the HA logic

```
if (disconnected_member > 0  and recover_attempt_count < N and time_now - last_recover_attempt >
X) then
   performance auto recovery
   set last_recover_attempt  equal to time_now
```

[1]also known as "autorecovery", the re-introduction of a failed or lost member to an HA group.


```
   if  (recovery failed)  then
      increment recover_attempt_count  by 1
   else
      decrement disconnected_member by 1
      reset recover_attempt_count to 0
   end if
end if
```

The HA automatic recovery design runs within a PKCS#11 call. The responsiveness of recovering a disconnected member is greatly influenced by the frequency of PKCS11 calls from the user application. Although the logic shows that it will attempt to recover a disconnected client in X secs, in reality, it will not run until the user application makes the next PKCS11 call.

Detecting the Failure of an HA Member

When an HA Group member first fails, the HA status for the group shows "device error" for the failed member. All subsequent calls return "token not present", until the member (HSM Partition or PKI token) is returned to service.

Here is an example of two such calls using CKDemo:

```
Enter your choice : 52
Slots available:
   slot#1 - LunaNet Slot
   slot#2 - LunaNet Slot
   slot#3 - HA Virtual Card Slot

Select a slot: 3

HA group 1599447001 status:
   HSM 599447001      - CKR_DEVICE_ERROR
   HSM 78665001       - CKR_OK
Status: Doing great, no errors (CKR_OK)

TOKEN FUNCTIONS
( 1) Open Session  ( 2) Close Session  ( 3) Login
( 4) Logout        ( 5) Change PIN     ( 6) Init Token
( 7) Init Pin      ( 8) Mechanism List ( 9) Mechanism Info
(10) Get Info      (11) Slot Info      (12) Token Info
(13) Session Info  (14) Get Slot List  (15) Wait for Slot Event
(16) InitToken(ind)(17) InitPin (ind)  (18) Login (ind)
(19) CloneMofN

OBJECT MANAGEMENT FUNCTIONS
(20) Create object (21) Copy object    (22) Destroy object
(23) Object size   (24) Get attribute  (25) Set attribute
(26) Find object   (27) Display Object

SECURITY FUNCTIONS
(40)  Encrypt file (41) Decrypt file   (42)  Sign
(43)  Verify       (44) Hash file      (45)  Simple Generate Key
(46)  Digest Key

HIGH AVAILABILITY RECOVERY FUNCTIONS
(50) HA Init       (51) HA Login       (52) HA Status

KEY FUNCTIONS
(60) Wrap key      (61) Unwrap key     (62) Generate random number
(63) Derive Key    (64) PBE Key Gen    (65) Create known keys
(66) Seed RNG      (67) EC User Defined Curves
```



```
CA FUNCTIONS
(70) Set Domain     (71) Clone Key       (72) Set MofN
(73) Generate MofN  (74) Activate MofN   (75) Generate Token Keys
(76) Get Token Cert (77) Sign Token Cert (78) Generate CertCo Cert
(79) Modify MofN    (86) Dup. MofN Keys  (87) Deactivate MofN

CCM FUNCTIONS
(80) Module List    (81) Module Info     (82) Load Module
(83) Load Enc Mod   (84) Unload Module   (85) Module function Call

OTHERS
(90) Self Test      (94) Open Access     (95) Close Access
(97) Set App ID     (98) Options

OFFBOARD KEY STORAGE:
(101) Extract Masked Object     (102) Insert Masked Object
(103) Multisign With Value      (104) Clone Object
(105) SIMExtract                (106) SIMInsert
(107) SimMultiSign

SCRIPT EXECUTION:
(108) Execute Script
(109) Execute Asynchronous Script
(110) Execute Single Part Script
(0) Quit demo
Enter your choice : 52

Slots available:
   slot#1 - LunaNet Slot
   slot#2 - LunaNet Slot
   slot#3 - HA Virtual Card Slot

Select a slot: 3

HA group 1599447001 status:
   HSM 599447001      - CKR_TOKEN_NOT_PRESENT
   HSM 78665001       - CKR_OK
Status: Doing great, no errors (CKR_OK)
--- end ---
```

Migrating Keys From Software to a SafeNet HSM

SafeNet HSMs expect key material to be in PKCS#8 format. PKCS#8 format follows BER (Basic encoding rules) /DER (distinguished encoding rules) encoding. An example of this format can be found in the document "Some examples of PKCS standards" produced by RSA, and available on their web site (http://www.rsasecurity.com/rsalabs/pkcs/index.html at the bottom of the page, under “Related Documents”).

Here is an example of a formatted key:

```
 0x30,
 0x82, 0x04, 0xbc, 0x02, 0x01, 0x00, 0x30, 0x0d, 0x06, 0x09, 0x2a, 0x86,
 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01, 0x01, 0x05, 0x00, 0x04, 0x82, 0x04,
 0xa6, 0x30, 0x82, 0x04, 0xa2, 0x02, 0x01, 0x00, 0x02, 0x82, 0x01, 0x01,
 0x00, 0xb8, 0xb5, 0x0f, 0x49, 0x46, 0xb5, 0x5d, 0x58, 0x04, 0x8e, 0x52,
 0x59, 0x39, 0xdf, 0xd6, 0x29, 0x45, 0x6b, 0x6c, 0x96, 0xbb, 0xab, 0xa5,
 0x6f, 0x72, 0x1b, 0x16, 0x96, 0x74, 0xd5, 0xf9, 0xb4, 0x41, 0xa3, 0x7c,
 0xe1, 0x94, 0x73, 0x4b, 0xa7, 0x23, 0xff, 0x61, 0xeb, 0xce, 0x5a, 0xe7,
```



```
0x7f, 0xe3, 0x74, 0xe8, 0x52, 0x5b, 0xd6, 0x5d, 0x5c, 0xdc, 0x98, 0x49,
0xfe, 0x51, 0xc2, 0x7e, 0x8f, 0x3b, 0x37, 0x5c, 0xb3, 0x11, 0xed, 0x85,
0x91, 0x15, 0x92, 0x24, 0xd8, 0xf1, 0x7b, 0x3d, 0x2f, 0x8b, 0xcd, 0x1b,
0x30, 0x14, 0xa3, 0x6b, 0x1b, 0x4d, 0x27, 0xff, 0x6a, 0x58, 0x84, 0x9e,
0x79, 0x94, 0xca, 0x78, 0x64, 0x01, 0x33, 0xc3, 0x58, 0xfc, 0xd3, 0x83,
0xeb, 0x2f, 0xab, 0x6f, 0x85, 0x5a, 0x38, 0x41, 0x3d, 0x73, 0x20, 0x1b,
0x82, 0xbc, 0x7e, 0x76, 0xde, 0x5c, 0xfe, 0x42, 0xd6, 0x7b, 0x86, 0x4f,
0x79, 0x78, 0x29, 0x82, 0x87, 0xa6, 0x24, 0x43, 0x39, 0x74, 0xfe, 0xf2,
0x0c, 0x08, 0xbe, 0xfa, 0x1e, 0x0a, 0x48, 0x6f, 0x14, 0x86, 0xc5, 0xcd,
0x9a, 0x98, 0x09, 0x2d, 0xf3, 0xf3, 0x5a, 0x7a, 0xa4, 0xe6, 0x8a, 0x2e,
0x49, 0x8a,  0xde,
0x73, 0xe9, 0x37, 0xa0, 0x5b, 0xef, 0xd0, 0xe0, 0x13, 0xac, 0x88, 0x5f,
0x59, 0x47, 0x96, 0x7f, 0x78, 0x18, 0x0e, 0x44, 0x6a, 0x5d,  0xec,
0x6e, 0xed, 0x4f, 0xf6, 0x6a, 0x7a, 0x58, 0x6b, 0xfe, 0x6c, 0x5a, 0xb9,
0xd2, 0x22, 0x3a, 0x1f, 0xdf, 0xc3, 0x09, 0x3f, 0x6b, 0x2e, 0xf1, 0x6d,
0xc3, 0xfb, 0x4e, 0xd4, 0xf2, 0xa3, 0x94, 0x13, 0xb0, 0xbf, 0x1e, 0x06,
0x2e, 0x29, 0x55, 0x00, 0xaa, 0x98, 0xd9, 0xe8, 0x77, 0x84, 0x8b, 0x3f,
0x5f, 0x5e, 0xf7, 0xf8, 0xa7, 0xe6, 0x02, 0xd2, 0x18, 0xb0, 0x52, 0xd0,
0x37, 0x2e, 0x53, 0x02, 0x03, 0x01, 0x00, 0x01, 0x02, 0x82, 0x01, 0x00,
0x0c, 0xdf, 0xd1, 0xe8, 0xf1, 0x9c, 0xc2, 0x9c, 0xd7, 0xf4, 0x73, 0x98,
0xf4, 0x87, 0xbd, 0x8d, 0xb2, 0xe1, 0x01, 0xf8, 0x9f, 0xac, 0x1f, 0x23,
0xdd, 0x78, 0x35, 0xe2, 0xd6, 0xd1, 0xf3, 0x4d, 0xb5, 0x25, 0x88, 0x16,
0xd1, 0x1a, 0x18, 0x33, 0xd6, 0x36, 0x7e, 0xc4, 0xc8, 0xe5, 0x5d, 0x2d,
0x74, 0xd5, 0x39, 0x3c, 0x44, 0x5a, 0x74, 0xb7, 0x7c, 0x48, 0xc1, 0x1f,
0x90, 0xe3, 0x55, 0x9e, 0xf6, 0x29, 0xad, 0xb4, 0x6d, 0x93, 0x78, 0xb3,
0xdc, 0x25, 0x0b, 0x9c, 0x73, 0x78, 0x7b, 0x93, 0x4c, 0xd3, 0x47, 0x09,
0xda, 0xe6, 0x69, 0x18, 0xc6, 0x0f, 0xfb, 0xa5, 0x95, 0xf5, 0xe8, 0x75,
0xe1, 0x01, 0x1b, 0xd3, 0x1c, 0xa2, 0x57, 0x03, 0x64, 0xdb, 0xf9, 0x5d,
0xf3, 0x3c, 0xa7, 0xd1, 0x4b, 0xb0, 0x90, 0x1b, 0x90, 0x62, 0xb4, 0x88,
0x30, 0x4b, 0x40, 0x4d, 0xcf, 0x7d, 0x89, 0x7a, 0xfb, 0x29, 0xc9, 0x64,
0x34, 0x0a, 0x52, 0xf6, 0x70, 0x7c, 0x76, 0x5a, 0x2e, 0x8f, 0x50, 0xd4,
0x92, 0x15, 0x97, 0xed, 0x4c, 0x2e, 0xf2, 0x3a, 0xd0, 0x58, 0x7e, 0xdb,
0xf1, 0xf4, 0xdd, 0x07, 0x76, 0x04, 0xf0, 0x55, 0x8b, 0x72, 0x2b, 0xa7,
0xa8, 0x78, 0x78, 0x67, 0xe6, 0xd8, 0xa5, 0xde, 0xe7, 0xc9, 0x1f, 0x5a,
0xa0, 0x89, 0xc7, 0x24, 0xa2, 0x71, 0xb6, 0x7b, 0x3b, 0xe6, 0x92, 0x69,
0x22, 0xaa, 0xe2, 0x47, 0x4b, 0x80, 0x3f, 0x6a, 0xab, 0xce, 0x4e, 0xcd,
0xe8, 0x94, 0x6c, 0xf7, 0x84, 0x73, 0x85, 0xfd, 0x85, 0x1d, 0xae, 0x81,
0xf7, 0xec, 0x12, 0x31, 0x7d, 0xc1, 0x99, 0xc0, 0x3c, 0x51, 0xb0, 0xdc,
0xb0, 0xba, 0x9c, 0x84, 0xb8, 0x70, 0xc2, 0x09, 0x7f, 0x96, 0x3d, 0xa1,
0xe2, 0x64, 0x27, 0x7a, 0x22, 0xb8, 0x75, 0xb9, 0xd1, 0x5f, 0xa5, 0x23,
0xf9, 0x62, 0xe0, 0x41, 0x02, 0x81, 0x81, 0x00, 0xf4, 0xf3, 0x08, 0xcf,
0x83, 0xb0, 0xab, 0xf2, 0x0f, 0x1a, 0x08, 0xaf, 0xc2, 0x42, 0x29, 0xa7,
0x9c, 0x5e, 0x52, 0x19, 0x69, 0x8d, 0x5b, 0x52, 0x29, 0x9c, 0x06, 0x6a,
0x5a, 0x32, 0x8f, 0x08, 0x45, 0x6c, 0x43, 0xb5, 0xac, 0xc3, 0xbb, 0x90,
0x7b, 0xec, 0xbb, 0x5d, 0x71, 0x25, 0x82, 0xf8, 0x40, 0xbf, 0x38, 0x00,
0x20, 0xf3, 0x8a, 0x38, 0x43, 0xde, 0x04, 0x41, 0x19, 0x5f, 0xeb, 0xb0,
0x50, 0x59, 0x10, 0xe1, 0x54, 0x62, 0x5c, 0x93, 0xd9, 0xdc, 0x63, 0x24,
0xd0, 0x17, 0x00, 0xc0, 0x44, 0x3e, 0xfc, 0xd1, 0xda, 0x4b, 0x24, 0xf7,
0xcb, 0x16, 0x35, 0xe6, 0x9f, 0x67, 0x96, 0x5f, 0xb0, 0x94, 0xde, 0xfa,
0xa1, 0xfd, 0x8c, 0x8a, 0xd1, 0x5c, 0x02, 0x8d, 0xe0, 0xa0, 0xa0,  0x02,
0x1d, 0x56, 0xaf, 0x13, 0x3a, 0x65, 0x5e, 0x8e, 0xde, 0xd1, 0xa8, 0x28,
0x8b, 0x71, 0xc9, 0x65, 0x02, 0x81, 0x81, 0x00, 0xc1, 0x0a,  0x47,
0x39, 0x91, 0x06, 0x1e, 0xb9, 0x43, 0x7c, 0x9e, 0x97, 0xc5, 0x09, 0x08,
0xbc, 0x22, 0x47, 0xe2, 0x96, 0x8e, 0x1c, 0x74, 0x80, 0x50, 0x6c, 0x9f,
0xef, 0x2f, 0xe5, 0x06, 0x3e, 0x73, 0x66, 0x76, 0x02, 0xbd, 0x9a, 0x1c,
0xfc, 0xf9, 0x6a, 0xb8, 0xf9, 0x36, 0x15, 0xb5, 0x20, 0x0b, 0x6b, 0x54,
0x83, 0x9c, 0x86, 0xba, 0x13, 0xb7, 0x99, 0x54, 0xa0, 0x93, 0x0d, 0xd6,
```



```
0x1e, 0xc1, 0x12, 0x72, 0x0d, 0xea, 0xb0, 0x14, 0x30, 0x70, 0x73, 0xef,
0x6b, 0x4c, 0xae, 0xb6, 0xff, 0xd4, 0xbb, 0x89, 0xa1, 0xec, 0xca, 0xa6,
0xe9, 0x95, 0x56, 0xac, 0xe2, 0x9b, 0x97, 0x2f, 0x2c, 0xdf, 0xa3, 0x6e,
0x59, 0xff, 0xcd, 0x3c, 0x6f, 0x57, 0xcc, 0x6e, 0x44, 0xc4, 0x27, 0xbf,
0xc3, 0xdd, 0x19, 0x9e, 0x81, 0x16, 0xe2, 0x8f, 0x65, 0x34, 0xa7, 0x0f,
0x22, 0xba, 0xbf, 0x79, 0x57, 0x02, 0x81, 0x80, 0x2e, 0x21, 0x0e, 0xc9,
0xb5, 0xad, 0x31, 0xd4, 0x76, 0x0f, 0x9b, 0x0f, 0x2e, 0x70, 0x33, 0x54,
0x03, 0x58, 0xa7, 0xf1, 0x6d, 0x35, 0x57, 0xbb, 0x53, 0x66, 0xb4, 0xb6,
0x96, 0xa1, 0xea, 0xd9, 0xcd, 0xe9, 0x23, 0x9f, 0x35, 0x17, 0xef, 0x5c,
0xb8, 0x59, 0xce, 0xb7, 0x3c, 0x35, 0xaa, 0x42, 0x82, 0x3f, 0x00, 0x96,
0xd5, 0x9d, 0xc7, 0xab, 0xec, 0xec, 0x04, 0xb5, 0x15, 0xc8, 0x40, 0xa4,
0x85, 0x9d, 0x20, 0x56, 0xaf, 0x03, 0x8f, 0x17, 0xb0, 0xf1, 0x96, 0x22,
0x3a, 0xa5, 0xfa, 0x58, 0x3b, 0x01, 0xf9, 0xae, 0xb3, 0x83, 0x6f, 0x44,
0xd3, 0x14, 0x2d, 0xb6, 0x6e, 0xd2, 0x9d, 0x39, 0x0c, 0x12, 0x1d, 0x23,
0xea, 0x19, 0xcb, 0xbb, 0xe0, 0xcd, 0x89, 0x15, 0x9a, 0xf5, 0xe4, 0xec,
0x41, 0x06, 0x30, 0x16, 0x58, 0xea, 0xfa, 0x31, 0xc1, 0xb8, 0x8e, 0x08,
0x84, 0xaa, 0x3b, 0x19, 0x02, 0x81, 0x80, 0x70, 0x4c, 0xf8, 0x6e, 0x86,
0xed, 0xd6, 0x85, 0xd4, 0xba, 0xf4, 0xd0, 0x3a, 0x32, 0x2d, 0x40, 0xb5,
0x78, 0xb8, 0x5a, 0xf9, 0xc5, 0x98, 0x08, 0xe5, 0xc0, 0xab, 0xb2, 0x4c,
0x5c, 0xa2, 0x2b, 0x46, 0x9b, 0x3e, 0xe0, 0x0d, 0x49, 0x50, 0xbf, 0xe2,
0xa1, 0xb1, 0x86, 0x59, 0x6e, 0x7b, 0x76, 0x6e, 0xee, 0x3b, 0xb6, 0x6d,
0x22, 0xfb, 0xb1, 0x68, 0xc7, 0xec, 0xb1, 0x95, 0x9b, 0x21, 0x0b, 0xb7,
0x2a, 0x71, 0xeb, 0xa2, 0xb2, 0x58, 0xac, 0x6d, 0x5f, 0x24, 0xd3, 0x79,
0x42, 0xd2, 0xf7, 0x35, 0xdc, 0xfc, 0x0e, 0x95, 0x60, 0xb7, 0x85, 0x7f,
0xf9, 0x72, 0x8e, 0x4a, 0x11, 0xc3, 0xc2, 0x09, 0x40, 0x5c, 0x7c, 0x43,
0x12, 0x34, 0xac, 0x59, 0x99, 0x76, 0x34,  0xcf,
0x20, 0x88, 0xb0, 0xfb, 0x39, 0x62, 0x3a, 0x9b, 0x03, 0xa6, 0x84, 0x2c,
0x03, 0x5c, 0x0c, 0xca, 0x33, 0x85, 0xf5, 0x02, 0x81, 0x80,  0x56,
0x99, 0xe9, 0x17, 0xdc, 0x33, 0xe1, 0x33, 0x8d, 0x5c, 0xba, 0x17, 0x32,
0xb7, 0x8c, 0xbd, 0x4b, 0x7f, 0x42, 0x3a, 0x79, 0x90, 0xe3,  0x70,
0xe3, 0x27, 0xce, 0x22, 0x59, 0x02, 0xc0, 0xb1, 0x0e, 0x57, 0xf5, 0xdf,
0x07, 0xbf, 0xf8, 0x4e, 0x10, 0xef, 0x2a, 0x62, 0x30, 0x03,  0xd4,
0x80, 0xcf, 0x20, 0x84, 0x25, 0x66, 0x3f, 0xc7, 0x4f, 0x56, 0x8c, 0x1e,
0xe1, 0x18, 0x91, 0xc1, 0xfd, 0x71, 0x5f, 0x65, 0x9b, 0xe4,  0x4f,
0xe0, 0x1a, 0x3a, 0xf8, 0xc1, 0x69, 0xdb, 0xd3, 0xbb, 0x8d, 0x91, 0xd1,
0x11, 0x4f, 0x7e, 0x91, 0x1b, 0xb4, 0x27, 0xa5, 0xab, 0x7c,  0x7b,
0x76, 0xd4, 0x78, 0xfe, 0x63, 0x44, 0x63, 0x7e, 0xe3, 0xa6, 0x60, 0x4f,
0xb9, 0x55, 0x28, 0xba, 0xba, 0x83, 0x1a, 0x2d, 0x43, 0xd5,  0xf7,
0x2e, 0xe0, 0xfc, 0xa8, 0x14, 0x9b, 0x91, 0x2a, 0x36, 0xbf, 0xc7, 0x14
```

The example above contains the exponent, the modulus, and private key material.

