THALES

Vormetric Data Security Platform

Vormetric Application Encryption (VAE)

Installation and API Reference Guide

Version 6.2.0

50-1000040-02

MARCH 12, 2019
DOCUMENT VERSION 1

Vormetric Data Security Platform

Installation and API Reference Guide

Version 6.2.0

March 12, 2019, Document Version 1

COSTS, CLAIMS, OR LIABILITIES ARISING OUT OF ANY AGREEMENT BETWEEN LICENSEE AND ANY THIRD PARTY. NO PROVISION OF ANY AGREEMENT BETWEEN LICENSEE AND ANY THIRD PARTY SHALL BE BINDING ON THALES.

Protected by U.S. patents:

6,678,828

6,931,530

7,143,288

7,283,538

7,334,124

Contents

PREFACE

The *Installation and API Reference Guide* describes the functions and features of VAE. It provides descriptions, syntax, and usage examples for each of the actions and data types for the VAE solution. This document describes the Vormetric implementation of the PKCS #11 standard.

DOCUMENTATION VERSION HISTORY

The following table describes the changes made for each document version.

Table 1: Documentation Changes

Product/Document Version	Date	Changes
6.2.0	3/12/19	Added new section " Running Samples with the Java Wrapper"; added GCM support info; added information on Identities for creating a key directly in a key group; added Content for new Java Wrapper; added information on asymmetric keys
6.1.0	6/24/18	6.1.0 release includes many doc fixes and content for Identities feature, Docker containers, Random API, and SQL Server.
6.0.2 v1	1/26/18	Minor edits, removed "Typographical Conventions," added C_UnwrapKey; edited C_GenerateKey and C_CreateObject; added "Storing keys on the server by default" and "Automated Key Versioning."
6.0v2	12/14/17	Compatibility matrix and supported operating systems removed to Release Notes only.
6.0 v1	10/15/17	6.0 features
5.2.5 Patch v2	6/30/17	Included information for FF1.
5.2.5 Patch v1	12/16/16	Update System Requirements. Fix Return Values in Digest functions.
5.2.5 v3	9/27/16	Added Digest functions and FIPS. Partial copy edit to fix wording.
5.2.5 v2	8/5/16	Fixed some typos and added note in Installation section about key cache only in memory and not written to disk.
5.2.5 v1	6/6/16	Fixed some typos and grammatical errors.
5.2.4 v2	3/23/16	Fixed some typos and information errors with the naming of the binaries for installation.

Table 1: Documentation Changes

Product/Document Version	Date	Changes
5.2.4 v1	2/16/16	Included information for the new feature of Format Preserving Encryption (FPE), Shared Secret Registration, and fixed technical and editorial issues.

ASSUMPTIONS

This documentation assumes knowledge of C and PKCS #11.

The system administrator must have root permissions for the systems on which Vormetric Application Encryption (VAE) software is installed.

RELATED DOCUMENTS

The following documents are available to registered users on the Vormetric Web site at:

https://help.vormetric.com

- *Vormetric Transparent Encryption Agent Installation and Configuration Guide*
- *Vormetric Data Security Platform Administrators Guide*

GUIDE TO VAE DOCUMENTATION

The following related documents are available to registered users on the Vormetric Web site at https://support.vormetric.com

1. **Data Security Manager (DSM) Installation and Configuration Guide**. In most cases, you will probably use an existing DSM, but use this document if you need to install a new Data Security Manager (DSM) Appliance.
2. **Vormetric Application Encryption (VAE) Installation and API Reference Guide**. (This book.) For developers who want to use application encryption with Vormetric's implementation of PKCS #11.
3. **Vormetric Security Intelligence Configuration Guide**. Use this to integrate your Vormetric VAE events logs with the ArcSight ESM, Splunk, or IBM QRadar

4. **Vormetric Data Security (VDS) Platform Event and Log Messages Reference**. A listing of all the VDS Platform event and log messages with severity, long and short form, and description.