Other Formats of Key Material

The format of key material depends on the application, and is therefore unpredictable. Key material commonly exists in any of the following formats; ASN1, PEM, P12, PFX, etc. Key material in those formats, or in another format, can likely be re-formatted to be acceptable for moving onto the SafeNet HSM.

Sample Program

The sample program below encrypts a known RSA private key, then unwraps the key pair onto the SafeNet HSM Partition.

```
/*****************************************************************************\
*
```



```
* File: UnwrapKey.cpp*
* Encrypts a PrivateKeyInfo structure with a generated DES key and then
* unwraps the RSA key onto a token.
*
* This file is provided as an example only.
*
*
* Copyright (C) 2011, SafeNet, Inc.
*
* All rights reserved. This file contains information that is
* proprietary to SafeNet, Inc. and may not be
* distributed or copied without written consent from
* SafeNet, Inc.
*
\*************************************************************/
#ifdef UNIX
#define _POSIX_SOURCE 1
#endif
#ifdef USING_STATIC_CHRYSTOKI
# define STATIC  ckdemo_cpp
#endif
#include <assert.h>
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <time.h>
#ifdef _WINDOWS
#include <conio.h>
#include <io.h>
#include <windows.h>
#endif
#ifdef UNIX
#include <unistd.h>
```



```
#endif
#include "source/cryptoki.h"
#include "source/Ckbridge.h"
#define DIM(a)              (sizeof(a)/sizeof(a[0]))
CK_BBOOL no = FALSE;
CK_BBOOL yes = TRUE;
const int MAX =100;
// Function Prototypes
CK_RV  Pinlogin(CK_SESSION_HANDLE hSession);
int getPinString(CK_CHAR_PTR pw);
// Main
int main( void )
{
  int error = 0;
  CK_RV retCode = CKR_OK;
  CK_SESSION_HANDLE hSessionHandle;
  CK_CHAR_PTR userPIN = (CK_CHAR_PTR)"default";
  CK_USHORT lenuserPIN = 7;
  CK_CHAR_PTR soPIN = (CK_CHAR_PTR)"default";
  CK_USHORT lensoPIN = 7;
  CK_USHORT usNumberOfSlots;
  CK_SLOT_ID_PTR pSlotList;
  CK_OBJECT_HANDLE hKey;
  CK_MECHANISM mech;
  CK_VERSION version;
  struct
  {
    CK_INFO info;
    char reserved[100]; // This is in case the library that we are
                // talking to requires a larger info structure
                // then the one defined.
  } protectedInfo;
//Disclaimer
  cout << "\n\n\n\n";
cout << "THE SOFTWARE IS PROVIDED BY SAFENET INCORPORATED (SAFENET) ON AN 'AS IS' BASIS, \n";
```



```
cout << "WITHOUT ANY OTHER WARRANTIES OR CONDITIONS, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED \n";

cout << "TO, WARRANTIES OF MERCHANTABLE QUALITY, SATISFACTORY QUALITY, MERCHANTABILITY OR FITNESS FOR\n";

cout << "A PARTICULAR PURPOSE, OR THOSE ARISING BY LAW, STATUTE, USAGE OF TRADE, COURSE OF DEALING OR\n";

cout << "OTHERWISE.  SAFENET DOES NOT WARRANT THAT THE SOFTWARE WILL MEET YOUR REQUIREMENTS OR \n";

cout << "THAT OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR THAT THE SOFTWARE WILL BE ERROR-FREE.\n";

cout << "YOU ASSUME THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE. NEITHER \n";

cout << "SAFENET NOR OUR LICENSORS, DEALERS OR SUPPLIERS SHALL HAVE ANY LIABILITY TO YOU OR ANY\n";

cout << "OTHER PERSON OR ENTITY FOR ANY INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, PUNITIVE, \n";

cout << "EXEMPLARY OR AY OTHER DAMAGES WHATSOEVER, INCLUDING, BUT NOT LIMITED TO, LOSS OF REVENUE OR \n";

cout << "PROFIT, LOST OR DAMAGED DATA, LOSS OF USE OR OTHER COMMERCIAL OR ECONOMIC LOSS, EVEN IF \n";

cout << "SAFENET HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR THEY ARE FORESEEABLE. \n";

cout << "SAFENET IS ALSO NOT RESPONSIBLE FOR CLAIMS BY A THIRD PARTY.  THE MAXIMUM AGGREGATE \n";

cout << "LIABILITY OF SAFENET TO YOU AND THAT OF SAFENET'S LICENSORS, DEALERS AND SUPPLIERS \n";

cout << "SHALL NOT EXCEED FORTY DOLLARS ($40.00CDN). THE LIMITATIONS IN THIS SECTION SHALL APPLY \n";

cout << "WHETHER OR NOT THE ALLEGED BREACH OR DEFAULT IS A BREACH OF A FUNDAMENTAL CONDITION OR TERM \n";

cout << "OR A FUNDAMENTAL BREACH.  SOME STATES/COUNTRIES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF\n";

cout << "LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO \n";

cout << "YOU.\n";

cout << "THE LIMITED WARRANTY, EXCLUSIVE REMEDIES AND LIMITED LIABILITY SET OUT HEREIN ARE FUNDAMENTAL \n";

cout << "ELEMENTS OF THE BASIS OF THE BARGAIN BETWEEN YOU AND SAFENET.  \n";

cout << "NO SUPPORT.  YOU ACKNOWLEDGE AND AGREE THAT THERE ARE NO SUPPORT SERVICES PROVIDED BY SAFENET\n";

cout << "INCORPORATED FOR THIS SOFTWARE\n" << endl;
```



```
  // Display Generic Warning
  cout << "\nInsert a token for the test...";
  cout << "\n\nWARNING!!! This test initializes the first ";
  cout << " token detected in the card reader.";
  cout << "\nDo not use a token that you don't want erased.";
  cout << "\nYou can use CTRL-C to abort now...Otherwise...";
  cout << "\n\n... press <Enter> key to continue ...\n";
  cout.flush();
  getchar(); // Wait for keyboard hit
#ifndef STATIC
  // Connect to Chrystoki
if(!CrystokiConnect())
{
cout << "\n" "Unable to connect to Chrystoki. Error = " << LibError() << "\n";
error = -1;
    goto exit_routine_1;
}
#endif
  // Verify this is the version of the library required
  retCode = C_GetInfo(&protectedInfo.info);
  if( retCode != CKR_OK )
  {
    cout << endl << "Unable to call C_GetInfo() before C_Initialize()\n";
error = -2;
    goto exit_routine_2;
  }
  else
  {
    CK_BYTE majorVersion = protectedInfo.info.version.major;
    CK_BYTE expectedVersion;
#ifndef PKCS11_2_0
    expectedVersion = 1;
#else
    expectedVersion = 2;
#endif
    if( expectedVersion != majorVersion )
```



```
    {
      cout << endl << "This version of the program was built for Cryptoki version "
          << (int)expectedVersion << ".\n"
          << "The loaded Cryptoki library reports its version to be "
          << (int)majorVersion << ".\n"
          << "Program will terminate.\n";
      // Wait to exit until user read message and acknowledges
      cout << endl << "Press <Enter> key to end.";
      getchar(); // Wait for keyboard hit
  error = -3;
      goto exit_routine_2;
    }
  }
  // Initialize the Library
retCode = C_Initialize(NULL);
if(retCode != CKR_OK)
{
cout << "\n" "Error 0x" << hex << retCode << " initializing cryptoki.\n";
error = -4;
    goto exit_routine_3;
}
// Get the number of tokens possibly available
retCode = C_GetSlotList(TRUE, NULL, &usNumberOfSlots);
if(retCode != CKR_OK)
{
cout << "\n" "Error 0x" << hex << retCode << " getting slot list.\n";
error = -5;
    goto exit_routine_3;
}
// Are any tokens present?
if(usNumberOfSlots == 0)
{
cout << "\n" "No tokens found\n";
error = -6;
    goto exit_routine_3;
  }
```



```
  // Get a list of slots
pSlotList = new CK_SLOT_ID[usNumberOfSlots];
retCode = C_GetSlotList(TRUE, pSlotList, &usNumberOfSlots);
if(retCode != CKR_OK)
{
cout << "\n" "Error 0x" << hex << retCode << " getting slot list.\n";
error = -7;
    goto exit_routine_4;
}
  // Open a session
retCode = C_OpenSession(pSlotList[0], CKF_RW_SESSION | CKF_SERIAL_SESSION,
    NULL, NULL, &hSessionHandle);
if(retCode != CKR_OK)
{
cout << "\n" "Error 0x" << hex << retCode << " opening session.\n";
error = -9;
    goto exit_routine_4;
}
Pinlogin(hSessionHandle);
if(retCode != CKR_OK)
{
cout << "\n" "Error 0x" << hex << retCode << " Calling PinLogin fn";
exit(hSessionHandle);
}
 // Encrypt an RSA Key and then unwrap it onto the token
  {
    // The following is an RSA Key that is formatted as a PrivateKeyInfo structure
  //BER encoded format
    const CK_BYTE pRsaKey[] = {
 0x30, 0x82, 0x04, 0xbc, 0x02, 0x01, 0x00, 0x30, 0x0d, 0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01,
0x01, 0x05, 0x00, 0x04,
 0x82, 0x04, 0xa6, 0x30, 0x82, 0x04, 0xa2, 0x02, 0x01, 0x00, 0x02, 0x82, 0x01, 0x01, 0x00, 0xb8, 0xb5, 0x0f, 0x49,
0x46, 0xb5, 0x5d, 0x58,
 0x04, 0x8e, 0x52, 0x59, 0x39, 0xdf, 0xd6, 0x29, 0x45, 0x6b, 0x6c, 0x96, 0xbb, 0xab, 0xa5, 0x6f, 0x72, 0x1b, 0x16,
0x96, 0x74, 0xd5, 0xf9,
 0xb4, 0x41, 0xa3, 0x7c, 0xe1, 0x94, 0x73, 0x4b, 0xa7, 0x23, 0xff, 0x61, 0xeb, 0xce, 0x5a, 0xe7, 0x7f, 0xe3, 0x74,
```


0xe8, 0x52, 0x5b, 0xd6,
0x5d, 0x5c, 0xdc, 0x98, 0x49, 0xfe, 0x51, 0xc2, 0x7e, 0x8f, 0x3b, 0x37, 0x5c, 0xb3, 0x11, 0xed, 0x85, 0x91, 0x15, 0x92, 0x24, 0xd8, 0xf1,
0x7b, 0x3d, 0x2f, 0x8b, 0xcd, 0x1b, 0x30, 0x14, 0xa3, 0x6b, 0x1b, 0x4d, 0x27, 0xff, 0x6a, 0x58, 0x84, 0x9e, 0x79, 0x94, 0xca, 0x78, 0x64,
0x01, 0x33, 0xc3, 0x58, 0xfc, 0xd3, 0x83, 0xeb, 0x2f, 0xab, 0x6f, 0x85, 0x5a, 0x38, 0x41, 0x3d, 0x73, 0x20, 0x1b, 0x82, 0xbc, 0x7e, 0x76,
0xde, 0x5c, 0xfe, 0x42, 0xd6, 0x7b, 0x86, 0x4f, 0x79, 0x78, 0x29, 0x82, 0x87, 0xa6, 0x24, 0x43, 0x39, 0x74, 0xfe, 0xf2, 0x0c, 0x08, 0xbe,
0xfa, 0x1e, 0x0a, 0x48, 0x6f, 0x14, 0x86, 0xc5, 0xcd, 0x9a, 0x98, 0x09, 0x2d, 0xf3, 0xf3, 0x5a, 0x7a, 0xa4, 0xe6, 0x8a, 0x2e, 0x49, 0x8a, 0xde, 0x73, 0xe9, 0x37, 0xa0, 0x5b, 0xef,
0xd0, 0xe0, 0x13, 0xac, 0x88, 0x5f, 0x59, 0x47, 0x96, 0x7f, 0x78, 0x18, 0x0e, 0x44, 0x6a, 0x5d, 0xec, 0x6e, 0xed, 0x4f, 0xf6, 0x6a, 0x7a,
0x58, 0x6b, 0xfe, 0x6c, 0x5a, 0xb9, 0xd2, 0x22, 0x3a, 0x1f, 0xdf, 0xc3, 0x09, 0x3f, 0x6b, 0x2e, 0xf1, 0x6d, 0xc3, 0xfb, 0x4e, 0xd4, 0xf2,
0xa3, 0x94, 0x13, 0xb0, 0xbf, 0x1e, 0x06, 0x2e, 0x29, 0x55, 0x00, 0xaa, 0x98, 0xd9, 0xe8, 0x77, 0x84, 0x8b, 0x3f, 0x5f, 0x5e, 0xf7, 0xf8,
0xa7, 0xe6, 0x02, 0xd2, 0x18, 0xb0, 0x52, 0xd0, 0x37, 0x2e, 0x53, 0x02, 0x03, 0x01, 0x00, 0x01, 0x02, 0x82, 0x01, 0x00, 0x0c, 0xdf, 0xd1,
0xe8, 0xf1, 0x9c, 0xc2, 0x9c, 0xd7, 0xf4, 0x73, 0x98, 0xf4, 0x87, 0xbd, 0x8d, 0xb2, 0xe1, 0x01, 0xf8, 0x9f, 0xac, 0x1f, 0x23, 0xdd, 0x78,
0x35, 0xe2, 0xd6, 0xd1, 0xf3, 0x4d, 0xb5, 0x25, 0x88, 0x16, 0xd1, 0x1a, 0x18, 0x33, 0xd6, 0x36, 0x7e, 0xc4, 0xc8, 0xe5, 0x5d, 0x2d, 0x74,
0xd5, 0x39, 0x3c, 0x44, 0x5a, 0x74, 0xb7, 0x7c, 0x48, 0xc1, 0x1f, 0x90, 0xe3, 0x55, 0x9e, 0xf6, 0x29, 0xad, 0xb4, 0x6d, 0x93, 0x78, 0xb3,
0xdc, 0x25, 0x0b, 0x9c, 0x73, 0x78, 0x7b, 0x93, 0x4c, 0xd3, 0x47, 0x09, 0xda, 0xe6, 0x69, 0x18, 0xc6, 0x0f, 0xfb, 0xa5, 0x95, 0xf5, 0xe8,
0x75, 0xe1, 0x01, 0x1b, 0xd3, 0x1c, 0xa2, 0x57, 0x03, 0x64, 0xdb, 0xf9, 0x5d, 0xf3, 0x3c, 0xa7, 0xd1, 0x4b, 0xb0, 0x90, 0x1b, 0x90, 0x62,
0xb4, 0x88, 0x30, 0x4b, 0x40, 0x4d, 0xcf, 0x7d, 0x89, 0x7a, 0xfb, 0x29, 0xc9, 0x64, 0x34, 0x0a, 0x52, 0xf6, 0x70, 0x7c, 0x76, 0x5a, 0x2e,
0x8f, 0x50, 0xd4, 0x92, 0x15, 0x97, 0xed, 0x4c, 0x2e, 0xf2, 0x3a, 0xd0, 0x58, 0x7e, 0xdb, 0xf1, 0xf4, 0xdd, 0x07, 0x76, 0x04, 0xf0, 0x55,
0x8b, 0x72, 0x2b, 0xa7, 0xa8, 0x78, 0x78, 0x67, 0xe6, 0xd8, 0xa5, 0xde, 0xe7, 0xc9, 0x1f, 0x5a, 0xa0, 0x89, 0xc7, 0x24, 0xa2, 0x71, 0xb6,
0x7b, 0x3b, 0xe6, 0x92, 0x69, 0x22, 0xaa, 0xe2, 0x47, 0x4b, 0x80, 0x3f, 0x6a, 0xab, 0xce, 0x4e, 0xcd, 0xe8, 0x94,

0x6c, 0xf7, 0x84, 0x73,
0x85, 0xfd, 0x85, 0x1d, 0xae, 0x81, 0xf7, 0xec, 0x12, 0x31, 0x7d, 0xc1, 0x99, 0xc0, 0x3c, 0x51, 0xb0, 0xdc, 0xb0, 0xba, 0x9c, 0x84, 0xb8,
0x70, 0xc2, 0x09, 0x7f, 0x96, 0x3d, 0xa1, 0xe2, 0x64, 0x27, 0x7a, 0x22, 0xb8, 0x75, 0xb9, 0xd1, 0x5f, 0xa5, 0x23, 0xf9, 0x62, 0xe0, 0x41,
0x02, 0x81, 0x81, 0x00, 0xf4, 0xf3, 0x08, 0xcf, 0x83, 0xb0, 0xab, 0xf2, 0x0f, 0x1a, 0x08, 0xaf, 0xc2, 0x42, 0x29, 0xa7, 0x9c, 0x5e, 0x52,
0x19, 0x69, 0x8d, 0x5b, 0x52, 0x29, 0x9c, 0x06, 0x6a, 0x5a, 0x32, 0x8f, 0x08, 0x45, 0x6c, 0x43, 0xb5, 0xac, 0xc3, 0xbb, 0x90, 0x7b, 0xec,
0xbb, 0x5d, 0x71, 0x25, 0x82, 0xf8, 0x40, 0xbf, 0x38, 0x00, 0x20, 0xf3, 0x8a, 0x38, 0x43, 0xde, 0x04, 0x41, 0x19, 0x5f, 0xeb, 0xb0, 0x50,
0x59, 0x10, 0xe1, 0x54, 0x62, 0x5c, 0x93, 0xd9, 0xdc, 0x63, 0x24, 0xd0, 0x17, 0x00, 0xc0, 0x44, 0x3e, 0xfc, 0xd1, 0xda, 0x4b, 0x24, 0xf7,
0xcb, 0x16, 0x35, 0xe6, 0x9f, 0x67, 0x96, 0x5f, 0xb0, 0x94, 0xde, 0xfa, 0xa1, 0xfd, 0x8c, 0x8a, 0xd1, 0x5c, 0x02, 0x8d, 0xe0, 0xa0, 0xa0,
0x02, 0x1d, 0x56, 0xaf, 0x13, 0x3a, 0x65, 0x5e, 0x8e, 0xde, 0xd1, 0xa8, 0x28, 0x8b, 0x71, 0xc9, 0x65, 0x02, 0x81, 0x81, 0x00, 0xc1, 0x0a,
0x47, 0x39, 0x91, 0x06, 0x1e, 0xb9, 0x43, 0x7c, 0x9e, 0x97, 0xc5, 0x09, 0x08, 0xbc, 0x22, 0x47, 0xe2, 0x96, 0x8e, 0x1c, 0x74, 0x80, 0x50,
0x6c, 0x9f, 0xef, 0x2f, 0xe5, 0x06, 0x3e, 0x73, 0x66, 0x76, 0x02, 0xbd, 0x9a, 0x1c, 0xfc, 0xf9, 0x6a, 0xb8, 0xf9, 0x36, 0x15, 0xb5, 0x20,
0x0b, 0x6b, 0x54, 0x83, 0x9c, 0x86, 0xba, 0x13, 0xb7, 0x99, 0x54, 0xa0, 0x93, 0x0d, 0xd6, 0x1e, 0xc1, 0x12, 0x72, 0x0d, 0xea, 0xb0, 0x14,
0x30, 0x70, 0x73, 0xef, 0x6b, 0x4c, 0xae, 0xb6, 0xff, 0xd4, 0xbb, 0x89, 0xa1, 0xec, 0xca, 0xa6, 0xe9, 0x95, 0x56, 0xac, 0xe2, 0x9b, 0x97,
0x2f, 0x2c, 0xdf, 0xa3, 0x6e, 0x59, 0xff, 0xcd, 0x3c, 0x6f, 0x57, 0xcc, 0x6e, 0x44, 0xc4, 0x27, 0xbf, 0xc3, 0xdd, 0x19, 0x9e, 0x81, 0x16,
0xe2, 0x8f, 0x65, 0x34, 0xa7, 0x0f, 0x22, 0xba, 0xbf, 0x79, 0x57, 0x02, 0x81, 0x80, 0x2e, 0x21, 0x0e, 0xc9, 0xb5, 0xad, 0x31, 0xd4, 0x76,
0x0f, 0x9b, 0x0f, 0x2e, 0x70, 0x33, 0x54, 0x03, 0x58, 0xa7, 0xf1, 0x6d, 0x35, 0x57, 0xbb, 0x53, 0x66, 0xb4, 0xb6, 0x96, 0xa1, 0xea, 0xd9,
0xcd, 0xe9, 0x23, 0x9f, 0x35, 0x17, 0xef, 0x5c, 0xb8, 0x59, 0xce, 0xb7, 0x3c, 0x35, 0xaa, 0x42, 0x82, 0x3f, 0x00, 0x96, 0xd5, 0x9d, 0xc7,
0xab, 0xec, 0xec, 0x04, 0xb5, 0x15, 0xc8, 0x40, 0xa4, 0x85, 0x9d, 0x20, 0x56, 0xaf, 0x03, 0x8f, 0x17, 0xb0, 0xf1, 0x96, 0x22, 0x3a, 0xa5,
0xfa, 0x58, 0x3b, 0x01, 0xf9, 0xae, 0xb3, 0x83, 0x6f, 0x44, 0xd3, 0x14, 0x2d, 0xb6, 0x6e, 0xd2, 0x9d, 0x39, 0x0c,


```
0x12, 0x1d, 0x23, 0xea,
 0x19, 0xcb, 0xbb, 0xe0, 0xcd, 0x89, 0x15, 0x9a, 0xf5, 0xe4, 0xec, 0x41, 0x06, 0x30, 0x16, 0x58, 0xea, 0xfa, 0x31,
0xc1, 0xb8, 0x8e, 0x08,
 0x84, 0xaa, 0x3b, 0x19, 0x02, 0x81, 0x80, 0x70, 0x4c, 0xf8, 0x6e, 0x86, 0xed, 0xd6, 0x85, 0xd4, 0xba, 0xf4, 0xd0,
0x3a, 0x32, 0x2d, 0x40,
 0xb5, 0x78, 0xb8, 0x5a, 0xf9, 0xc5, 0x98, 0x08, 0xe5, 0xc0, 0xab, 0xb2, 0x4c, 0x5c, 0xa2, 0x2b, 0x46, 0x9b, 0x3e,
0xe0, 0x0d, 0x49, 0x50,
 0xbf, 0xe2, 0xa1, 0xb1, 0x86, 0x59, 0x6e, 0x7b, 0x76, 0x6e, 0xee, 0x3b, 0xb6, 0x6d, 0x22, 0xfb, 0xb1, 0x68, 0xc7,
0xec, 0xb1, 0x95, 0x9b,
 0x21, 0x0b, 0xb7, 0x2a, 0x71, 0xeb, 0xa2, 0xb2, 0x58, 0xac, 0x6d, 0x5f, 0x24, 0xd3, 0x79, 0x42, 0xd2, 0xf7, 0x35,
0xdc, 0xfc, 0x0e, 0x95,
 0x60, 0xb7, 0x85, 0x7f, 0xf9, 0x72, 0x8e, 0x4a, 0x11, 0xc3, 0xc2, 0x09, 0x40, 0x5c, 0x7c, 0x43, 0x12, 0x34, 0xac,
0x59, 0x99, 0x76, 0x34,
 0xcf, 0x20, 0x88, 0xb0, 0xfb, 0x39, 0x62, 0x3a, 0x9b, 0x03, 0xa6, 0x84, 0x2c, 0x03, 0x5c, 0x0c, 0xca, 0x33, 0x85,
0xf5, 0x02, 0x81, 0x80,
 0x56, 0x99, 0xe9, 0x17, 0xdc, 0x33, 0xe1, 0x33, 0x8d, 0x5c, 0xba, 0x17, 0x32, 0xb7, 0x8c, 0xbd, 0x4b, 0x7f, 0x42,
0x3a, 0x79, 0x90, 0xe3,
 0x70, 0xe3, 0x27, 0xce, 0x22, 0x59, 0x02, 0xc0, 0xb1, 0x0e, 0x57, 0xf5, 0xdf, 0x07, 0xbf, 0xf8, 0x4e, 0x10, 0xef,
0x2a, 0x62, 0x30, 0x03,
 0xd4, 0x80, 0xcf, 0x20, 0x84, 0x25, 0x66, 0x3f, 0xc7, 0x4f, 0x56, 0x8c, 0x1e, 0xe1, 0x18, 0x91, 0xc1, 0xfd, 0x71,
0x5f, 0x65, 0x9b, 0xe4,
 0x4f, 0xe0, 0x1a, 0x3a, 0xf8, 0xc1, 0x69, 0xdb, 0xd3, 0xbb, 0x8d, 0x91, 0xd1, 0x11, 0x4f, 0x7e, 0x91, 0x1b, 0xb4,
0x27, 0xa5, 0xab, 0x7c,
 0x7b, 0x76, 0xd4, 0x78, 0xfe, 0x63, 0x44, 0x63, 0x7e, 0xe3, 0xa6, 0x60, 0x4f, 0xb9, 0x55, 0x28, 0xba, 0xba, 0x83,
0x1a, 0x2d, 0x43, 0xd5,
 0xf7, 0x2e, 0xe0, 0xfc, 0xa8, 0x14, 0x9b, 0x91, 0x2a, 0x36, 0xbf, 0xc7, 0x14
 };
CK_BYTE
 knownRSA1Modulus[] = {
0xb8, 0xb5, 0x0f, 0x49, 0x46, 0xb5, 0x5d, 0x58, 0x04, 0x8e, 0x52, 0x59, 0x39, 0xdf, 0xd6,
0x29,
0x45, 0x6b, 0x6c, 0x96, 0xbb, 0xab, 0xa5, 0x6f, 0x72, 0x1b, 0x16, 0x96, 0x74, 0xd5, 0xf9,
0xb4,
0x41, 0xa3, 0x7c, 0xe1, 0x94, 0x73, 0x4b, 0xa7, 0x23, 0xff, 0x61, 0xeb, 0xce, 0x5a, 0xe7,
0x7f,
0xe3, 0x74, 0xe8, 0x52, 0x5b, 0xd6, 0x5d, 0x5c, 0xdc, 0x98, 0x49, 0xfe, 0x51, 0xc2, 0x7e,
```



```
0x8f,
0x3b, 0x37, 0x5c, 0xb3, 0x11, 0xed, 0x85, 0x91, 0x15, 0x92, 0x24, 0xd8, 0xf1, 0x7b, 0x3d,
0x2f,
0x8b, 0xcd, 0x1b, 0x30, 0x14, 0xa3, 0x6b, 0x1b, 0x4d, 0x27, 0xff, 0x6a, 0x58, 0x84, 0x9e,
0x79,
0x94, 0xca, 0x78, 0x64, 0x01, 0x33, 0xc3, 0x58, 0xfc, 0xd3, 0x83, 0xeb, 0x2f, 0xab, 0x6f,
0x85,
0x5a, 0x38, 0x41, 0x3d, 0x73, 0x20, 0x1b, 0x82, 0xbc, 0x7e, 0x76, 0xde, 0x5c, 0xfe, 0x42,
0xd6,
0x7b, 0x86, 0x4f, 0x79, 0x78, 0x29, 0x82, 0x87, 0xa6, 0x24, 0x43, 0x39, 0x74, 0xfe, 0xf2,
0x0c,
0x08, 0xbe, 0xfa, 0x1e, 0x0a, 0x48, 0x6f, 0x14, 0x86, 0xc5, 0xcd, 0x9a, 0x98, 0x09, 0x2d,
0xf3,
0xf3, 0x5a, 0x7a, 0xa4, 0xe6, 0x8a, 0x2e, 0x49, 0x8a, 0xde, 0x73, 0xe9, 0x37, 0xa0, 0x5b,
0xef,
0xd0, 0xe0, 0x13, 0xac, 0x88, 0x5f, 0x59, 0x47, 0x96, 0x7f, 0x78, 0x18, 0x0e, 0x44, 0x6a,
0x5d,
0xec, 0x6e, 0xed, 0x4f, 0xf6, 0x6a, 0x7a, 0x58, 0x6b, 0xfe, 0x6c, 0x5a, 0xb9, 0xd2, 0x22,
0x3a,
0x1f, 0xdf, 0xc3, 0x09, 0x3f, 0x6b, 0x2e, 0xf1, 0x6d, 0xc3, 0xfb, 0x4e, 0xd4, 0xf2, 0xa3,
0x94,
0x13, 0xb0, 0xbf, 0x1e, 0x06, 0x2e, 0x29, 0x55, 0x00, 0xaa, 0x98, 0xd9, 0xe8, 0x77, 0x84,
0x8b,
0x3f, 0x5f, 0x5e, 0xf7, 0xf8, 0xa7, 0xe6, 0x02, 0xd2, 0x18, 0xb0, 0x52, 0xd0, 0x37, 0x2e,
0x53,
    },
    knownRSA1PubExponent[] = { 0x01, 0x00, 0x01 };
    char *pPlainData = 0;
    unsigned long ulPlainDataLength;
    char *pEncryptedData = 0;
    unsigned long ulEncryptedDataLength = 0;
    CK_MECHANISM mech;
    CK_USHORT usStatus=0,
            usKeyLength;
    CK_OBJECT_HANDLE hKey;
    CK_OBJECT_CLASS SymKeyClass = CKO_SECRET_KEY;
```



```
    CK_BBOOL bTrue = 1,
                bFalse = 0,
                bToken = bTrue,
                bSensitive = bTrue,
                bPrivate = bTrue,
                bEncrypt = bTrue,
                bDecrypt = bTrue,
                bSign = bFalse, // "..."
                bVerify = bFalse, //Will not allow sign/verify operation.
                bWrap = bTrue,
                bUnwrap = bTrue,
#ifdef EXTRACTABLE
                bExtract = bTrue,
#endif //EXTRACTABLE
                bDerive = bTrue;
    CK_KEY_TYPE keyType;
    CK_USHORT usValueBits;
    char pbPublicKeyLabel[128];
    CK_ATTRIBUTE_PTR pPublicTemplate;
    CK_USHORT usPublicTemplateSize = 0;
    char iv[8] = { '1', '2', '3', '4', '5', '6', '7', '8' };
    CK_ATTRIBUTE SymKeyTemplate[] = {
      {CKA_CLASS, 0, sizeof(SymKeyClass)},
      {CKA_KEY_TYPE, 0, sizeof(keyType)},
      {CKA_TOKEN, 0, sizeof(bToken)},
      {CKA_SENSITIVE, 0, sizeof(bSensitive)},
      {CKA_PRIVATE, 0, sizeof(bPrivate)},
      {CKA_ENCRYPT, 0, sizeof(bEncrypt)},
      {CKA_DECRYPT, 0, sizeof(bDecrypt)},
      {CKA_SIGN, 0, sizeof(bSign)},
      {CKA_VERIFY, 0, sizeof(bVerify)},
      {CKA_WRAP, 0, sizeof(bWrap)},
      {CKA_UNWRAP, 0, sizeof(bUnwrap)},
      {CKA_DERIVE, 0, sizeof(bDerive)},
      {CKA_VALUE_LEN,0, sizeof(usKeyLength) },
      {CKA_LABEL, 0, 0}  // Always keep last!!!
```



```
#ifdef EXTRACTABLE     //Conditional stuff must be at the end!!!!!
      {CKA_EXTRACTABLE, 0, sizeof(bExtract)},
#endif //EXTRACTABLE
    };
    CK_OBJECT_HANDLE hUnWrappedKey, hPublicRSAKey;
    char *pbWrappedKey;
    unsigned long    ulWrappedKeySize;
    CK_OBJECT_CLASS privateKey = CKO_PRIVATE_KEY,
publicKey = CKO_PUBLIC_KEY;
    CK_KEY_TYPE rsaType = CKK_RSA;
    CK_BYTE pLabel[] = "RSA private Key",
pbPublicRSAKeyLabel[] = "RSA Public Key";
    CK_ATTRIBUTE *pTemplate;
    CK_ULONG usTemplateSize,
ulPublicRSAKeyTemplateSize;
    CK_ATTRIBUTE pPublicRSAKeyTemplate[] = {
      {CKA_CLASS, 0, sizeof(publicKey) },
      {CKA_KEY_TYPE, 0,    sizeof(rsaType) },
      {CKA_TOKEN, 0, sizeof(bToken) },
      {CKA_PRIVATE, 0,   sizeof(bPrivate) },
      {CKA_ENCRYPT, 0,   sizeof(bEncrypt) },
      {CKA_VERIFY, 0, sizeof(bSign) },
      {CKA_WRAP, 0, sizeof(bWrap) },
 {CKA_MODULUS, 0, sizeof(knownRSA1Modulus) },
 {CKA_PUBLIC_EXPONENT, 0, sizeof(knownRSA1PubExponent) },
      {CKA_LABEL, 0, sizeof(pbPublicRSAKeyLabel) }
    };
    CK_ATTRIBUTE pPrivateKeyTemplate[] = {
      {CKA_CLASS, &privateKey, sizeof(privateKey) },
      {CKA_KEY_TYPE, &rsaType,    sizeof(rsaType) },
      {CKA_TOKEN, &bToken, sizeof(bToken) },
      {CKA_SENSITIVE,&bSensitive, sizeof(bSensitive) },
      {CKA_PRIVATE, &bPrivate, sizeof(bPrivate) },
      {CKA_DECRYPT, &bEncrypt, sizeof(bEncrypt) },
      {CKA_SIGN, &bSign, sizeof(bSign) },
 //{CKA_SIGN_RECOVER, &bTrue, sizeof(bTrue)    },
```



```
        {CKA_UNWRAP, &bWrap, sizeof(bWrap) },
{CKA_EXTRACTABLE, &bFalse, sizeof(bFalse)     },
{CKA_LABEL,      pLabel, sizeof(pLabel) }
    };
    // Generate a DES3 Key
    SymKeyTemplate[0].pValue = &SymKeyClass;
    SymKeyTemplate[1].pValue = &keyType;
    SymKeyTemplate[2].pValue = &bToken;
    SymKeyTemplate[3].pValue = &bSensitive;
    SymKeyTemplate[4].pValue = &bPrivate;
    SymKeyTemplate[5].pValue = &bEncrypt;
    SymKeyTemplate[6].pValue = &bDecrypt;
    SymKeyTemplate[7].pValue = &bSign;
    SymKeyTemplate[8].pValue = &bVerify;
    SymKeyTemplate[9].pValue = &bWrap;
    SymKeyTemplate[10].pValue = &bUnwrap;
    SymKeyTemplate[11].pValue = &bDerive;
    SymKeyTemplate[12].pValue =  &usKeyLength;
    SymKeyTemplate[13].pValue = pbPublicKeyLabel;
#ifdef EXTRACTABLE
    SymKeyTemplate[14].pValue = &bExtract;
#endif //EXTRACTABLE
    mech.mechanism = CKM_DES3_KEY_GEN;
    mech.pParameter = 0;
    mech.usParameterLen = 0;
    keyType = CKK_DES3;
    usKeyLength = 24;
    strcpy( pbPublicKeyLabel, "Generated DES3 Key" );
    pPublicTemplate = SymKeyTemplate;
    usPublicTemplateSize = DIM(SymKeyTemplate);
    // Adjust size of label (ALWAYS LAST ENTRY IN ARRAY)
    pPublicTemplate[usPublicTemplateSize-1].usValueLen = strlen(
pbPublicKeyLabel );
    retCode = C_GenerateKey(   hSessionHandle,
                        (CK_MECHANISM_PTR)&mech,
                        pPublicTemplate,
```



```
                    usPublicTemplateSize,
                    &hKey);
    if(retCode == CKR_OK)
    {
      cout << pbPublicKeyLabel << ": " << hKey << endl;
    }
    else
    {
  cout << "\n" "Error 0x" << hex << retCode;
      cout << " generating the DES3 Key.\n";
  error = -11;
      goto exit_routine_6;
    }
    // Encrypt the RSA Key
    mech.mechanism = CKM_DES3_CBC;
    mech.pParameter = iv;
    mech.usParameterLen = sizeof(iv);
    pPlainData = (char *)(pRsaKey);
    ulPlainDataLength = sizeof(pRsaKey);
    // Allocate memory for output buffer
    if( retCode == CKR_OK )
    {
      pEncryptedData = new char [ulPlainDataLength + 2048]; // Leave
// extra room for
// RSA Operations
      if( !pEncryptedData )
      {
        retCode = CKR_DEVICE_ERROR;
      }
    }
    // Start encrypting
    if( retCode == CKR_OK )
    {
      retCode = C_EncryptInit(hSessionHandle, &mech, hKey);
    }
    // Continue encrypting
```



```
if( retCode == CKR_OK )
{
  CK_USHORT usInDataLen,
        usOutDataLen = (CK_USHORT) (ulPlainDataLength + 2048);
  CK_ULONG ulBytesRemaining = ulPlainDataLength;
  char *   pPlainTextPointer = pPlainData;
  char *   pEncryptedDataPointer = pEncryptedData;
  while (ulBytesRemaining > 0)
  {
    if (ulBytesRemaining > 0xfff0) // We are longer than a USHORT can handle
    {
      usInDataLen = 0xfff0;
      ulBytesRemaining -= usInDataLen;
    }
    else
    {
      usInDataLen = (CK_USHORT) ulBytesRemaining;
      ulBytesRemaining -= usInDataLen;
    }
    retCode = C_EncryptUpdate( hSessionHandle,
                  (CK_BYTE_PTR)pPlainTextPointer,
                  usInDataLen,
                  (CK_BYTE_PTR)pEncryptedDataPointer,
                  &usOutDataLen );
    pPlainTextPointer += usInDataLen;
    pEncryptedDataPointer += usOutDataLen;
    ulEncryptedDataLength += usOutDataLen;
  }
}

// Finish encrypting
if( retCode == CKR_OK )
{
  CK_USHORT usOutDataLen;
  CK_BYTE_PTR pOutData = (CK_BYTE_PTR)pEncryptedData;
  pOutData += ulEncryptedDataLength;
```

```
    retCode = C_EncryptFinal(hSessionHandle, pOutData, &usOutDataLen);
    ulEncryptedDataLength += usOutDataLen;
  }
  else
  {
cout << "\n" "Error 0x" << hex << retCode;
    cout << " somewhere in the encrypting.\n";
    if( pEncryptedData )
    {
      delete pEncryptedData;
    }
error = -12;
    goto exit_routine_6;
  }
  mech.mechanism = CKM_DES3_CBC;
  mech.pParameter = (void*) "12345678"; // 8 byte IV
  mech.usParameterLen = 8;
  pTemplate = pPrivateKeyTemplate;
  usTemplateSize = DIM(pPrivateKeyTemplate);
 pbWrappedKey = pEncryptedData;
  ulWrappedKeySize = ulEncryptedDataLength;
  if( retCode == CKR_OK )
  {
    retCode = C_UnwrapKey( hSessionHandle,
                  &mech,
                  hKey,
                  (CK_BYTE_PTR)pbWrappedKey,
                  (CK_USHORT)ulWrappedKeySize,
                  pTemplate,
                  usTemplateSize,
                  &hUnWrappedKey);
  }
  // Report unwrapped key handle
  if( retCode == CKR_OK )
  {
    cout << "\n Private key Unwrapped key is:" << hUnWrappedKey <<"\n\n";
```

```
    }
    else
    {
  cout << "\n" "Error 0x" << hex << retCode;
       cout << " unwrapping.\n";
       if( pEncryptedData )
       {
         delete pEncryptedData;
       }
  error = -13;
       goto exit_routine_6;
    }
    // Release temporary memory
    if( pEncryptedData )
    {
      delete pEncryptedData;
    }
  // Create the Public Key that goes with the Private Key
  if( retCode == CKR_OK )
  {
 // Unwrap it onto the token
  pPublicRSAKeyTemplate[0].pValue = &publicKey;
  pPublicRSAKeyTemplate[1].pValue = &rsaType;
  pPublicRSAKeyTemplate[2].pValue = &bToken;
  pPublicRSAKeyTemplate[3].pValue = &bPrivate;
  pPublicRSAKeyTemplate[4].pValue = &bEncrypt;
  pPublicRSAKeyTemplate[5].pValue = &bSign;
  pPublicRSAKeyTemplate[6].pValue = &bWrap;
  pPublicRSAKeyTemplate[7].pValue = knownRSA1Modulus;
  pPublicRSAKeyTemplate[8].pValue = knownRSA1PubExponent;
  pPublicRSAKeyTemplate[9].pValue = pbPublicRSAKeyLabel;
  pTemplate = pPublicRSAKeyTemplate;
  usTemplateSize = DIM(pPublicRSAKeyTemplate);
  retCode = C_CreateObject(  hSessionHandle,
pTemplate,
 usTemplateSize,
```



```
&hPublicRSAKey);
 if(retCode == CKR_OK)
 {
cout << pbPublicRSAKeyLabel << ": " << hPublicRSAKey << endl;
 }
 else
 {
cout << "\n" "Error 0x" << hex << retCode;
cout << " creating the RSA Public Key.\n";
error = -14;
goto exit_routine_6;
 }
  }
if( retCode == CKR_OK )
  {
CK_CHAR label[] = "RSA Key";
CK_ATTRIBUTE RSAFindPriTemplate[] =
{
CKA_LABEL, label, sizeof(label)
};
CK_ULONG numHandles;
CK_OBJECT_HANDLE handles[1000];
retCode = C_FindObjectsInit( hSessionHandle, RSAFindPriTemplate, 1 );
if(retCode != CKR_OK)
{
cout << "C_FindObjectsInit not returning OK (" << hex << retCode << ")\n\n";
goto exit_routine_6;
}
retCode =C_FindObjects( hSessionHandle , handles, 90,
&numHandles );
if(retCode != CKR_OK)
{
cout << "C_FindObjects not returning OK (" << hex <<
retCode << ")\n\n";
goto exit_routine_6;
}
```



```
cout << "Everything's GOOD\n\n";
for(int i=0; i < numHandles; i++)
{
cout << handles[i] << "\n";
}
}
}
//CJM-> END OF TEST CODE
  // Beginning of exit routines
exit_routine_6:
  // Logout
  retCode = C_Logout(hSessionHandle);
  if(retCode != CKR_OK)
{
cout << "\n" "Error 0x" << hex << retCode << " logging out.";
}
exit_routine_5:
// Close the session
  retCode = C_CloseSession(hSessionHandle);
if(retCode != CKR_OK)
{
cout << "\n" "Error 0x" << hex << retCode << " closing session.";
}
exit_routine_4:
  delete pSlotList;
exit_routine_3:
#ifdef PKCS11_2_0
  C_Finalize(0);
#else
  C_Terminate();
#endif
exit_routine_2:
#ifndef STATIC
  // No longer need Chrystoki
  CrystokiDisconnect();
#endif
```



```
exit_routine_1:
  cout << "\nDone. (" << dec << error << ")\n";
  cout.flush();
  return error;
}
CK_RV  Pinlogin(CK_SESSION_HANDLE hSession)
{

CK_RV retCode;
unsigned char buffer[MAX];
int count =0;
cout << "Please enter the USER password : " << endl;
//calling get PinString to mask input, variable "count"
//holds length of "buffer"(password)
//needed for Login call
count = getPinString(buffer);
//Login as user on  token in slot
retCode = C_Login(hSession, CKU_USER, buffer, count);
if(retCode != CKR_OK)
{
cout << "\n" "Error 0x" << hex << retCode;
     cout << " logging in as user.";
 exit(hSession);
 return -3;
}
cout << "logging into the token....";
cout << "\nlogged into token " << endl;
return retCode;
}
//////////////////////////////////////////////////////////////////////////
// getPinString()
// ==============
//
// This function retrieves a pin string from the user.  It modifies the
// console mode before starting so that the characters the user types are
// not echoed, and a '*' character is displayed for each typed character
```



```
// instead.
//
// Backspace is supported, but we don't get any fancier than that.
//////////////////////////////////////////////////////////////////////////
int getPinString(CK_CHAR_PTR pw)
{
   int len=0;
   char c=0;
   // Unfortunately, the method of turning off character echo is
// different for Windows and Unix platforms.  So we have to
// conditionally compile the appropriate section.  Even the basic
// password retrieval is slightly different, since
   // Windows and Unix use different character codes for the return key.
#ifdef WIN32
   DWORD mode;
   // This console mode stuff only applies to windows.  We'll have to
// do something else when it comes to unix.
   if (GetConsoleMode(GetStdHandle(STD_INPUT_HANDLE), &mode)) {
      if (SetConsoleMode(GetStdHandle(STD_INPUT_HANDLE), mode & (!ENABLE_ECHO_INPUT))) {
         while (c  != '\r') {
            // wait for a character to be hit
            while (!_kbhit()) {
               Sleep(100);
            }
            // get it
            c = _getch();
            // check for carriage return
            if (c != '\r') {
               // check for backspace
               if (c!='\b') {
                  // neither CR nor BS -- add it to the password string
                  printf("*");
                  *pw++ = c;
                  len++;
               } else {
// handle backspace -- delete the last character &
```



```
// erase it from the screen
                if (len > 0) {
                  pw--;
                  len--;
                  printf("\b \b");
}
}
}
}
        // Add the zero-termination
        *pw = '\0';
        SetConsoleMode(GetStdHandle(STD_INPUT_HANDLE), mode);
        printf("\n");
}
  }
#endif
  return len;
}
```