Searching through all the documents

Technical information for Vormetric products can be spread across many documents. Instead of searching through each individual document to find the information you need, you can use the following procedure to search all of the VTE documents with a single search in Windows (the same process should work for UNIX/Linux):

1. Copy all the .pdf files of a specific product into a single directory. For example, using the Vormetric Transparent Encryption:

```
C:\Documents\PDFs\5.2.3>dir
Admin_Guide_V1.pdf
Agent_Install_&_Config_Guide_v2.pdf
Data_Transformation_Guide_v1.pdf
DSM_Automation_Reference_v1.pdf
DSM_Install_Guide_v1.pdf
Event_&_Log_Messages_Ref_v1.pdf
GettingStarted_v1.pdf
VSI_Reference_v1.pdf
RN_DSM.pdf
RN_Linux.pdf
RN_RHEL7.pdf
RN_UNIX.pdf
RN_Windows.pdf
VDSCompatibilityMatrix.pdf
```

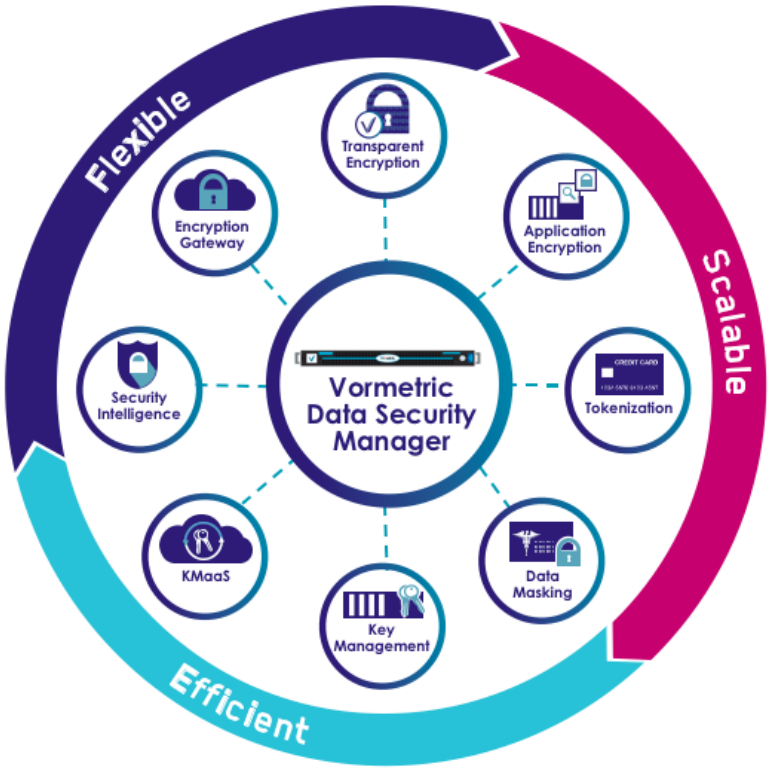
2. Bring up Adobe Reader or Adobe Acrobat.
3. Open any PDF file from that directory: **File > Open > Select File**.
4. Click **Edit > Advanced Search**.
5. Under "Where *would you like to search?"* click "All **PDF Documents in**", then select the directory containing all the VTE PDF files. In this case, `C:\Documents\PDFs\5.2.3`
6. In the "What *word or phrase would you like to search for?*" enter the search phrase and click search.

You can do this with any set of PDF files.

Vormetric Data Security Platform—Overview

The Vormetric Data Security (VDS) Platform protects data wherever it resides. The platform solves security and compliance issues with encryption, key management, privileged user access control, and security intelligence logging. It protects data in databases, files, and Big Data nodes across public, private, hybrid clouds and traditional infrastructures.

Figure 1: The Vormetric "Solar System"



The platform consists of products that share a common, extensible infrastructure. At the heart of the platform is the DSM, which coordinates policies, keys, and the collection of security intelligence, all of which is managed and observed through your browser. In addition to the DSM, the Vormetric Data Security Platform consists of the following products:

- **Vormetric Application Encryption (VAE)** provides a framework to deliver application-layer encryption such as column- or field-level encryption in databases, Big Data, or PaaS applications. VAE is an application encryption library providing a standards-based API to do cryptographic and encryption key management operations into existing corporate applications.