Audit Logging

By default, the HSM logs select events to the file hsm.log.

For more robust and verifiable logging, SafeNet HSM (after version 5.2) includes the Audit Logging feature, to log select HSM events to files that can be securely verified for audit purposes.

The HSM creates a log secret unique to the HSM, computed during the first initialization after manufacture. The log secret resides in flash memory (permanent, non-volatile memory), and is used to create log records that are sent to a log file. Later, the log secret is used to prove that a log record originated from a legitimate HSM and has not been tampered with.

Audit Log Records

A log record consists of two fields – the log message and the HMAC for the previous record. When the HSM creates a log record, it uses the log secret to compute the SHA256-HMAC of all data contained in that log message, plus the HMAC of the previous log entry. The HMAC is stored in HSM flash memory. The log message is then transmitted, along with the HMAC of the previous record, to the host. The host has a logging daemon to receive and store the log data on the host hard drive.

For the first log message ever returned from the HSM to the host there is no previous record and, therefore, no HMAC in flash. In this case, the previous HMAC is set to zero and the first HMAC is computed over the first log message concatenated with 32 zero-bytes. The first record in the log file then consists of the first log message plus 32 zero-bytes. The second record consists of the second message plus HMAC1 = HMAC (message1 || 0x0000). This results in the organization shown below.

MSG 1	HMAC 0
	. . .
MSG n-1	HMAC n-2
MSG n	HMAC n-1
. . .	
MSG n+m	HMAC n+m-1
MSG n+m+1	HMAC n+m
. . .	
MSG end	HMAC n+m-1

Recent HMAC in NVRAM	HMAC end

To verify a sequence of *m* log records which is a subset of the complete log, starting at index *n*, the host must submit the data illustrated above. The HSM calculates the HMAC for each record the same way as it did when the record was originally generated, and compares this HMAC to the value it received. If all of the calculated HMACs match the received HMACs, then the entire sequence verifies. If an HMAC doesn't match, then the associated record and all following records can be considered suspect. Because the HMAC of each message depends on the HMAC of the previous one, inserting or altering messages would cause the calculated HMAC to be invalid.

The HSM always stores the HMAC of the most-recently generated log message in flash memory. When checking truncation, the host would send the newest record in its log to the HSM; and, the HSM would compute the HMAC and compare it to the one in flash. If it does not match, then truncation has occurred.

Audit Log Message Format

Each message is a fixed-length, comma delimited, and newline-terminated string. The table below shows the width and meaning of the fields in a message.

Offset	Length (Chars)	Description
0	10	Sequence number
10	1	Comma
11	17	Timestamp
28	1	Comma
29	256	Message text, interpreted from raw data
285	1	Comma
286	64	HMAC of previous record as ASCII-HEX

Offset	Length (Chars)	Description
350	1	Comma
351	96	Data for this record as ASCII-HEX (raw data)
447	1	Newline '\n'

The raw data for the message is stored in ASCII-HEX form, along with a human-readable version. Although this format makes the messages larger, it simplifies the verification process, as the HSM expects to receive raw data records.

Example

The following example shows a sample log record. It is separated into multiple lines for readability even though it is a single record. Some white spaces are also omitted.

```
38,12/08/13 15:30:50,session 1 Access 2147483651:22621 operation LUNA_CREATE_CONTAINER
returned LUNA_RET_SM_UNKNOWN_TOSM_STATE(0x00300014) (using PIN (entry=LUNA_ENTRY_DATA_AREA)),
29C51014B6F131EC67CF48734101BBE301335C25F43EDF8828745C40755ABE25,
2600001003600B00EA552950140030005D5800000300008001000000000000000000000000000000000000
```

The sequence number is “38”. The time is “12/08/13 15:30:50”.

The log message is “session 1 Access 2147483651:22621 operation LUNA_CREATE_CONTAINER returned LUNA_RET_SM_UNKNOWN_TOSM_STATE(0x00300014) (using PIN (entry=LUNA_ENTRY_DATA_AREA))”. In the message text, the “who” is the session identified by “session 1 Access 2147483651:22621” (the application is identified by the access ID major = 2147483651, minor = 22621). The “what” is “LUNA_CREATE_CONTAINER”. The operation status is “LUNA_RET_SM_UNKNOWN_TOSM_STATE(0x00300014)”.

The HMAC of previous record is “29C51014B6F131EC67CF48734101BBE301335C25F43EDF8828745C40755ABE25”.

The remainder is the raw data for this record as ASCII-HEX.

Log External

An important element of the security audit logging feature is the ‘Log External’ function. This SafeNet extension to PKCS #11 allows a user application to insert text of the user’s choice into the log record stream. The function call is **CA_LogExternal ( )**. It can be used, for example, to insert an application name or the name of the user who is logged into the application and have the inserted text string protected as part of the audit log in the same way as records that have been generated by the HSM itself. It is recommended that applications use the **CA_LogExternal ( )** function when the application starts to insert the application name and also to insert the user name each time an individual user logs into or out of the application. The function is called as:

CA_LogExternal(CK_SLOT_ID slotID, CK_SESSION_HANDLE hSession, CK_CHAR_PTR pData, CK_ULONG puldataLen);

where:

- **slotID** is PKCS #11 slot containing the HSM or partition being addressed,
- **hSession** is the handle of the session with which the record is to be associated,
- **pData** is the pointer to the character array containing the external message and
- **puldataLen** is the length of the character array.

Note that the input character array is limited to a maximum of 100 characters and it will be truncated at 100 characters if **puldataLen** > 100.

For applications that cannot add this function call, it is possible to use the lunacm command-line function audit log external within a startup script to insert a text record at the time the application is started.

When a user logs in to the SafeNet Network HSM lunash:> session, the CA_LogExternal ( ) function is automatically called to register the user name and access ID. Subsequent HSM operations can be tracked by the access ID.

You must configure the “log external” event category in order for the HSM to log the CA_LogExternal() messages.

About Secure Identity Management

For customer applications involving large numbers of keys, that might exceed the internal flash-memory capacity of the SafeNet Network HSM K6 engine, support is provided for secure external storage of keys.

For the most part, SIM functionality must be supported by custom programming. Our Software Development Kit (available separately) includes documentation and samples for Cryptoki and Java APIs.

The following characteristics apply to the SIM capability:

- SIM is a purchased capability that must be enabled when your SafeNet Network HSM is manufactured. SIM cannot be implemented with a SafeNet Network HSM that was not explicitly enabled for SIM.
- The database-management aspects of large numbers of externally stored keys are beyond the scope of SafeNet Network HSM. SafeNet Network HSM ensures the security of those keys, without reference to their management and retrieval. Such management is the responsibility of the customer's application.
- All keys that are externally stored with this feature are strongly encrypted, using symmetric keys that are never exposed outside the HSM server. Additional encryption and security measures are employed within the HSM server to afford multiple levels of security.
- All manipulations of the keys take place within protected, volatile memory inside the SafeNet Network HSM K6 engine.

Note: Each SafeNet Network HSM leaving the factory has a unique masking key, which is used for Secure Identity Management. To give several SafeNet Enterprise HSMs the same masking key, choose one and perform hsm -backup. Then, using that Backup HSM, perform hsm -restore onto each SafeNet Network HSM that must share that masking key.

Note: When the HSM is initialized, a new masking secret is created. The new masking secret will be backed up onto a backup token if "hsm backup" is performed, but the **old** masking secret will continue to be used for all masking operations until the HSM is powered off.

A SafeNet Network HSM with SIM enabled can support only a single HSM Partition.

WARNING! If the masking key is lost, then all extracted key material (all the keys in your database) is effectively lost as well. Therefore, perform an HSM Backup, to backup the SIM Masking Key.

Secure Identity Management (SIM) APIs

Note: The SafeNet Network HSM HA feature and SIM can be used simultaneously in SafeNet Network HSM release 3.0 and later.

Applications use the following APIs to extract/insert keys under SIM. The multisign function call is an optimization that allows you to insert and sign (potentially) many objects at once.

```
CK_RV CK_ENTRY CA_ExtractMaskedObject(CK_SESSION_HANDLE hSession,
    CK_ULONG ulObjectHandle,
    CK_BYTE_PTR pMaskedKey,
    CK_USHORT_PTR pusMaskedKeyLen);
CK_RV CK_ENTRY CA_InsertMaskedObject( CK_SESSION_HANDLE hSession,
    CK_ULONG_PTR pulObjectHandle,
    CK_BYTE_PTR pMaskedKey,
    CK_USHORT usMaskedKeyLen);
CK_RV CK_ENTRY CA_MultisignValue( CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_ULONG ulMaskedKeyLen,
    CK_BYTE_PTR pMaskedKey,
    CK_ULONG_PTR pulBlobCount,
    CK_ULONG_PTR pulBlobLens,
    CK_BYTE_PTR CK_PTR ppBlobs,
    CK_ULONG_PTR pulSignatureLens,
    CK_BYTE_PTR CK_PTR ppSignatures);
```

The SafeNet Software Developers Kit contains example code in our ckdemo example program that shows how to use this API.

In general, the normal life cycle of a key pair is assumed to consist of the following steps:

- the key pair is generated
- the public exponent and modulus are extracted for the creation of a certificate (CA_ExtractMaskedObject)
- the keys are used (some number of times, over a period of years) for cryptographic operations

You can use **CA_MultisignValue** to perform signing operations on multiple objects at one time. **CA_MultisignValue** is a self-contained call that cleans up after itself by destroying the inserted key before exiting.

You can use **CA_InsertMaskedObject** to use the inserted key for other operations (such as encryption) that you would invoke via standard cryptoki calls. You must clean up by deleting the object when you have finished, to free the volatile memory that was used.

The external keys are destroyed (wiped from the database) when no longer needed.

SIM II (Enhancements to SIM)

SIM II provides enhancements to SIM for the Cyptoki API and the Java API, as described in the following sections:

Cryptoki API

Three forms of authorization data are supported:

- text-based PINs
- a challenge/response mechanism similar to the one used in SafeNet HSM (with Trusted Path Authentication) login
- a PED key mechanism similar to our legacy M-of-N activation for the HSM.

The form of authorization data is identified using the following definitions:

```
typedef CK_ULONG SIM_AUTHORIZATION_FORM;
#define SIM_AUTHORIZATION_PIN 0
#define SIM_AUTHORIZATION_CHALLENGE 1
#define SIM_AUTHORIZATION_PED 2
```

Three new API functions are added to cryptoki.h, as follows:

The CK_RV CA_SIMExtract function

```
CK_RV CA_SIMExtract(CK_ULONG handleCount, CK_ULONG *handleList,
   CK_ULONG authForm, CK_ULONG authDataCount, CK_ULONG subsetRequired,
   CK_BYTE **authDataList,
   CK_BOOL deleteAfterExtract,
   CK_ULONG *pBlobSize, CK_BYTE *pBlob );
```

This function takes a list of object handles, extracts them using the given authorization data for protection and returns the extracted set of objects as a single data blob. The objects are left on the partition or destroyed, based on the value of the delete-after-extract flag.

The **authDataCount** parameter defines the N value. The **subsetRequired** parameter defines the M value. The **authDataList** parameter should have N entries in it if it is used.

For an authorization data form of PED or challenge/response, **authDataList** parameter is null – values are defined through the PED.

The CK_RV SIMInsert function

```
CK_RV SIMInsert( CK_ULONG blobSize, CK_BYTE *pBlob,
   CK_ULONG authForm, CK_ULONG authDataCount, CK_BYTE **authDataList,
   CK_ULONG *pHandleListSize, CK_ULONG *pHandleList );
```

This function takes a previously extracted blob as input, validates the authorization data, inserts the objects contained in the blob into the HSM, and returns the list of handles assigned to the objects.

For an authorization data form of PED, the **authDataCount** and **authDataList** parameters are not used. For other authorization data forms, the **authDataCount** value should equal M, and the **authDataList** should have M elements in it.

The CK_RV SIMMultiSign function

```
CK_RV SIMMultiSign( CK_ULONG blobSize, CK_BYTE *pBlob,
   CK_ULONG authForm, CK_ULONG authDataCount, CK_BYTE **authDataList,
   CK_ULONG inputDataCount,
   CK_ULONG *inputDataLengths, CK_BYTE **inputDataTable,
   CK_ULONG *signatureLengths, CK_BYTE **signatureTable);
```

This function takes a previously extracted blob as input, validates the authorization data, then uses the key material in the given key blob to sign the various pieces of data in the input data table, returning the signatures through the signature table. The key blob must contain a single key, otherwise an error is returned.

The authorization data parameters are handled as for the SIMInsert function.

Java API

The standard java keystore API supports a single password for each keystore, and a single password for each key in the keystore. We provide a keystore implementation that stores key material in a file, using SIM to extract the key

material. The password on the keystore is not used, but the password for each key is used as authorization data for the SIM masking process.

When a key is stored in this type of keystore, it is extracted using SIM and the appropriate authorization data, but the key is left on the HSM. When a key is retrieved from this type of keystore, it is inserted onto the HSM.

The standard keystore API supports 1-of-1 authorization inputs of the text form. Different authorization data forms are supported through a custom API. The **LunaTokenManager** class is enhanced to provide a new method to allow the authorization data for subsequent keystore operations to be defined. If the password parameter of a keystore **SetKeyEntry** or **SetCertificateEntry** method call is given a null value, the actual authorization data will be taken from the **LunaTokenManager** interface.

Note that it is up to application to serialize calls to **LunaTokenManager** and the keystore object if multiple threads are simultaneously using keystores. That is, each thread must ensure that it sets its authorization data in **LunaTokenManager** and then performs its keystore operation without being interrupted by another thread changing the **LunaTokenManager** authorization data.

Example Operations Using CKDemo

The following examples show how to use the ckdemo utility to perform SIM operations.

Multisign Challenge (Trusted Path Authentication Only)

1. Open Ckdemo and login as user.
2. Create a 1024 bit RSA key pair - 45,7,1024,1,1,1,1,1,1,1,1,1
3. Sim Extract (105)
   Enter your choice : 105
   Enter handle of object to add to blob (0 to end list, -1 to cancel): 10
   Enter handle of object to add to blob (0 to end list, -1 to cancel): 0
   Enter authentication form:
   0 - none
   1 - password
   2 - challenge response
   3 - PED-based
   enter “2”
   Enter number of authorization secrets (N value): 3
   Enter subset size required for key use (M value): 2
4. The SafeNet PED displays your challenge secrets, be sure to record them.
   Delete after extract? [0 = false, 1 = true] : 1
5. For every instance of data to sign, enter “12345678”.
   The signatures should complete and be placed in a file.
6. Ensure that the private key has been extracted by performing CKDemo command 26,6 . This shows all the objects on the token. The private key handle that you noted earlier should not be there.
7. Now, insert the blobfile back onto the token:
   Select Insert masked object (106)
   Enter “simkey.blob” as the keyblob to be re-inserted
   Input 2 of the 3 challenges that you recorded earlier.
8. CKDemo 26,6 should reveal that the private key has been re-inserted.

SIM2 Multisign PED-based (PED/Trusted Path Configuration Only)

1. Open Ckdemo and login as user.
2. Create a 1024 bit RSA key pair - 45,7,1024,1,1,1,1,1,1,1,1,1. Note the private and public key handles.
3. Sim Extract (105)
   Enter your choice : 105
   Enter handle of object to add to blob (0 to end list, -1 to cancel): 10
   Enter handle of object to add to blob (0 to end list, -1 to cancel): 0
   Enter authentication form:
   3 - none
   4 - password
   5 - challenge response
   6 - PED-based
   Enter "3"
4. Delete after extract? [0 = false, 1 = true] : 1
5. For every instance of data to sign, enter "12345678".
   The signatures should complete and the key should be placed in the file simkey.blob.
6. Ensure that the private key has been extracted by performing a 26,6 . This will show all the objects on the token. The private key handle that you noted earlier should not be there.
7. Now, insert the blobfile back onto the token:
   Select Insert masked object (106)
   Enter "simkey.blob" as the keyblob to be re-inserted.
   Input 2 of the 3 challenges that you recorded earlier.
8. CKDemo command 26,6 should reveal that the private key has been re-inserted.

Using SIM in a Multi-HSM Environment

Here are the basic steps to follow when setting up to use SIM with two SafeNet appliance units.

1. Initialize the first SafeNet appliance. Refer to the Configuration section of this Help. The domain created during this initialization (a text string for Password Authenticated SafeNet appliance, or a red PED Key for PED Authenticated SafeNet appliance) will be used as the domain for backup tokens and for the second SafeNet appliance.
2. Create the partition on the first SafeNet appliance.
3. Connect the backup HSM to the appliance USB port.
4. Insert the token into SafeNet Dock2, which is connected to the appliance USB port.
5. Initialize the backup HSM or token using token backup init lush command, with the same domain. Follow the on-screen prompts. Use the domain from step 1.
6. Initialize the second SafeNet appliance. Use the same cloning domain as was used on the first SafeNet appliance .
7. Create the partition on the second SafeNet appliance.
8. Connect the backup HSM to the appliance USB port.
9. Insert the token into SafeNet Dock2, which is connected to the appliance USB port.
10. Perform `hsm restore` from the admin shell. Once this is completed, you now have both SafeNet appliances able to mask and unmask keys using the same "master" key.

11. Set up your Clients and register both SafeNet appliances with each Client. In ckdemo, if you select option 14 (Slot List) and select “Only slots with token present”, you should see two LunaNet slots.
12. When the lunaSign::Login function executes it will always login to slot 1 and slot 1 will always be there as long as at least 1 SafeNet appliance is operational and accessible. The Login function returns the number of slots with “tokens” present (in other words the number of accessible SafeNet appliance partitions). In normal operation in the above case the value should be 2. If it returns with less than 2, then there is an added function that can be called that will return the identity of the still live unit.

7
Java Interfaces

This chapter describes the Java interfaces to the PKCS#11 API. It contains the following topics:

SafeNet JSP Overview and Installation

The SafeNet JSP is part of an application program interface (API) that allows Java applications to make use of certain SafeNet products.

As with other APIs, some existing Java-based applications might have generic requirements and calls that can already work with SafeNet products. In other cases, it might be necessary for you or your vendor to create an application or to adapt one, using the JSP API.

You have the choice of:

- using a previously integrated third-party application, known to work with this SafeNet product
- performing your own integration with a Java-based application supplied by you or a third party, or
- developing your own application using our Java API.

Develop your own Java apps using our included Software Development Kit, which includes SafeNet Java API usage notes for developers, as well as development support by SafeNet. A standard Java development environment is required, in addition to the API provided by SafeNet.

Please refer to the current-version SafeNet HSM Customer Release Notes (CRN) for the most up-to-date list of supported platforms and APIs.

JDK Compatibility

We formally test SafeNet HSMs and our Java provider with SUN JDK for all platforms except AIX, and with IBM JDK for the AIX platform. We have not had problems with OpenJDK, although it has not been part of our formal test suite. The SafeNet JCE provider is compliant with the JCE specification, and should work with any JVM that implements the Java language specification.

Occasional problems have been encountered with respect to IBM JSSE.

GNU JDK shipped with most Linux systems has historically been incomplete and not suitable.

Installation

To use the SafeNet JavaSP service providers four main components are needed.

Java SDK 1.6.0.xx or 1.7.0.xx or 1.8.0.xx

First, acquire and install the Java SDK or RTE (available from the Java site, not included with the SafeNet software). Java must be installed before the SafeNet software, as some of the Java files must be manipulated as described in the JSP portions of the Getting Started section of this Help. Note that the JVM 1.6.x_xx or JVM 1.7.x or JVM 1.8.x is part of the Java SDK.

Java Cryptographic JCE Policy files (optional)

If you intend to generate large key sizes, you will need two cryptographic JCE Policy files v 1.6.x or v 1.7.x or 1.8.x (available from the Java web site). The Getting Started section of this Help has instructions on what to do with the two files (local_policy.jar and US_export_policy.jar).

If you see errors like "Invalid Key size", that is usually an indication that the JCE is not properly installed.

SafeNet Client CD

Follow the installation procedure for the SafeNet Client as described in the *Installation Guide*.

SafeNet JavaSP

When installing the SafeNet Client software, also choose the option to install SafeNet JSP. Instructions are provided in the platform-specific pages, including instructions for installing SafeNet JSP for each operating system (files to copy/replace, editing to perform, etc.) so that SafeNet Network HSM and SafeNet JSP can work with the JRE.

Note: Java Provider (JSP) - both GMC and GMAC are supported. "GmacAesDemo.java" provides a sample for using GMAC with Java.

Java Parameter Specification class LunaGmacParameterSpec.java defines default values recommended by the NIST specification.

Post-Installation Tasks

"Extractable" Option

The Luna provider provides an option to make newly secret keys extractable from the HSM, via the LunaSlotManager.setSecretKeysExtractable() method.

Some situations exist in which keys should be extractable but this method cannot be used; for example, when the Luna provider is performing crypto operations for a TLS server[1]. We now provide a configuration option to enable this behavior. To make secret keys extractable, add the following line to java.security:

[1] [because you cannot call this method from within your application and have it apply to the TLS server]

com.safenetinc.luna.provider.createExtractableKeys=true

This value will be read by the Luna provider on startup; to change the setting after the application has started, use the LunaSlotManager method. Using that method overrides the setting in the file for that application, but does not overwrite it permanently.

When Java, the SafeNet Client and SafeNet JSP are installed as directed, you may then perform any integration required for your own, or third-party Java application.

Using SafeNet JCE/JCA with 64-bit Libraries

If you are using SafeNet JCE/JCA with the 64-bit libraries for SafeNet Network HSM, you must include the "-d64" switch in the Java command-line.

For example: java -d64 -jar jMultitoken.jar

For most 64-bit platforms, 64-bit is supported. Some 64-bit platforms support the option of running in 32-bit mode), as a backward compatibility feature.

If you use the 64-bit installation and do not use the "- d64" command-line switch in your Java command lines, the system attempts (by default) to use the 32-bit library (which is not installed, because you installed 64-bit in this example...), and the result is an error message complaining about the kernel model.

Using ECC Keys for TLS with Java 7

For optimal Java performance when using Elliptic Curve keys to perform TLS with Java 7, where those keys reside in the HSM, you must configure the SunEC security provider (sun.security.ec.SunEC) to be **below** the LunaProvider in your java.security file.

We suggest that you **not** attempt to resolve a performance issue by having the LunaProvider as the default because that would result in the symmetric keys also being used in the HSM which is not optimal for performance.

A Security Note for Java Developers

The SafeNet JSP is a Java API that is intended to be used as an interface between customer-written or third-party Java applications and the SafeNet HSM. Managing security issues associated with the overall operational environment in which the application is running, including the user interface, is the responsibility of the application.

A common example would be input and capture of user name and password. The application, or a set of organizational procedures, is responsible for making the access control decision regarding whether the user has the necessary permissions (at the organizational level) to access the HSM's services and then must provide protection for the password as it is entered, and erasure from memory after the operation is completed. The SafeNet JSP will control access to the HSM based on the correct password being input from the application via the Login method, but security outside the HSM is your responsibility.

Non-standard ECDSA

The SafeNet provider maps the "ECDSA" signature algorithm to "NONEwithECDSA". The Java convention is to map it to "SHA1withECDSA". This is noted here in case you wish to use it in provider inter-operability testing. This mapping is noted in the Javadoc as well.

For comparison, "RSA" maps to "NONEwithRSA" while "DSA" maps to "SHA1withDSA".

SafeNet JSP Configuration

SafeNet JSP consists of a single JCA/JCE service provider, that allows a Java-based application to use SafeNet HSM products for secure cryptographic operations. Please refer to the Javadocs accompanying the toolkit, for the most current information regarding the SafeNet JSP packages and LunaProvider functionality.

Installation

You must acquire a Java JDK or JRE separately and install it before installing the SafeNet JSP. See the QuickStart that came with your software package.

In order to use the LunaProvider you must place the jar file in your classpath. We recommend placing it in your <jre>/lib/ext folder. In addition the JNI component, which may be a .dll or .so file depending on your system architecture, should be placed in your library path.

Java -- Encryption policy files for unlimited strength ciphers

Additionally, you might need to apply the unlimited strength ciphers policy. The unlimited strength ciphers policy files can be downloaded from Oracle.

The US_export_policy.jar and local_policy.jar are to be copied to JAVA_HOME/jre/lib/security (or the equivalent directory that applies to your setup).

```
[root@my-sa5client]# echo $JAVA_HOME
/usr/java/default
[root@my-sa5client]# cp -p local_policy.jar /usr/java/default/jre/lib/security/
[root@my-sa5client]# cp -p US_export_policy.jar /usr/java/default/jre/lib/security/
```

SafeNet Java Security Provider

In general, you should use the standard JCA/JCE classes and methods to work with SafeNet HSMs. The following sections provide examples of when you may wish to use the special SafeNet methods.

Class Hierarchy

All public classes in the SafeNet Java crypto provider are included in the com.safenetinc.luna package or subpackages of that package. Thus the full class names are (for example):

- com.safenetinc.luna.LunaSlotManager
- com.safenetinc.luna.provider.key.LunaKey

If your application is compliant with the JCA/JCE spec, you will generally not need to directly reference any SafeNet implementation classes. Use the interfaces defined in the java.security packages instead. The exception is if you need to perform an HSM-specific operation, such as modifying PKCS#11 attributes.

Throughout the rest of this document, the short form of the class names is used for convenience and readability. The full class names (of SafeNet or other classes) are used only where necessary to resolve ambiguity.

Special Classes/Methods

The JCA/JCE interfaces were not designed with hardware security modules (HSMs) in mind and do not include methods for managing aspects of a hardware module. SafeNet JSP provides some additional functions in addition to the standard JCA/JCE API.

The LunaSlotManager class provides custom methods that allow some HSM-specific information to be retrieved. It also provides a way to log in to the HSM if your application cannot make use of the standard KeyStore interface. For details please check the Javadoc which comes with the product.

It is not always necessary to use the LunaSlotManager class. With proper use of the JCE API provided in SafeNet JSP, your code can be completely hardware-agnostic.

The LunaKey class implements the Key interface and provides all of the methods of that class along with custom methods for manipulating key objects on SafeNet hardware.

Note: Sensitive attributes cannot be retrieved from keys stored on SafeNet hardware. Thus certain JCE-specified methods (such as PrivateKeyRSA.getPrivateExponent()) will throw an exception.

The LunaCertificateX509 class implements the X509Certificate methods along with custom methods for manipulating certificate objects on SafeNet hardware.

Examples

The SafeNet JSP comes with several sample applications that show you how to use the Luna provider. The samples include detailed comments.

To compile on windows without an IDE (administrator privileges may be required)

```
cd <SafeNet Network HSM install>/jsp/samples
javac com\safenetinc\luna\sample\*.java
```

To run

```
java com.safenetinc.luna.sample.KeyStoreLunaDemo (or any other sample class in that package)
```

Authenticating to the HSM

In order to make use of an HSM, it is necessary to activate the device through a login. Depending on the security level of the device, the login will require a plain-text password and/or a PED key.

The preferred method of logging in to the module is through the Java KeyStore interface. The store type is “Luna” and the password for the key store is the challenge for the partition specified.

KeyStore files for the Luna KeyStore must be created manually. The content of the KeyStore file differs if you wish to reference the partition by the slot number or label (preferred). Details of authenticating to the HSM via the KeyStore interface are explained in the Javadoc for LunaKeyStore and in the KeyStoreLunaDemo sample application.

Keys in a Luna KeyStore cannot have individual passwords. Only the KeyStore password is used. If your HSM requires PED keys to be presented for authentication and the partition is not already activated, loading the KeyStore will cause the PED to prompt you to present this key.

Other than the KeyStore interface your application may also make use of the LunaSlotManager class or by using a login state created outside of the application through a utility called ‘salogin’. Use of salogin is strongly discouraged unless you have a very specific need.

LunaKeyStoreMP is Deprecated

LunaKeyStoreMP is deprecated for SafeNet JSP, and may be discontinued in a future release. LunaKeyStoreMP was used in previous releases to allow logical partitioning of the key space on HSMs that have only one partition. This allowed you to create a separate MP key store for each individual client that accessed the partition. Recent SafeNet

releases, however, support multiple partitions, and dedicating a partition per client is a superior solution for management and security reasons.

> **Note:** LunaKeyStoreMP is retained for backwards compatibility reasons only. Do not use LunaKeyStoreMP when creating new applications.

Logging Out

Logging out of the HSM is performed implicitly when the application is terminated normally. Logging out of the HSM while the application is running can be done with the LunaSlotManager class. Please note that any ephemeral (non-persistent) key material present on the HSM will be destroyed when the session is logged out. Because the link to the HSM will be severed, cryptographic objects that were created by the LunaProvider will no longer be usable. Attempting to use these objects after logging out will result in undefined behaviour.

All key material which was persisted on the HSM (either through the KeyStore interface or using the proprietary Make Persistent method) will remain on the HSM after a logout and will be accessible again when the application logs back in to the HSM.

Keytool

The SafeNet JSP may be used in combination with Java's keytool utility to store and use keys on a SafeNet HSM, see "Using Java Keytool with SafeNet HSM" on page 363.

Cleaning Up

Keys that are made persistent will continue to exist on the HSM until they are explicitly destroyed, or until the HSM is reinitialized. Persistent keys that are no longer needed can be explicitly destroyed to free resources on the HSM.

Keys may be removed using the Keytool, or programmatically through the KeyStore interface or other methods available through the API.

LunaSlotManager contains methods that report the number of objects that exist on the HSM. See the Javadoc for LunaSlotManager for more information.

PKCS#11/JCA Interaction

Keys created using the SafeNet PKCS#11 API can be used with the SafeNet JSP; the inverse is also true.

Certificate Chains

The PKCS#11 standard does not provide a certificate chain representation. When a Java certificate chain is stored on a SafeNet token, the certificates of the chain appear as individual objects when viewed through the PKCS#11 API. In order for the LunaProvider to properly identify PKCS#11-created certificates as part of a chain attached to a private key, the certificates must follow the labeling scheme described below.

Java Aliases and PKCS#11 Labels

The PKCS#11 standard defines a large set of object attributes, including the object label. This label is analogous to the Object alias in a java KeyStore.

The SafeNet KeyStore key entry or a SafeNet KeyStore certificate entry will have a PKCS#11 object label exactly equal to the Java alias. Similarly, a key created through PKCS#11 will have a Java alias equal to the PKCS#11 label.

Because a java certificate chain cannot be represented as a single PKCS#11 object, the individual certificates in the chain will each appear as individual PKCS#11 objects. The labels of these PKCS#11 objects will be composed of the alias of the corresponding key entry, concatenated with "--certX", where 'X' is the index of the certificate in the java certificate chain.

For example, consider a token that has a number of objects created through the Java API. The objects consist of the following:

- A key entry with alias "signing key", consisting of a private key and a certificate chain of length 2
- A trusted certificate entry with alias "root cert"
- A secret key with alias "session key"

If all objects on the token were viewed through a PKCS#11 interface, 5 objects would be seen:

- A private key with label "signing key"
- A certificate with label "signing key--cert0"
- A certificate with label "signing key--cert1"
- A certificate with label "root cert"
- A secret key with label "session key"

Note: PKCS#11 labels (strings of ascii characters) and Java aliases (of the java.lang.String type) are usually fully compatible, but problems can arise if non-printable characters are used. To maintain compatibility between Java and PKCS#11, avoid embedding non-printable or non-ascii characters in aliases or object labels.

RSA Cipher

Previously, by default, the SafeNet JSP RSA cipher mode used raw RSA X.509 encryption, with no padding.

For improved security and compatibility, default padding for RSA cipher has been changed from NoPadding to PKCS1v1_5.

The JCPROV PKCS#11 Java Wrapper

This section describes how to install and use the JCPROV Java wrapper for the PKCS#11 API. It contains the following topics:

JCPROV Overview

JCPROV is a Java wrapper for the PKCS#11 API. JCPROV is designed to be as similar to the PKCS#11 API as the Java language allows, allowing developers who are familiar with the PKCS#11 API to rapidly develop Java-based

programs that exercise the PKCS#11 API.

> **Note: JCPROV** - at time of writing (August 2015) GMAC is supported, but GCM is not. Use CK_AES_CMAC_PARAMS.java to define the GMAC operation. Implementation is the same as for PKCS#11.

JDK compatibility

The JCPROV Java API is compatible with JDK 1.5.0 or higher.

The JCPROV library

The JCPROV library is implemented in **jcprov.jar**, under the namespace **safenet.jcprov**. It is accompanied by a shared library that provides the native methods used to access the appropriate PKCS#11 library. The name of the shared library is platform dependent, as follows:

Operating system	Shared library
Windows (32 and 64 bit)	jcprov.dll
Linux	libjcprov.so
Solaris	libjcprov.so
HP-UX	libjcprov.sl
AIX	libjcprov.so

Installing JCPROV

Use the SafeNet Client Installer to install the JCPROV software (runtime and SDK packages). The software is installed in the location specified in the following table:

Operating system	Installation location
Windows	C:\Program Files\safenet\lunaclient\jcprov
Linux	/usr/safenet/lunaclient/jcprov
Solaris	/opt/safenet/lunaclient/jcprov
HP-UX	/opt/safenet/lunaclient/jcprov
AIX	/usr/safenet/lunaclient/jcprov

The installation includes a **samples** subdirectory ( ) and a **javadocs** subdirectory ( ).

Changing the Java JNI libraries (AIX only)

The Java VM on AIX does not support mixed mode JNI libraries. Mixed mode libraries are shared libraries that provide both 32-bit and 64-bit interfaces. It is therefore essential that you select the correct JNI library to use with your Java VM.

CAUTION: When JCPROV is installed, links are automatically created to use the 32-bit versions of the JNI libraries. You need to update the links if you are using a 64-bit operating system.

To configure the JNI library for use with a 32-bit Java VM

1. Ensure that the **/usr/safenet/lunaclient/jcprov/lib/libjcprov.a** symbolic link points to a 32-bit version of the library (**libjcprov_32.a**), for example **/usr/safenet/lunaclient/jcprov/lib/libjcprov_32.a**.
2. Ensure that the **/usr/safenet/lunaclient/jcprov/lib/libjcryptoki.a** symbolic link points to a 32-bit version of the library(**libjcryptoki_32.a**), for example **/usr/safenet/lunaclient/jcprov/lib/libjcryptoki_32.a**.

To configure the JNI library for use with a 64-bit Java VM

1. Ensure that the **/usr/safenet/lunaclient/jcprov/lib/libjcprov.a** symbolic link points to a 64-bit version of the library (**libjcprov_64.a**), for example **/usr/safenet/lunaclient/jcprov/lib/libjcprov_64.a**.
2. Ensure that the **/usr/safenet/lunaclient/jcprov/lib/libjcryptoki.a** symbolic link points to a 64-bit version of the library (**libjcryptoki_64.a**), for example **/usr/safenet/lunaclient/jcprov/lib/libjcryptoki_64.a**.

JCPROV Sample Programs

Several sample programs are included to help you become familiar with JCPROV. The binaries for the sample programs are included in the **jcprovsamples.jar** file. You must compile the binaries before you can use the sources provided.

Compiling and running the JCPROV sample programs

CAUTION: You require JDK 1.5.0 or newer to compile the JCPROV sample programs.

It is recommended that you compile the samples in their installed locations, so that the path leading to the samples directory in the installation location will allow them to be executed as documented below.

Prerequisites

For best results, perform the following actions before attempting to compile the sample programs:

- add **jcprov.jar** to your **CLASSPATH** environment variable
- add a path to the **CLASSPATH** environment variable that allows JCPROV to use the **safenet.jcprov.sample** namespace. This is required since all of the applications are registered under this namespace.

To compile the JCPROV sample programs on UNIX/Linux

1. Create a temporary compile directory.
   mkdir –p safenet/jcprov/samples
2. Copy the sample program and makefile into the temporary compile directory.
   cp <jcprov_installation_directory>**/jcprov/samples/* safenet/jcprov/samples**
3. Set the **CLASSPATH** environment variable to point to **jcprov.jar** and the root path for the sample programs.
   export CLASSPATH=<jcprov_installation_directory>**/jcprov.jar:`pwd`**
4. Change directory to the sample programs path.
   cd safenet/jcprov/samples

5. Use the **javac** program to compile the examples.
   javac GetInfo.java
6. Use the **java** program to run the samples.
   java safenet.jcprov.samples.GetInfo -info

To compile the JCPROV sample programs on Windows

1. Set the **CLASSPATH** environment variable to point to **jcprov.jar** and the root path for the sample programs:
   C:\> **set "CLASSPATH= C:\Program Files\safenet\lunaclient\jcprov\jcprov.jar; C:\program files\safenet\jcprov\samples"**
2. Use the **javac** program to compile the examples:
   C:\Program Files\safenet\lunaclient\jcprov\samples> **javac GetInfo.java**
3. Use the **java** program to run the samples:
   C:\Program Files\safenet\lunaclient\jcprov\samples> **java safenet.jcprov.samples.GetInfo -info**

JCPROV Sample Classes

JCPROV provides the following sample classes. The sample classes are located in the <jcprov_installation_ directory>/**samples** directory.

DeleteKey

Demonstrates the deletion of keys.

Usage

java safenet.jcprov.sample.DeleteKey -keyType <keytype> **-keyName** <keyname> [**-slot** <slotId>] [**-password** <password>]

Parameters

Parameter	Description
-keytype	Specifies the type of key you want to delete. Enter this parameter followed by one of the following supported key types: • **des** - single DES key • **des2** - double-length, triple-DES key • **des3** - triple-length, triple-DES key • **rsa** - RSA key pair
-keyName	Specifies the name (label) of the key you want to delete. Enter this parameter followed by the name (label) of the key you want to delete.
-slot	Specifies the slot for the HSM or partition that contains the key you want to delete. Optionally enter this parameter followed by the slot identifier for the HSM or partition that contains the key you want to delete. If this parameter is not specified, the default slot is used. **Default:** 1
-password	Specifies the password for the slot. Optionally enter this parameter followed by the slot password to delete a private key.

EncDec

Demonstrates encryption and decryption operations by encrypting and decrypting a string.