- **Vormetric Cloud Encryption Gateway (VCEG)** safeguards files in cloud-storage environments, including Amazon S3 and Box. VCEG encrypts sensitive data before it is saved to the cloud, enabling security teams to establish visibility and control around cloud assets.
- **Vormetric Key Management (VKM)** centralizes 3rd-party encryption keys and stores certificates securely. It provides standards-based enterprise encryption key management for Transparent Database Encryption (TDE), KMIP-compliant devices, and offers vaulting and inventory of certificates.
- **Vormetric Protection for Teradata Database** provides granular controls to secure assets in Teradata environments. It simplifies the process of using column-level encryption in your Teradata database. It provides documented, standards-based APIs and user-defined functions (UDFs) for cryptographic and key management operations.
- **Vormetric Security Intelligence** provides comprehensive logging combined with Security Information Event Management (SIEM) systems to detect advanced persistent threats and insider threats. In addition, the logs satisfy compliance and regulatory audits.
- **Vormetric Tokenization with Dynamic Data Masking (VTS)** replaces sensitive data in your database with unique identification symbols called tokens. This reduces the number of places that clear-text sensitive data reside, and thus reduces the scope of complying with PCI DSS and corporate security policies.
- **Vormetric Transparent Encryption (VTE)** secures any database, file, or volume across your enterprise without changing the applications, infrastructure, or user experience.

SERVICE UPDATES AND SUPPORT INFORMATION

The license agreement that you have entered into to acquire the Thales products (“License Agreement") defines software updates and upgrades, support and services, and governs the terms under which they are provided. Any statements made in this guide or collateral documents that conflict with the definitions or terms in the License Agreement, shall be superseded by the definitions and terms of the License Agreement. Any references made to “upgrades” in this guide or collateral documentation can apply either to a software update or upgrade.

For support and troubleshooting issues:

- http://help.thalesesecurity.com
- http://support.vormetric.com
- support@thalesesecurity.com
- (877) 267-3247

For Thales Sales:

- http://enterprise-encryption.vormetric.com/contact-sales.html
- sales@thalesesec.net
- (408) 433-6000

1 Vormetric Application Encryption

Vormetric Application Encryption (VAE) enables data encryption at the application level. Applications can use VAE to encrypt a column in a database or a field, such as credit card numbers or Social Security numbers. VAE can also be used to encrypt an entire data file or directory.

VAE provides some of the same functionality as Vormetric Transparent Encryption (VTE), but they differ significantly. VTE does not support application-level data encryption, but it supports encryption of an entire data file or directory through an application's interface. VAE and VTE both offer this file-level encryption, but they use different mechanisms for it.

NOTE: The Vormetric Application Encryption software contains the same components as the Vormetric Key Agent, which is used for non-Oracle or non-MS-SQL applications. The installer and configuration for VAE refer to the Key Agent because the installation and configuration process is the same.

NOTE: VAE does not have a daemon that pings the DSM periodically, so it will not know about changes on the DSM. However, it does pick up new configurations from the DSM every 15 minutes.

Product Overview

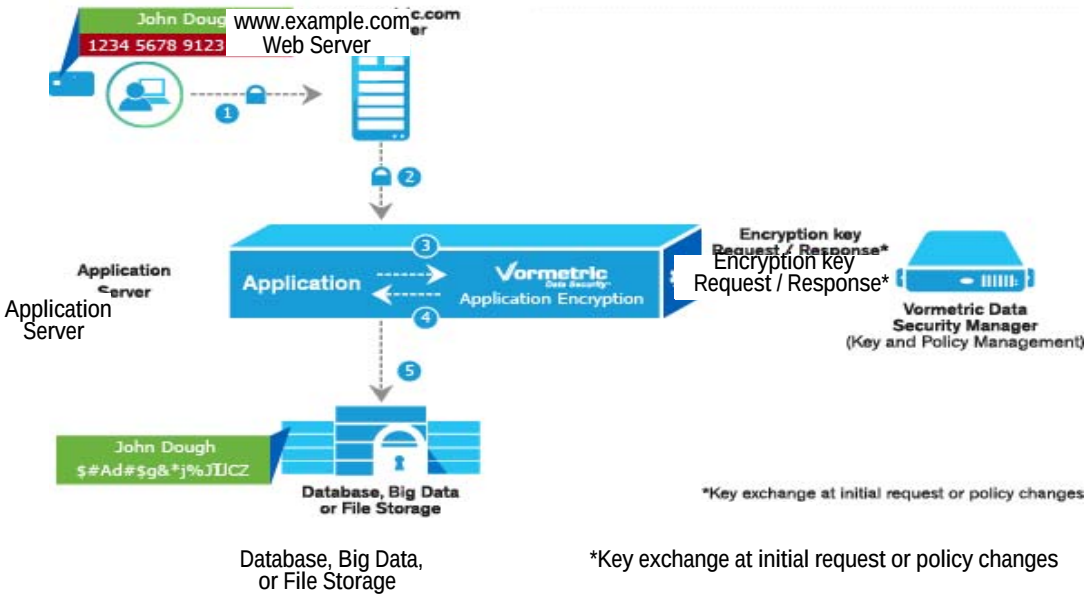
Application Encryption Workflow

The Application Encryption workflow is shown in Figure 1.