Usage

java safenet.jcprov.sample.EncDec -keyType <keytype> **-keyName** <keyname> [**-slot** <slotId>] [**-password** <password>]

Parameters

Parameter	Description
-keytype	Specifies the type of key you want to use to perform the encryption/decryption operation. Enter this parameter followed by one of the following supported key types: • **des** - single DES key • **des2** - double-length, triple-DES key • **des3** - triple-length, triple-DES key • **rsa** - RSA key pair
-keyName	Specifies the name (label) of the key you want to use to perform the encryption/decryption operation. Enter this parameter followed by the name (label) of the key you want to use to perform the encryption/decryption operation.
-slot	Specifies the slot for the HSM or partition that contains the key you want to use to perform the encryption/decryption operation. Optionally enter this parameter followed by the slot identifier for the HSM or partition that contains the key you want to use to perform the encryption/decryption operation. If this parameter is not specified, the default slot is used. **Default:** 1
-password	Specifies the password for the slot. Optionally enter this parameter followed by the slot password to encrypt/decrypt a private key.

GenerateKey

Demonstrates the generation of keys.

Usage

java safenet.jcprov.sample.GenerateKey -keyType <keytype> **-keyName** <keyname> [**-slot** <slotId>] [**-password** <password>]

Parameters

Parameter	Description
-keytype	Specifies the type of key you want to generate. Enter this parameter followed by one of the following supported key types: • **des** - single DES key • **des2** - double-length, triple-DES key • **des3** - triple-length, triple-DES key • **rsa** - RSA key pair

Parameter	Description
-keyName	Specifies the name (label) of the key you want to generate. Enter this parameter followed by the name (label) of the key you want to generate.
-slot	Specifies the slot for the HSM or partition where you want to generate the key. Optionally enter this parameter followed by the slot identifier for the HSM or partition where you want to generate the key. If this parameter is not specified, the default slot is used. **Default:** 1
-password	Specifies the password for the slot. Optionally enter this parameter followed by the slot password to generate a private key.

GetInfo

Demonstrates the retrieval of slot and token information.

Usage

java safenet.jcprov.sample.GetInfo {**-info** | **-slot** [<slotId>] | **-token** [<slotId>]}

Parameters

Parameter	Description
-info	Retrieve general information.
-slot	Retrieve slot information for the specified slot. Enter this parameter followed by the slot identifier for the slot you want to retrieve information from. If <slotId> is not specified, information is retrieved for all available slots.
-token	Retrieve token information for the HSM or partition in the specified slot. Enter this parameter followed by the slot identifier for the HSM or partition you want to retrieve information from. If <slotId> is not specified, information is retrieved for all available slots.

Threading

This sample program demonstrates different ways to handle multi-threading.

This program initializes the Cryptoki library according to the specified locking model. Then a shared handle to the specified key is created. The specified number of threads is started, where each thread opens a session and then enters a loop which does a triple DES encryption operation using the shared key handle.

It is assumed that the key exists in slot 1, and is a Public Token object.

Usage

java ...Threading -numThreads <numthreads> **-keyName** <keyname> **-locking** { **none** | **os** | **functions** } [**-v**]

Parameters

Parameter	Description
-numthreads	Specifies the number of threads you want to start. Enter this parameter followed by an integer that specifies the number of threads you want to start.
-keyName	Specifies the triple-DES key to use for the encryption operation. Enter this parameter followed by the name (label) of the key to use for the encryption operation.
-locking	Specifies the locking model used when initializing the Cryptoki library. Enter this parameter followed by one of the following locking models: • **none** - do not use locking when initializing the Cryptoki library. If you choose this option, some threads should report failures. • **os** - use the native operating system mechanisms to perform locking. • **functions** - use Java functions to perform locking
-v	Specifies the password for the slot. Optionally enter this parameter followed by the slot password to generate a private key.

JCPROV API Documentation

The JCPROV API is documented in a series of javadocs. The documentation is located in the <jcprov_installation_ directory>**/javadocs** directory.

Java or JSP Errors

In the process of using our JSP (Java Service Provider) or programming for Java clients, you might encounter a variety of errors generated by various levels of the system. In rare cases those might be actual problems with the system, but in the vast majority of cases the errors are the system (or the Client-side libraries) telling you that you (or your application) have done something "wrong". In other words, the error messages are guidance to ensure that your actions and your programs are giving the system what it needs (in the right order and format) to complete the tasks that you ask of it.

Keep in mind that there are several levels involved. The SafeNet appliance and its HSM keycard have both software and firmware built in. Among other things, the system software handles the system side of communication between you (either as administrator or as Client) and the HSM on the appliance. In general, a client-side program (or programmer) would not encounter error messages directly from the system. If an error condition arises on the system, the most likely visibility would be error messages in the system logs - viewed by the appliance administrator - or else client-side messages based upon the interaction of the client-side software (ours and yours) with the appliance.

On the client side, the JSP and any Java programs that you use would be overlaid on, and using, the SafeNet library, which is an extended version of PKCS#11, customized to make use of our HSM (the standard itself and the cryptoki library are oriented toward in-software implementation of cryptographic functions, with some generic support of generic HSM functions, leaving room for each HSM supplier to support their own special functions by extending the standard). PKCS#11 is an RSA Laboratories cryptographic standard, and our libraries are a C-language implementation of that standard. You can view all that is known about PKCS #11 error conditions and messages at the RSA website.

See "Library Codes" on page 1 for a summary of error codes and their meanings, which includes the SafeNet extensions to the PKCS#11 standard that are specific to our HSM. Note that "error codes" do not usually indicate a problem with the appliance or HSM - they indicate an exception condition has been encountered, possibly because you

(or your application) stopped/canceled a requested action before it could complete, provided incorrect or incomplete or wrongly-formatted input data, and so on, or possibly because a network connection has been disrupted, power has failed, or any of a variety of situations has been detected.

The JSP and your Java programming are overlaid on top of the PKCS#11 and SafeNet libraries. An error reported by a Java application might refer to a problem at the Java or JSP level, or the error might have been passed through from a lower level.

If you receive a cryptic error that looks something like:

```
Exception in thread "main"
com.safenetinc.crypto.LunaCryptokiException: function 'C_Initialize' returns 0x30
```

then this error has been passed through from a lower layer and is not a Java or JSP error. You should look in the Error Codes page (link above) or in the PKCS#11 standard for the meaning of any error in a similar format.

In general, we wrap cryptoki exception codes. Most exceptions thrown by the JSP are in accordance with the specification. Check the Javadoc for the API call that threw the exception.

- LunaException is used to report a LunaProvider-specific exception.
- LunaCryptokiException reports errors returned by the HSM. Those might be wrapped in other Exceptions

Re-Establishing a Connection Between Your Java Application and SafeNet Network HSM

The following snippet of java code re-establishes a connection between a Java Application and SafeNet Network HSM in the event of a disconnect (for example, firewall rules, network issues).

Note: Stop all existing crypto operations before performing the reconnect.

```
public void reconnectHsmServer() {
     LunaSlotManager lsm = LunaSlotManager.getInstance();
     lsm.reinitialize();
     lsm.login("<HSM partition password>");
}
```

Note: The reinitialize() call is a disruptive call. It unloads and reloads the dll, in order to perform a cleanup and refresh. When reinitialize() is called, there are no safe API methods that may be called and any calls in progress will result in undefined behaviour, leading, most-likely, to a JVM crash. Before calling reinitialize(), ensure that all threads making use of the API are halted or stopped, and that no other calls are made until reinitialize() has completed.

Recovering From the Loss of All HA Members

The reinitialize method of the **LunaSlotManager** class takes the role of the PKCS#11 functions **C_Finalize** and **C_Initialize**. It is intended to be used when a complete loss of communication happens with all the members of your High Availability (HA) group.

This section describes the situations in which you should use this method, the effect this method has on a running application, and how to use this method safely. It is assumed that the auto-insert (auto-recovery) features of the HA group are enabled.

You should read this section if you are developing an application that uses the LunaProvider in an environment that leverages an HA group of SafeNet Network HSM appliances, so that you can safely recover an entire HA group.

When to Use the reintialize Method

When using the high-availability (HA) features of SafeNet Network HSM, the auto-insert (auto-recovery) feature will resolve situations where connectivity is lost to a subset of members for a brief time. However, if you lose connection to all members then the connection cannot be automatically recovered. Finalizing the library and initializing it again is the only way to recover other than restarting the application.

Why the Method Must Be Used

In an HA group, we rely on having at least one member present in order to maintain state. If all of the members have been lost, then we cannot make any determination of which member has a known good state. Also, when a connection to a member is lost, the authenticated state is lost. When an individual member returns, we can use the authenticated state from another member to authenticate to the one that has returned. When all members are lost, then the authenticated state is lost on all members.

What Happens on the HSM

The Network Trust Link Service (NTLS) on the HSM appliance is responsible for cleaning up any cryptographic resources, such as session objects, and cryptographic operation contexts when a connection to the client is lost. This happens when the socket closes.

Effect on Running Applications

All resources created within the LunaProvider must be treated as junk after the library is finalized. Sessions will no longer be valid, session objects will point to non-existent objects or worse to a wrong object, and **Signature/Cipher/Mac/etc** objects will have invalid data.

Even **LunaKey** objects, which represent persistent objects, may contain invalid data. When the virtual slot is constructed in the library, the virtual object table is built from the objects present on each individual member. There is no guarantee that objects will have the same handle from one initialization to the next. This is true from the moment the connection to the group is severed. All these resources must be released before calling the reinitialize method. Beyond causing undesirable behavior when used, if these objects are garbage collected after cryptographic operations resume, they can result in the deletion of new objects or sessions.

Using the Method Safely

The first indication that all communications may have been lost with the group is a **LunaException** reporting an error code of **0x30** (Device Error). Other possible error codes that can indicate this status are **0xE0** (Token not present) and **0xB3** (Session Handle invalid). The **LunaException** class does not provide the error code as a discrete value and you will have to parse the message string to determine this value.

At this point, you should validate that the group has been lost. The **com.safenetinc.luna.LunaHAStatus** object is best suited for this. Your application should know the slot number of the HA slot that you are using because it may not be able to query this information from the label when the slot is missing.

Example

```
LunaHAStatus status = new LunaHAStatus(haSlotNumber);
```

You can query the object for detailed information or just use the **isOK()** method to determine if the group has been lost. The **isOK()** method will return true if all members are still present. If all members are gone, an exception will be thrown.

If no application is thrown, the application should be able to proceed operating, and any individual members of the HA group that have been lost will be recovered by the library. Further details on failed members can be queried through the LunaHAStatus object.

In many highly threaded applications, such as web applications, it is desirable to have a singleton, which is responsible for keeping track of the health of the HSM connection. This can be done by having worker threads report information to this singleton, by having a specific health check thread, or through a combination of the two.

Once the error state is discovered, all worker threads should be stopped or allowed to return an error. It may take up to 40 seconds from the time the group was lost for all threads to discover that there is an error. It can take 20 seconds for any given command to time out as a result of network failure. Once this happens, new commands will not be sent to that HSM, but a command may have just been sent and that command will have its own 20-second timeout. As mentioned above, in the section on application effects, all of the objects created or managed by the LunaProvider must be considered at this point to contain junk data. Operating after recovery with this junk data can cause undesired effects. This means all keys, signature, cipher, Mac, KeyGenerator, KeyPairGenerator, X509Certificate, and similar objects must be released to the garbage collector. Instances of most non-SPI (LunaAPI, LunaSlotManager, LunaTokenManager, etc.) objects do not pose a problem, but any instances of LunaSession held in the application during the course of the reinitialize can cause problems if they are returned to the session pool after the reinitialization takes place.

Cryptographic processing in the application should be halted until connection with the HSMs is back to a known good state. It may be appropriate to hold operations in a queue for processing later or to return an Out of Service message.

Once the objects have been released and no further processing will occur, the application should attempt recovery of the connection. This is done through the **com.safenetinc.luna.LunaSlotManager.reinitialize** method. This method will first clear session objects held within the provider before finalizing the library. After the library is finalized, it will initialize it again by invoking the **C_Initialize** method. This method will establish a connection with all the HSMs if possible. The same **isOK()** method of **LunaHAStatus** can be used to determine if the group has been recovered successfully.

It is also important to only have a single thread call the **reinitialize** method. When multiple threads try to unload or load the library at the same time, errors can occur.

Elliptic Curve Problem in SUN JDK 1.6 and earlier

If you are using a version of SUN JDK earlier than 6 (JDK 1.6), avoid using these specific elliptic curves:

- secp160k1
- secp192k1
- secp224k1
- secp256k1
- sect233k1
- sect239k1
- sect283k1
- sect409k1

- sect571k1
- X9.62 c2pnb208w1

Due to a Java bug, those curves can result in errors if they (validly) have coefficients of zero.

If you must use those curves, then update to SUN Java 1.6, which fixes the problem.

The bug is acknowledged on the SUN website, here:

http://bugs.sun.com/bugdatabase/view_bug.do;jsessionid=1b1e7de9a9bde55460e50e7fbed5?bug_id=6542846

Using Java Keytool with SafeNet HSM

This page describes how to use the Java KeyTool application with the LunaProvider.

Limitations

The following limitations apply:

- You cannot use the importkeystore command to migrate keys from a Luna KeyStore to another KeyStore.
- Private keys cannot be extracted from the KeyStore unless you have the Key Export model of the HSM.
- By default secret keys created with the LunaProvider are non-extractable.

The example below uses a KeyStore file containing only the line “slot:1”. This tells the Luna KeyStore to use the token in slot 1.

For information on creating keys through Key Generator or Key Factory classes please see the LunaProvider Javadoc or the JCA/JCE API documentation.

Keys (with self signed certificates) can be generated using the keytool by specifying a valid Luna KeyStore file and specifying the KeyStore type as “Luna”. The password presented to authenticate to the KeyStore is the challenge password of the partition.

Example

```
keytool -genkeypair -alias myKey -keyalg RSA -sigalg SHA256withRSA -keystore keystore.luna -
storetype Luna
Enter keystore password:
What is your first and last name?
[Unknown]:  test
What is the name of your organizational unit?
[Unknown]:  codesigning
What is the name of your organization?
[Unknown]:  SafeNet Inc
What is the name of your City or Locality?
[Unknown]:  Ottawa
What is the name of your State or Province?
[Unknown]:  ON
What is the two-letter country code for this unit?
[Unknown]:  CA
Is CN=test, OU=codesigning, O=SafeNet Inc, L=Ottawa, ST=ON, C=CA correct?
[no]:  yes
Enter key password for <myKey>
(RETURN if same as keystore password):
```

Keytool Usage and Examples

The LunaProvider is unable to determine which PKCS#11 slot to use without providing a keystore file. This file can be manually created to specify the desired slot by either the slot number or partition label. The naming of the files is not important - only the contents.

The keytool examples below refer to a keystore file named bylabel.keystore. Its content is just one line:

```
tokenlabel:a-partition-name
```

where a-partition-name is the name of the partition you want the Java client to use.

Here is the (one line) content of a keystore file that specifies the partition by slot number:


```
slot:1
```

where 1 is the slot number of the partition you want the Java client to use.

To test that the Java configuration is correct, execute:

```
my-sa6client:~/luna-keystores$ keytool -list -v -storetype Luna -keystore
bylabel.keystore
```

The system requests the password of the partition and shows its contents.

Here is a sample command to create an RSA 2048 bit key with SHA256withRSA self-signed certificate. This example uses java 6, other versions might be slightly different.

keytool -genkeypair -alias keyLabel -keyalg RSA -keysize 2048 -sigalg SHA256withRSA -storetype Luna -keystore bylabel.keystore -validity 365

```
Enter keystore password:
What is your first and last name?
  [Unknown]:  mike
What is the name of your organizational unit?
  [Unknown]:  appseng
What is the name of your organization?
  [Unknown]:  safenet
What is the name of your City or Locality?
  [Unknown]:  ottawa
What is the name of your State or Province?
  [Unknown]:  on
What is the two-letter country code for this unit?
  [Unknown]:  ca
Is CN=mike, OU=appseng, O=safenet, L=ottawa, ST=on, C=ca correct?
  [no]:  yes
Enter key password for <keyLabel>
  (RETURN if same as keystore password):
```

With the Luna provider there is no concept of a key password and anything entered is ignored.

The following is a more elaborate sequence of keytool usage where the final goal is to have the private key generated in the HSM through keytool “linked” to its certificate.

Import CA certificate

It is mandatory to import the CA certificate – keytool verifies the chain before importing a client certificate:

```
my-sa5client:~/luna-keystores$ keytool -importcert -storetype Luna -keystore bylabel.keystore -
alias root-ugoca -file Ugo_CA.crt
```

It is not required to import this certificate in the Java default cacerts keystore.

Generate private key

Generate the private key. It is NOT important that the sigalg specified matches the one used by the CA. You can also have OU, O, L, ST, and C different from the ones in the CA certificate.

```
my-sa6client:~/luna-keystores$ keytool -genkeypair -alias java-client2-key -keyalg RSA -keysize
2048 -sigalg SHA256withRSA -storetype Luna -keystore bylabel.keystore
Enter keystore password:
What is your first and last name?
[Unknown]:  java-client2
What is the name of your organizational unit?
[Unknown]:  SE
```



```
What is the name of your organization?
[Unknown]:  SFNT
What is the name of your City or Locality?
[Unknown]:  bgy
What is the name of your State or Province?
[Unknown]:  bg
What is the two-letter country code for this unit?
[Unknown]:  IT
Is CN=java-client2, OU=SE, O=SFNT, L=bgy, ST=bg, C=IT correct?
[no]:  yes
Enter key password for <java-client2-key>
(RETURN if same as keystore password):
```

Verify that the private key is in the partition:

```
my-sa6client:~/luna-keystores$ keytool -list -v -storetype Luna -keystore bylabel.keystore
Enter keystore password:
Keystore type: LUNA
Keystore provider: LunaProvider
Your keystore contains 2 entries
Alias name: root-ugoca
Creation date: Oct 4, 2012
Entry type: trustedCertEntry
Owner: EMAILADDRESS=ugo@computer.org, CN=Ugo CA, OU=SE, O=SFNT, L=bgy, ST=bg, C=IT
Issuer: EMAILADDRESS=ugo@computer.org, CN=Ugo CA, OU=SE, O=SFNT, L=bgy, ST=bg, C=IT
Serial number: 1
Valid from: Thu Oct 04 09:02:00 CEST 2012 until: Tue Oct 04 09:02:00 CEST 2022
Certificate fingerprints:
      MD5:  A2:15:4F:94:70:2B:D2:F7:C0:96:B1:47:F2:1D:03:E9
      SHA1: B3:4A:68:0A:8D:12:39:86:11:CE:EF:22:1B:D1:DE:8D:E9:19:2B:F4
      Signature algorithm name: SHA256withRSA
      Version: 3
*******************************************
*******************************************
Alias name: java-client2-key
Creation date: Oct 4, 2012
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=java-client2, OU=SE, O=SFNT, L=bgy, ST=bg, C=IT
Issuer: CN=java-client2, OU=SE, O=SFNT, L=bgy, ST=bg, C=IT
Serial number: 506d42dd
Valid from: Thu Oct 04 10:03:41 CEST 2012 until: Wed Jan 02 09:03:41 CET 2013
Certificate fingerprints:
      MD5:  7A:37:72:6B:8A:05:B6:49:91:70:0F:C4:04:1F:69:D9
      SHA1: 05:CD:9F:A5:37:0B:A6:A3:65:24:56:40:5E:29:2D:95:2D:53:8F:5F
      Signature algorithm name: SHA256withRSA
      Version: 3
```

Create the CSR

Create the CSR to be submitted to the CA.

```
my-sa5client:~/luna-keystores$ keytool -certreq -alias java-client2-key -file client2-ugoca.csr
-storetype Luna -keystore bylabel.keystore
Enter keystore password:
```

Now have the CSR signed by the CA. Have the issued certificate exported to include the certificate chain. Without the chain, keytool fails with the error:


```
java.lang.Exception: Failed to establish chain from reply
```

If you do not have the chain, you can use the steps in the section below to build the chain yourself.

To translate a PKCS#7 exported certificate from DER format to PEM format use the following:

```
my-sa5client $ openssl pkcs7 -inform der -in Luna_Key.p7b -outform pem -out Luna_Key-pem.p7b
```

Microsoft CA exports certificates with chain only in PKCS#7 PEM encoded format.