1. A user submits personal information to purchase items from a website.
2. The web server sends personal information to an application server.
3. The application calls Vormetric Application Encryption (VAE) to encrypt the sensitive personal data.

4. VAE returns the encrypted value to the application.
5. The application stores the encrypted value.

Figure 1: Application Encryption Workflow



Database, Big Data, or File Storage

*Key exchange at initial request or policy changes

Components

Vormetric Application Encryption (VAE) makes use of several other components of the Vormetric Data Security (VDS) Platform, including:

Data Security Manager (DSM)

The DSM is the central component of the VDS Platform. It consists of a policy engine and a central key and policy manager. The DSM stores and manages host encryption keys, data access policies, administrative domains, and administrator profiles.

Key Agent

The Vormetric Key Agent provides a library that implements the PKCS #11 interface. This library is a dynamically loadable library (`dll`) on Windows and a shared object (`so`) on Linux and UNIX. The Key Agent's PKCS #11 library communicates over a secure channel to the DSM for all significant functionality. The Key Agent can also be referred to as the Vormetric Application Encryption Agent.

Functionality

VAE APIs support real-time I/O encryption and decryption of “at rest” data and log files. Data is encrypted by adding VAE API calls to existing applications.

The APIs in the Vormetric library are a subset of the PKCS #11 specification version 2.20. They are platform-independent to cryptographic tokens, and are traditionally used for HSMs (hardware security modules) and smartcards.

VAE supports single and multi-part encryption and decryption using RSA 1024, RSA 2048, AES 128, and AES 256.

VAE provides API support for the following:

- Create a key
- Find a key
- Destroy a key
- Export a key
- Import a key
- Versioned key support
- Encrypt data
- Decrypt data
- Sign data
- Verify data signature
- Compute the digest for data with a key (HMAC)
- Thales-specific opaque objects
- Random number generation

NOTE: See “API Reference” on page 57 for a complete list of the supported APIs.

Key Cache Options

You can choose to store keys on the DSM only or allow them to also be cached in memory on the host. By default, newly created keys are cached on the host. For greater security, you can

keep the keys on the DSM; however, doing this will affect performance. Table compares the advantages of each option.

Table 1: Key Storage Options

	Key stored on the DSM	Key cached on host
Performance	- Network round-trip slows performance	- Very fast for bulk encryption/high-volume transactions of data.
Security	- Most secure. The key never leaves the DSM. - In compliance with PCI security standards	- Key is cached on the client in memory. - In compliance with PCI security standards

NOTE: For supported agent operating systems and compatibility matrices, see "Supported OS and Compatibility" on page 113.

2

Vormetric Encryption Concepts

This chapter is intended for readers who are not familiar with the Thales/Vormetric security solutions. It includes detailed descriptions of how the VAE communicates with the DSM, along with the various ways that Thales/Vormetric handles keys for encryption and decryption. It also describes how the solution authenticates for a key with the PKCS system of PINs.

VAE and DSM

The VAE encryption library must have a host to authenticate it, and the DSM is responsible for this activity. Authentication is complete after the VAE is registered with DSM. Should the user need to do registration again, they run `register_host` by itself, from the bin directory.

This section discusses the two methods available for registration, along with communication with the DSM during operation. For details on the procedures, refer to the installation chapter in this manual and to the DSM documentation.

Fingerprint

The fingerprint method requires the DSM Security Administrator to add the FQDN or IP address of each host to the DSM before registering VAE.

During the registration, the DSM generates the certificate and passes it down to the VAE with the fingerprint. The security administrator must verify the fingerprint to make sure the certificate is valid.