Import client certificate

Now import the client certificate:

```
user@myserver:~/luna-keystores$ keytool -importcert -storetype Luna -keystore bylabel.keystore -
alias java-client2-key -file java-client2.crt
Enter keystore password:
Certificate reply was installed in keystore
```

Ensure that it is linked to the private key generated previously – the chain length is not 1 (“Certificate chain length: 2)

```
user@myserver:~/luna-keystores$ keytool -list -v -storetype Luna -keystore bylabel.keystore
Enter keystore password:
Keystore type: LUNA
Keystore provider: LunaProvider
Your keystore contains 2 entries
Alias name: root-ugoca
Creation date: Oct 4, 2012
Entry type: trustedCertEntry
Owner: EMAILADDRESS=ugo@computer.org, CN=Ugo CA, OU=SE, O=SFNT, L=bgy, ST=bg, C=IT
Issuer: EMAILADDRESS=ugo@computer.org, CN=Ugo CA, OU=SE, O=SFNT, L=bgy, ST=bg, C=IT
Serial number: 1
Valid from: Thu Oct 04 09:02:00 CEST 2012 until: Tue Oct 04 09:02:00 CEST 2022
Certificate fingerprints:
      MD5:  A2:15:4F:94:70:2B:D2:F7:C0:96:B1:47:F2:1D:03:E9
      SHA1: B3:4A:68:0A:8D:12:39:86:11:CE:EF:22:1B:D1:DE:8D:E9:19:2B:F4
      Signature algorithm name: SHA256withRSA
      Version: 3
*******************************************
*******************************************
Alias name: java-client2-key
Creation date: Oct 4, 2012
Entry type: PrivateKeyEntry
Certificate chain length: 2
Certificate[1]:
Owner: CN=java-client2, OU=SE, O=SFNT, L=bgy, ST=bg, C=IT
Issuer: EMAILADDRESS=ugo@computer.org, CN=Ugo CA, OU=SE, O=SFNT, L=bgy, ST=bg, C=IT
Serial number: 5
Valid from: Thu Oct 04 10:07:00 CEST 2012 until: Fri Oct 04 10:07:00 CEST 2013
Certificate fingerprints:
       MD5:  4B:F0:9E:BC:EB:6A:88:2B:87:3A:76:35:7C:DE:4B:B4
       SHA1: F1:0C:BC:E3:A1:97:E4:8B:24:2D:44:43:7A:EA:71:52:B3:C3:20:D7
       Signature algorithm name: SHA256withRSA
       Version: 3
Certificate[2]:
Owner: EMAILADDRESS=ugo@computer.org, CN=Ugo CA, OU=SE, O=SFNT, L=bgy, ST=bg, C=IT
Issuer: EMAILADDRESS=ugo@computer.org, CN=Ugo CA, OU=SE, O=SFNT, L=bgy, ST=bg, C=IT
Serial number: 1
```



```
Valid from: Thu Oct 04 09:02:00 CEST 2012 until: Tue Oct 04 09:02:00 CEST 2022
Certificate fingerprints:
       MD5:  A2:15:4F:94:70:2B:D2:F7:C0:96:B1:47:F2:1D:03:E9
       SHA1: B3:4A:68:0A:8D:12:39:86:11:CE:EF:22:1B:D1:DE:8D:E9:19:2B:F4
       Signature algorithm name: SHA256withRSA
       Version: 3
```

How to build a certificate with chain ...

When you receive the client certificate without the chain, it is possible to build a PKCS#7 certificate that includes the chain (and then feed it to keytool -importcert). In short, the "single" certificates without the chain can be "stacked" together by manually editing a PEM cert file; this PEM cert file can then be translated into a PKCS#7 cert. How? Like this:

1. Prerequisites. Have all the certs in .crt format. The cert in this format is represented as an ASCII file starting with the line
   `-----BEGIN CERTIFICATE-----`
   and ending with
   `-----END CERTIFICATE-----`
   For example, if the client cert is issued by a subCA and the subCA is signed by a root CA, you will have 3 cert files – the client cert, the subCA cert, and the root CA cert. If the certs are not in .crt format, openssl can be used to transform the format that you have into .crt format. See notes below.
2. Open a new text file, calling it, for example, cert-with-chain.crt. Insert into this file the content of the certificates in the chains. For the above example, you must insert FIRST the client cert, THEN the subCA cert, THEN the root CA cert. The content of the file would then resemble the following:
   ```
   -----BEGIN CERTIFICATE-----
                <-- client cert goes here
   -----END CERTIFICATE-----
   -----BEGIN CERTIFICATE-----
                <-- subCA cert goes here
   -----END CERTIFICATE-----
   -----BEGIN CERTIFICATE-----
                <-- root CA cert goes here
   -----END CERTIFICATE-----
   ```
3. Use the following openssl command to convert the new certificate with chain, that you just created above, to a PKCS#7 certificate with chain:
   ```
   my-sa6 $ openssl crl2pkcs7 -nocrl -certfile HSM_Luna-manual-chain.crt -out
   HSM_Luna-manual-chain.p7b -certfile root_CA.crt
   ```

Keytool is then able to import this .p7b certificate into the Luna keystore and correctly validate the chain.

Additional minor notes

1. Command to add a CA to the default CA cert store "cacerts":
   ```
   root@myserver:~# keytool -importcert -trustcacerts -alias root-ugoca -file
   /home/ugo/luna-keystores/Ugo_CA.crt -keystore /etc/java-6-sun-
   /security/cacerts
   ```
2. Use the following openssl command to convert a PKCS#7 certificate DER-encoded into a PKCS#7 PEM-encoded certificate:


```
user@myserver:~/tmp/$ openssl pkcs7 -inform der -in java-client2.p7b -out
java-client2-pem.p7b
```

3. Use the following openssl command to convert a PKCS#7 DER-encoded certificate into a .crt PEM certificate :
   ```
   user@myserver:~/tmp/$ openssl pkcs7 -print_certs -inform der -in Ugo_CA.p7b
   -out Ugo_CA-p7-2-crt.crt
   ```
4. Use the following openssl command to convert a PEM certificate with chain to a PKCS#7 with chain:
   ```
   user@myserver:~/tmp/$ openssl crl2pkcs7 -nocrl -certfile HSM_Luna-manual-
   chain.crt -out HSM_Luna-manual-chain.p7b -certfile Ugo_CA.crt
   ```


JSP Dynamic Registration Sample

You may prefer to dynamically register the SafeNet provider in order to avoid possible negative impacts on other applications running on the same machine. Using dynamic registration also allows you to keep installation as straightforward as possible for your customers.

This sample code shows an example of dynamic registration with SafeNet's SafeNet provider. The SafeNet provider is registered in position 2, ensuring that the "SUN" provider is still the default. If you want the SafeNet provider to be used when no provider is explicitly specified, it should be registered at position 1.

Sample Code

```
try {
  com.safenetinc.luna.LunaSlotManager.getInstance().login("<HSM Partition Password>");
  java.security.Provider provider = new com.safenetinc.luna.provider.LunaProvider();
  // removing the provider is only necessary if it is already registered
  // and you want to change its position
  java.security.Security.removeProvider(provider.getName());
  java.security.Security.insertProviderAt(provider, 2);
  com.safenetinc.luna.LunaSlotManager.getInstance().logout();
} catch (Exception e) {
  System.out.println("Exception caught during loading of the providers: "
      + ex.getMessage());
}
```


8
Microsoft Interfaces

This chapter describes the Microsoft interfaces to the PKCS#11 API. It contains the following topics:

The SafeNet CSP Registration Tool and Utilities

This section describes how to use the SafeNet CSP registration tool and related utilities to configure the SafeNet HSM client to use a SafeNet HSM with Microsoft Certificate Services. You must be the Administrator or a member of the Administrators group to run the SafeNet CSP tools.

The SafeNet CSP can be used by any application that acquires the context of the SafeNet CSP. All users who login and use the applications that acquired the context have access to the SafeNet CSP. After you register the SafeNet HSM partitions with SafeNet CSP, your CSP and KSP code should work in the same manner whether our HSM (crypto provider) is selected, or the default provider is used.

Note: The SafeNet CSP is an optional installation. It is installed by default in <luna_client_install_dir>/**CSP**. If the CSP is not installed, re-run the installer.

The Keymap Utility

Use the **keymap** utility if you have previously been using another provider (with its keys in the SafeNet HSM) and wish to migrate to MS CSP keeping your established keys. The keymap utility simply creates on the SafeNet HSM the data object that MS CSP expects, which in turn makes your existing keys available to MS CSP. See <luna_client_install_dir>/**CSP/keymap.exe**.

The ms2Luna Utility

Use the **ms2Luna** utility if you already have MS CSP in use with software key storage and you now wish to continue with your keys held on the SafeNet HSM. See <luna_client_install_dir>/**CSP/ms2luna.exe**.

The CSP Registration Tool

You can use the CSP registration tool (<luna_client_install_dir>/**CSP**/**register.exe**) to perform the following functions:

- register HSM partitions for use with the SafeNet CSP. The password for each HSM Partition is secured such that only the user for which the password was secured is able to un-secure it. See "Registering Partitions" on the next page

- register which non-RSA cryptographic algorithms you want performed in software only. See "Registering the Cryptographic Algorithms to be Performed in Software" on page 373
- enable key counting in KSP/CSP. See "Enabling Key Counting" on page 374.

Command Syntax

register.exe [**/partition** | **/algorithms** | **/library** | **/usagelimit**] [**/highavailability**] [**/strongprotect**] [**/cryptouser**] [**/?**]

Parameter	Shortcut	Description
/partition	**/p**	Register a partition and it's encrypted challenge. You are prompted through the required steps to select and register a SafeNet HSM partition. This is the default option. If you type **register** with no additional parameters, then **/partition** is assumed. For example, if you type **register /highavail** or **register /strongprotect**, then **/partition** is invoked and the additional option that you selected (i.e., **/highavail** or **/strongprotect**) is run along with it . That is, typing **register /highavail** is the same as typing **register /partition /highavail**.
/highavail	**/h**	Register only high availability (HA) partitions.
/strongprotect	**/s**	Strongly protect the challenge for registered partition
/algorithms	**/a**	Register the desired software ONLY algorithms
/library	**/l**	Register CSP library and signature in the registry
/usagelimit	**/u**	Register CSP RSA key maximum usage limit
/cryptouser	**/c**	Use CSP as Crypto User

Registering Partitions

The syntax used to register partitions depends on whether the partitions use high availability (HA) or not, as detailed in the following procedures.

To register a standard HSM partition

1. Enter the following command and respond to the prompts:
2. C:\Program Files\SafeNetLunaClient\CSP> **register**

   For example:

```
************************************************************
SafeNet Luna CSP, Partition Registration
Protect the HSM's challenge for the selected partitions.
NOTE:
This is a WEAK protection of the challenge!!
After you have configured all applications that will use
the Luna CSP, and run them once, you MUST run:
register /partition /strongprotect *
to strongly protect the registered challenges!!
************************************************************
This procedure is a destructive procedure and will completely replace
any previous settings!!
Do you wish to continue?: [y/n]
```



```
Do you want to register the partition named 'nes'? [y/n]:
Please enter the SafeNet Network HSM challenge for the partition 'nes' :
Success registering the ENCRYPTED challenge for partition 'nes'.
Only the Luna CSP will be able to use this data!
Registered 1 partition(s) for use by the Luna CSP!

All available Partitions are presented for you to register or not.
```

3. Install and/or configure your application(s).
4. Run each of your applications once to use SafeNet CSP.
5. Enter the following command to strongly protect the registered challenges:

   register /partition /strongprotect *

> **CAUTION:** You must run **register /strongprotect** to ensure the protection of the HSM partition passwords.

> **Note:** Once you run the **/strongprotect** option, only those users that existed previous to the **/strongprotect** command are allowed to use the SafeNet CSP. If the **/strongprotect** option is not used, then any/all users can use the SafeNet CSP.

6. Enter the following command to reconnect to the library:

   register.exe /library

7. Run all applications as usual.

To register an HA partition

When registering an HA Partition for use, follow these steps.

1. Enter the following command and respond to the prompts:

   C:\Program Files\SafeNet\LunaClient\CSP> **register /highavail**

> **Note:** Use the **/highavail** option only if you have HA set up for your SafeNet Enterprise HSMs.

2. For example:

```
************************************************************
SafeNet Luna CSP, Partition Registration
Protect the HSM's challenge for the selected partitions.
NOTE:
This is a WEAK protection of the challenge!!
After you have configured all applications that will use
the Luna CSP, and run them once, you MUST run:
 register /partition /strongprotect *
to strongly protect the registered challenges!!
************************************************************
This procedure is a destructive procedure and will completely replace
any previous settings!!
Do you wish to continue?: [y/n]
Do you want to register the partition named 'nes'? [y/n]:
Please enter the SafeNet Network HSM challenge for the partition 'nes' :
Success registering the ENCRYPTED challenge for partition 'nes'.
```



```
Only the Luna CSP will be able to use this data!
Registered 1 partition(s) for use by the Luna CSP!
```

Note: If you are using HA, then only the HA virtual partition is presented for registering.

3. Install and/or configure your application(s).
4. Run each of your applications once to use SafeNet CSP.
5. Enter the following command to strongly protect the registered challenges:

   register /partition /strongprotect *

CAUTION: You must run **register /strongprotect** to ensure the protection of the HSM partition passwords.

Note: Once you run the **/strongprotect** option, only those users that existed previous to the **/strongprotect** command are allowed to use the SafeNet CSP. If the **/strongprotect** option is not used, then any/all users can use the SafeNet CSP.

6. Enter the following command to reconnect to the library:

   register.exe /library

7. Run all applications as usual.

Registering the Cryptographic Algorithms to be Performed in Software

Certain operations (symmetric), such as the hash operation may be performed faster in software than on the SafeNet HSM. The **register /algorithms** command allows you to choose which algorithms to de-register from the SafeNet HSM. The trade-off is a gain in speed, at the cost of some security (exposing the operation in software). Signing and other asymmetric operations are always done on the HSM.

To register algorithms for software-only use

1. Enter the following command and respond to the prompts:

   ```
   C:\Program Files\SafeNet\LunaClient\CSP> register /algorithms
   ```

2. You are prompted for yes or no responses about which algorithms are to be registered for software-only use. For example:

```
*********************************************************************
SafeNet Luna CSP, Algorithm Registration

Register algorithms to be done in software by the Microsoft CSP(s).
BY DEFAULT, ALL ALGORITHMS ARE DONE IN HARDWARE BY THE SafeNet Network HSM.
ONLY NON RSA ALGORITHMS MAY BE CONFIGURED FOR SOFTWARE.
RSA PUBLIC/PRIVATE ALGORITHMS WILL ALWAYS BE IN HARDWARE.
*********************************************************************
Do you want algorithm 'CALG_RC2', done in software?(y/n):
Do you want algorithm 'CALG_RC4', done in software?(y/n):
Do you want algorithm 'CALG_RC5', done in software?(y/n):
Do you want algorithm 'CALG_DES', done in software?(y/n):
Do you want algorithm 'CALG_3DES_112', done in software?(y/n):
Do you want algorithm 'CALG_3DES', done in software?(y/n):
Do you want algorithm 'CALG_MD2', done in software?(y/n):
```



```
Do you want algorithm 'CALG_MD5', done in software?(y/n):
Do you want algorithm 'CALG_SHA', done in software?(y/n):
Do you want algorithm 'CALG_MAC', done in software?(y/n):
Do you want algorithm 'CALG_HMAC', done in software?(y/n):
Success registering software only algorithms:
CALG_RC2,CALG_RC4,CALG_RC5,...!
```

If you chose **no** for all prompts, then all algorithms revert to hardware and the following is displayed:

```
All algorithms have been de-registered and will now only be done in hardware!
```

Enabling Key Counting

Key counting allows you to specify the maximum number of times that a key can be used. It sets the upper limit from 0 to MAX(UInt32).

To enable key counting

1. Enter the following command and respond to the prompts. Enter the key usage limit, or enter 0 to turn off the feature:

   C:\Program Files\SafeNet\LunaClient\CSP> **register /usagelimit**

   For example:

   ```
   C:\Program Files\SafeNet\LunaClient\CSP>register /usagelimit
   register v1.0.1

   Enter the key usage limit: 2000

   Successfully configured the key usage limit to 2000 uses.
   ```

KSP for CNG

CNG (Cryptography Next Generation) is Microsoft's cryptographic application programming environment (API) replacing the Windows cryptoAPI (CAPI). CNG is applicable to Windows Server 2008 and Windows Server 2012. CNG adds new algorithms along with additional flexibility and functionality, compared with the old API.

Just as SafeNet provides our CSP for applications running in older Windows crypto environments (and JSP for Java), we offer KSP to allow your Windows Server 2008 CNG applications to make use of the SafeNet HSM. You can still use CSP with Windows Server 2008 and CAPI for your legacy applications, but future development will all take place using CNG, for which you will need to install KSP.

KSP must be installed on any computer that is intended to act via CNG as a Client of the HSM, running crypto operations in hardware. You need KSP to integrate SafeNet cryptoki with CNG and to use the newer functions and algorithms in Microsoft IIS.

After you register the SafeNet HSM partitions with SafeNet KSP, your KSP code should work in the same manner whether our HSM (crypto provider) is selected, or the default provider is used.

Note: TRANSITION ISSUES Be aware when working in a mixed environment or updating applications that previously used CAPI and the SafeNet CSP - the new algorithms supported by CNG (such as SHA512 and ECDSA) in Certificate Services are not recognized by systems that use CAPI. If Certificate Services is configured to use any of these new Algorithms then the signed certificates can be installed only on systems that are aware of these new algorithms. Any of the systems that use CAPI will not be able to use this feature. The installation of certificate will fail.

Installing KSP

KSP is installed using the SafeNet Client installer. Note that it is not installed by default and must be explicitly selected when you install the SafeNet Client. You can also install KSP after you install the SafeNet Client by re-running the installer.

The KSP installer installs the following utilities in the C:\Program Files\SafeNet\LunaClient\KSP folder:

Utility name	Description
KspConfig.exe	A GUI utility used to configure KSP.
kspcmd.exe	A command-line utility used to configure KSP.
ksputil.exe	A command-line utility used to make keys available to other clients, such as in a clustering configuration.
ms2Luna.exe	A command-line utility used to migrate software-based keys to a SafeNet HSM.

Configuring KSP

After installing KSP, use the KSP configuration wizard to register your HSM Partitions for use with CNG. The KSP configuration tool secures the Password for each HSM Partition such that only the user for which the Password was secured is able to un-secure it.

Briefly, the important points are:

- Register the cryptoki to be used.

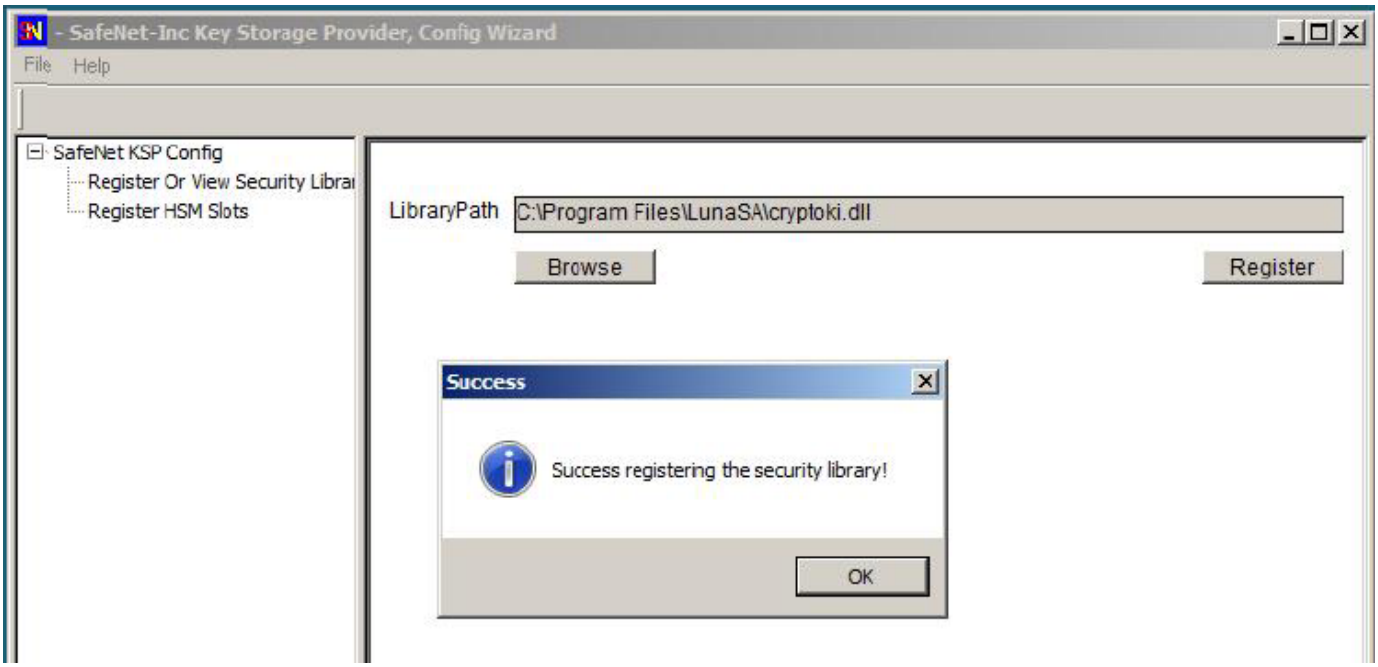
- Register the slot-to-be-used to the local admin (which allows the admin to interact with the slot)
- Register the slot-to-be-used to the local system (which allows the operating system to interact with the slot).

> **Note:** Only the Administrator or a member of the Administrators group can run "KspConfig.exe". The SafeNet KSP can be used by any application that acquires the context of the SafeNet KSP. All users who login and use the applications that acquired the context have access to the SafeNet KSP.

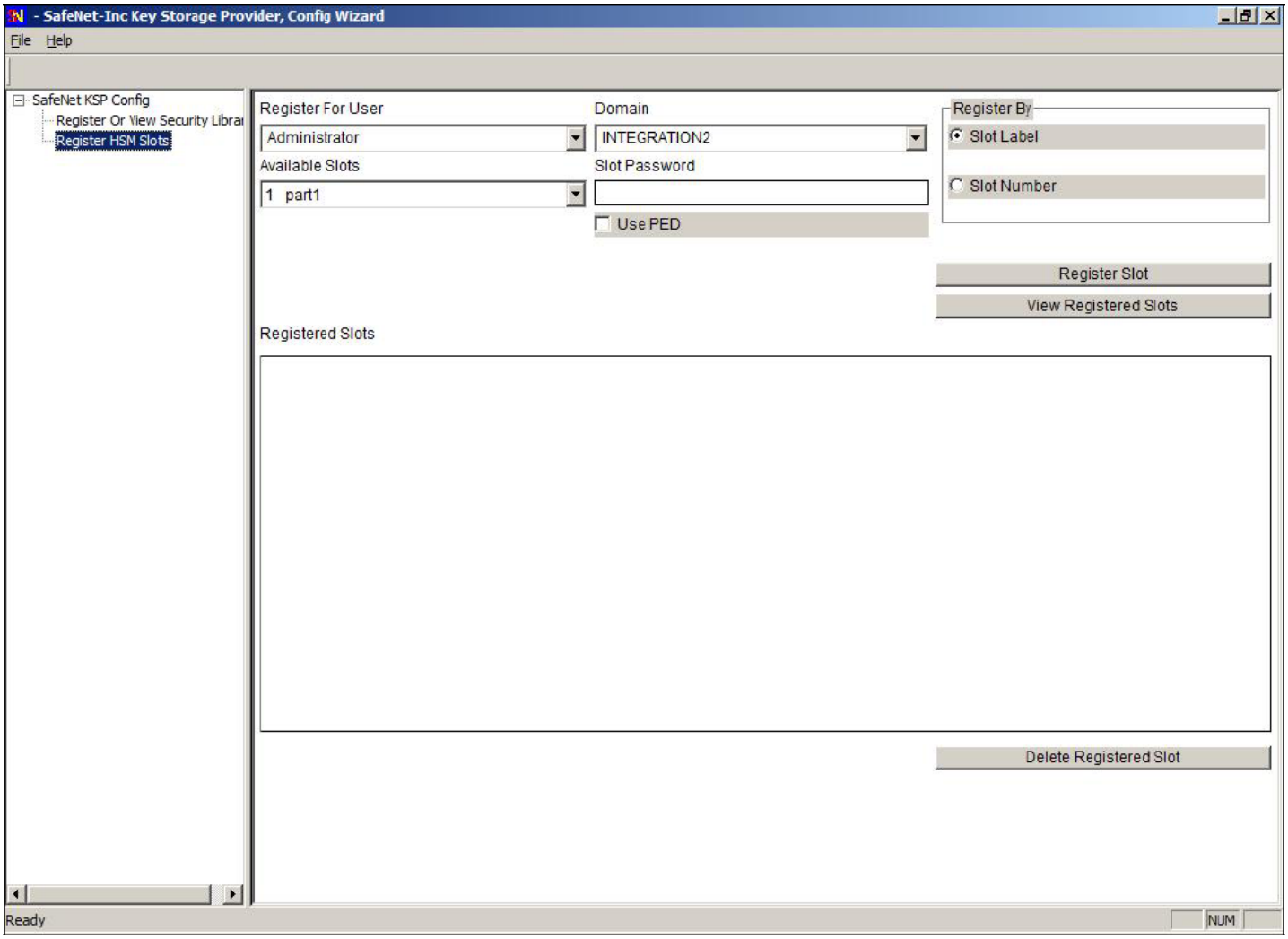
To configure KSP

1. Go to C:\Program Files\SafeNet\LunaClient\KSP and launch KspConfig.exe (the KSP configuration wizard).
2. In the left-hand pane (tree view) double-click "Register Or View Security Library"

3. In the right-hand pane, browse to the library C:\Program Files\SafeNet\LunaClient\cryptoki.dll and click **Register**.
4. When the success message appears, click **OK**.

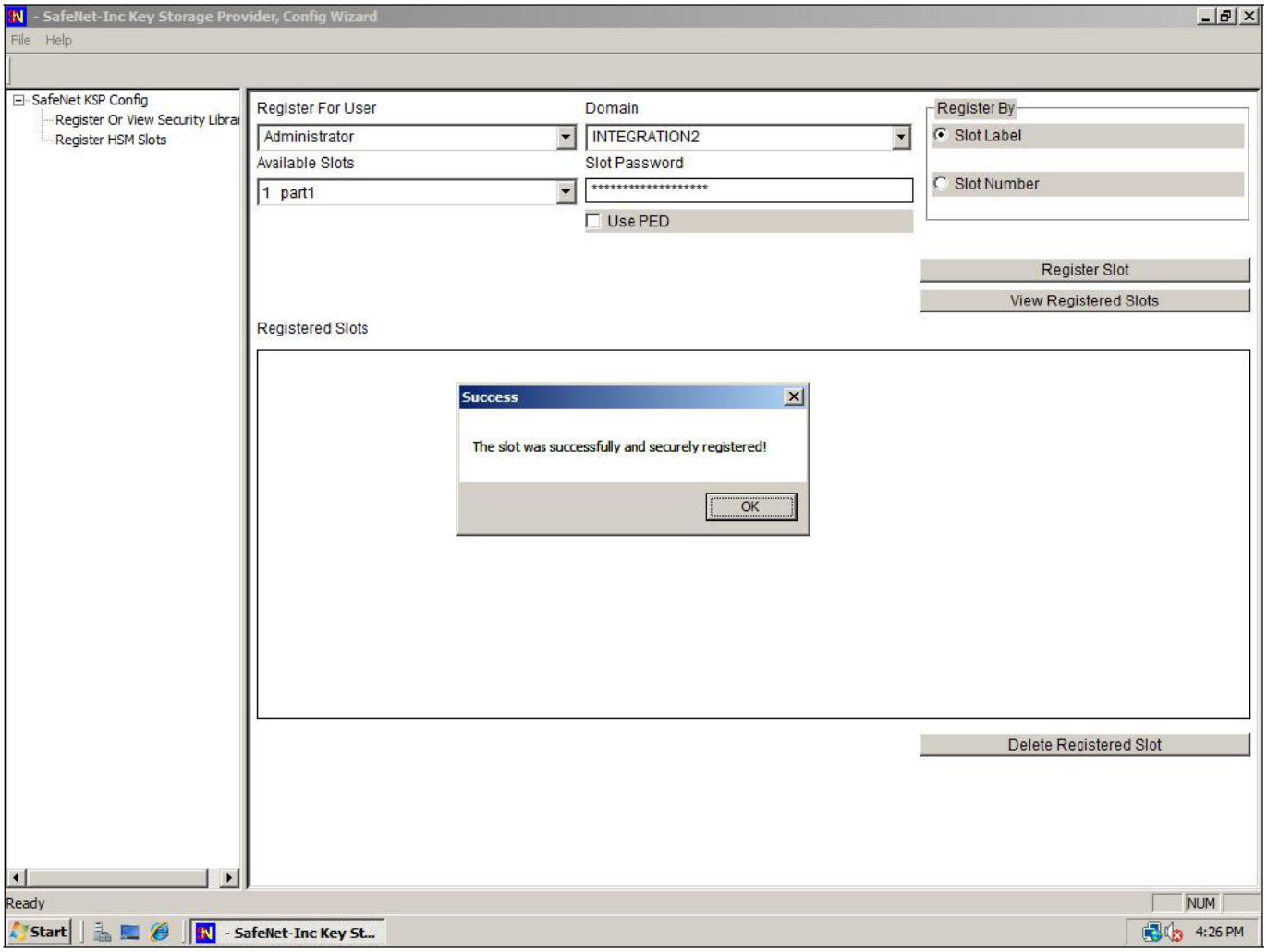


5. Return to the left-hand pane and double-click "Register HSM Slots", and click [Next]. In general, we recommend that you register by slot label, rather than slot number, if you are using an HA configuration .

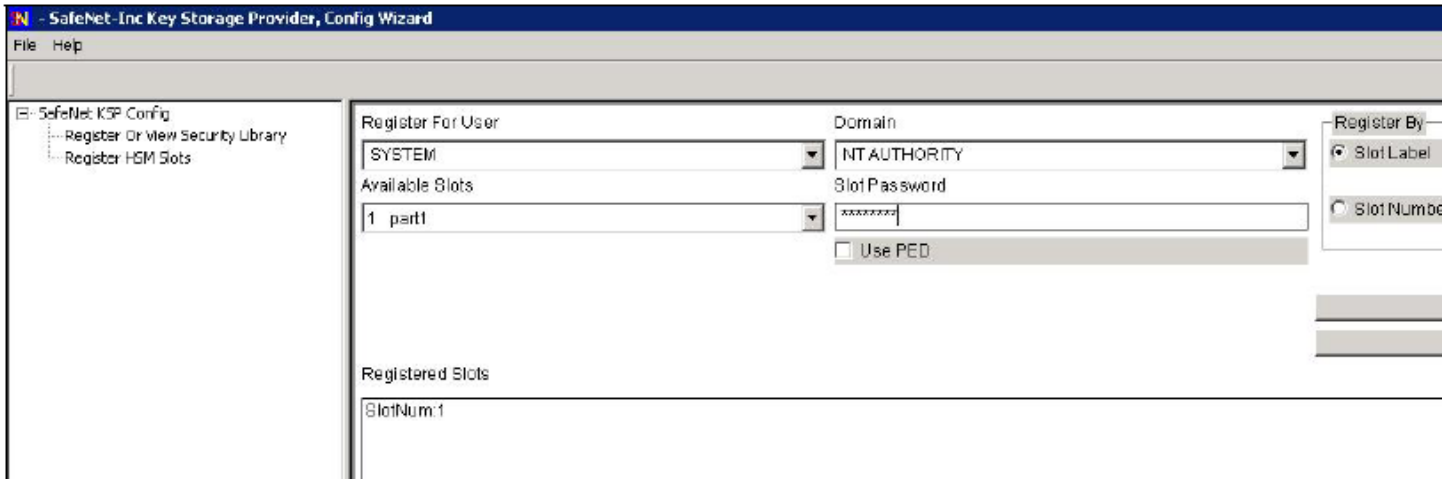


6. In the "Slot Password" field, type in the password for the indicated slot.To the right of the window, click the [Register Slot] button to register the slot for Domain/User. A success message appears.

   Note that the "Register for User" field should be Administrator (or the admin equivalent account that will be managing this setup) and "Domain" should match the domain or local computer with which you are logged in.



7. Return to the "Domain" pull-down list select "SYSTEM" under "Register for User"and select "NT AUTHORITY" under "Domain", supply the password for the slot being registered, and again click Register Slot] to complete the KSP configuration.



8. Once you have the slots registered, you can begin connecting with your client application to perform crypto operations in your HSM Partitions (or HA virtual slots). If a SafeNet-tested Integration procedure for your application is not available for download from the SafeNet website, contact SafeNet Customer Support.

If It Doesn't Work?

When you open the KspConfig program, if it fails to display a list of available slots, then it might be that you have not properly set up your SafeNet HSM.

Open a Windows Command Prompt window, change directory to the "C:\Program Files\SafeNet\LunaClient\" directory, and use the "lunacm" command-line utility to see and modify the status of the HSM and HSM Partitions.

Algorithms Supported

Here, for comparison, are the algorithms supported by our CSP and KSP APIs.

Algorithms supported by the SafeNet CSP

CALG_RSA_SIGN

CALG_RSA_KEYX

CALG_RC2

CALG_RC4

CALG_RC5

CALG_DES

CALG_3DES_112

CALG_3DES

CALG_MD2

CALG_MD5

CALG_SHA

CALG_SHA_256

CALG_SHA_384

CALG_SHA_512

CALG_MAC

CALG_HMAC

Algorithms supported by the SafeNet KSP

NCRYPT_RSA_ALGORITHM

NCRYPT_DSA_ALGORITHM

NCRYPT_ECDSA_P256_ALGORITHM

NCRYPT_ECDSA_P384_ALGORITHM

NCRYPT_ECDSA_P521_ALGORITHM

NCRYPT_ECDH_P256_ALGORITHM

NCRYPT_ECDH_P384_ALGORITHM

NCRYPT_ECDH_P521_ALGORITHM

NCRYPT_DH_ALGORITHM

NCRYPT_RSA_ALGORITHM

Enabling Key Counting

Key counting allows you to specify the maximum number of times that a key can be used. It sets the upper limit from 0 to MAX(UInt32).

To enable key counting

1. Enter the following command and respond to the prompts. Enter the key usage limit, or enter 0 to turn off the feature:

   C:\Program Files\SafeNet\LunaClient\KSP> **kspcmd usagelimit**

   For example:

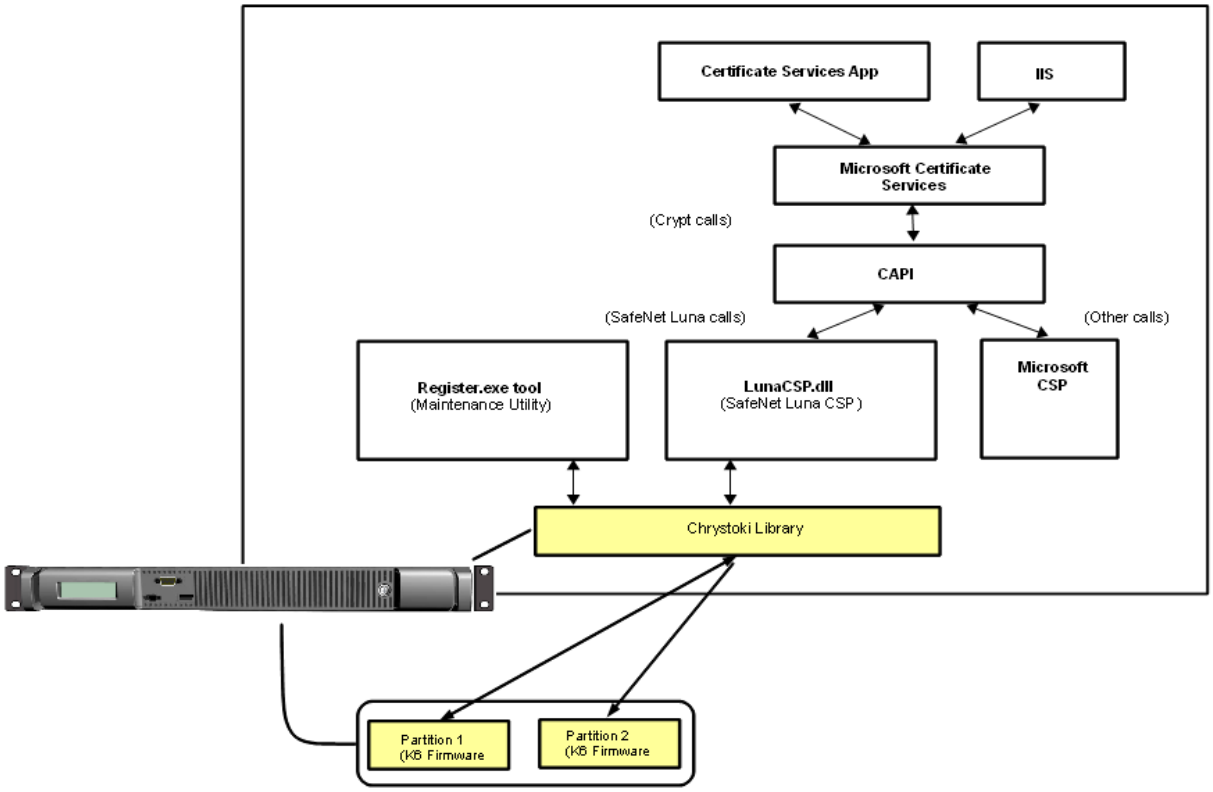
```
C:\Program Files\SafeNet\LunaClient\KSP>kspcmd usageLimit 2000
This Servers Host Name is: LUNA_CLIENT and the logged on user is: admin@LUNA_CLIENT

Enter the key usage limit: 2000

Successfully configured the key usage limit to 2000 uses.


C:\Program Files\SafeNet\LunaClient\KSP>kspcmd u
This Servers Host Name is: LUNA_CLIENT and the logged on user is: admin@LUNA_CLIENT

Warning, max key usage is already set to 2000.
Changing this will not modify previously created keys!
Only keys created subsequent to making this change will be affected!
Do you wish to continue?[y/n]:
```

SafeNet CSP Calls and Functions

For integration with Microsoft Certificate Services and other applications, the LunaCSP.dll library accepts Crypt calls and gives access to token functions (via CP calls) as listed in this section. Key pairs and certificates are generated, stored and used on the SafeNet HSM.

The diagram below depicts the relationship of the SafeNet components to the other layers in the certificate system.

Figure 1: SafeNet CSP architecture



Note, in the diagram, that the SafeNet CSP routes relevant calls through the statically linked Crystoki library to the HSM via CP calls. Other calls from the application layer – those not directed at the token/HSM, and not matching the SafeNet CSP supported functions (see next section) – are passed to the Microsoft CSP.

Programming for SafeNet HSM with SafeNet CSP

The SafeNet CSP DLL exports the following functions, each one corresponding to an equivalent (and similarly named) Crypt call from the application layer:

- CPAcquireContext
- CPGetProvParam
- CPSetProvParam
- CPReleaseContext
- CPDeriveKey
- CPDestroyKey
- CPDuplicateKey
- CPExportKey
- CPGenKey
- CPGenRandom

- CPGetKeyParam
- CPGetUserKey
- CPImportKey
- CPSetKeyParam
- CPDecrypt
- CPEncrypt
- CPCreateHash
- CPDestroyHash
- CPGetHashParam
- CPHashData
- CPHashSessionKey
- CPSetHashParam
- CPSignHash
- CPVerifySignature

Note: The CPVerifySignature function is able to verify signatures of up to 2048 bits, regardless of the size of the signatures produced by CPSignHash. This ensures that the CSP is able to validate all compatible certificates, even those signed with large keys.

Note: The MSDN (Microsoft Developers Network) web site provides syntax and descriptions of the corresponding Crypt calls that invoke the functions in the above list.

Algorithms

SafeNet CSP supports the following algorithms:

- CALG_RSA_SIGN [RSA Signature] [256 - 4096 bits]. The CSP uses the RSA Public-Key Cipher for digital signatures.
- CALG_RSA_KEYX [RSA Key Exchange] [256- 4096 bits] The CSP must use the RSA Public-Key Cipher key exchange. The exchange key pair can be used both to exchange session keys and to verify digital signatures.
- CALG_RC2 [RSA Data Securities RC2 (block cipher)] [8 - 1024 bits].
- CALG_RC4 [RSA Data Securities RC4 (stream cipher)] [8 - 2048 bits].
- CALG_RC5 [RSA Data Securities RC5 (block cipher)] [8 - 2048 bits].
- CALG_DES [Data Encryption Standard (block cipher)] [56 bits].
- CALG_3DES_112 [Double DES (block cipher)] [112 bits].
- CALG_3DES [Triple DES (block cipher)] [168 bits].
- CALG_MAC [Message Authentication Code] (with RC2 only).
- CALG_HMAC [Hash-based MAC].
- CALG_MD2 [Message Digest 2 (MD2)] [128 bits].

- CALG_MD5 [Message Digest 5 (MD5)] [128 bits].
- CALG_SHA [Secure Hash Algorithm (SHA-1)] [160 bits].
- CALG_SHA224 [Secure Hash Algorithm (SHA-2)] [224 bits].
- CALG_SHA256 [Secure Hash Algorithm (SHA-2)] [256 bits].
- CALG_SHA384 [Secure Hash Algorithm (SHA-2)] [384 bits].
- CALG_SHA512 [Secure Hash Algorithm (SHA-2)] [512 bits].

Note: If you intend to perform key exchanges between the SafeNet CSP and the Microsoft CSP with RC2 keys, the attribute KP_ EFFECTIVE_KEYLEN must be set to 128 bits. For RC2 and RC4, the salt value of the keys must be transferred by making a call to get the salt value of the original key and to set the salt value of an imported key. This is done with the CryptGetKeyParam(KP_ SALT) and CryptSetKeyParam(KP_ SALT) functions respectively.