Shared Secret

The shared secret method requires the DSM Security Administrator to create a shared secret registration password—a case-sensitive string of characters—for auto-registering a host in a domain or host group.

VAE installers use this shared secret to add and register hosts to the DSM for a domain or host group. This method can automatically add host names or IP addresses to the DSM. This approach eliminates the need to verify that the host and DSM share valid certificates. You can add multiple hosts dynamically, during VAE installation and registration, with a single shared secret password.

If you choose the Shared Secret method, you need the DSM Administrator to supply a shared secret for the domain, or host group, in which the new host will reside. Then, obtain the shared secret and the validity period (one hour, day, week, or month). You can then register within that period.

Run-time DSM Functionality

The VAE APIs give the developer ways to share existing keys with the DSM. The two methods that comprise this functionality include `C_CreateObject`, which imports key bytes onto the DSM and creates a new key in the DSM, and `C_WrapKey`, which exports a wrapped pair of keys (one from the DSM and another key imported to the DSM).

NOTE: See "C_CreateObject" on page 81 and "C_WrapKey" on page 76 for complete details.

NOTE: The DSM administrator must set all of the VAE key templates being to retrieve these keys. Consult the Vormetric Key Management (VKM) Manual for complete details.

Key Management

Every encryption system depends on its keys to make it both unique and reversible. Poor key management is one of the most common causes of an unauthorized hack.

Manual key management was the original solution, but many of today's systems have too much traffic and complexity to make this approach possible any longer. When secured data needs to be available at any time, and when it also needs to be protected as it moves across the network, automated key management is essential.

The Thales e-Security's solution, which includes VAE and DSM, ensures that an overarching security policy is enforced throughout the network, resulting in lower cost and reduced risk of security loopholes.

Key Headers

Thales e-Security provides optional header content for its generated ciphertext output. Generating ciphertext with headers allows each key to be formally versioned, and it also prevents the need for by trial and error data decryption due to a lack of a unique identifier for any given key.

Thales e-Security offers three header formats, all of which are based on the DPM format from RSA. The three variants available are:

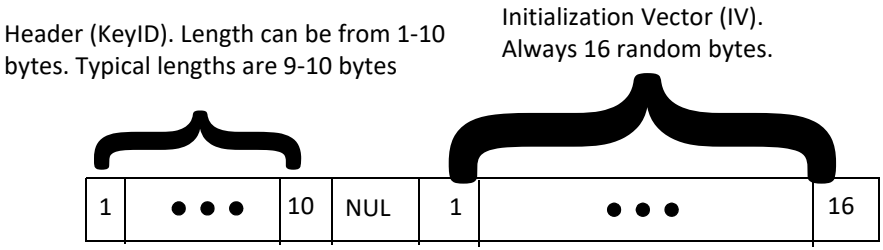
- Version 1.5
- Version 2.1
- Version 2.7

NOTE: Keys created with the DSM GUI have no DPM attributes.

DPM Version 1.5

Version 1.5 of the DPM header contains three elements, beginning with the header ID, followed by a null byte and an Initialization Vector (IV), respectively. Figure 2 shows the bit layout of this header.

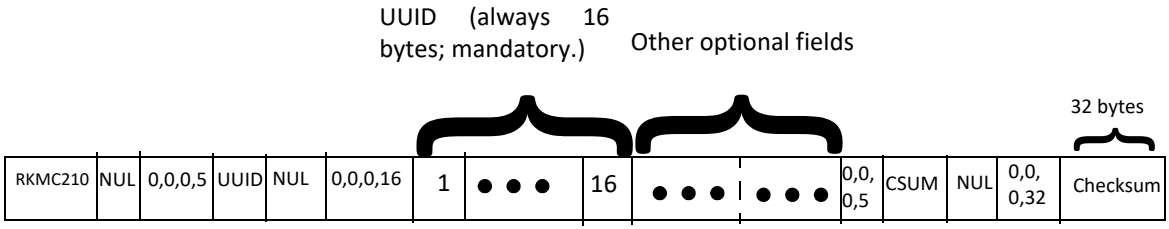
Figure 2: Header Version 1.5 Bit Diagram



DPM Version 2.1

Version 2.1 of the DPM header contains a minimum of 12 elements, beginning with a fixed ASCII string, followed by a NUL byte, a length field, the string “UUID,” another NUL, a second length field, the UUID itself, other optional fields, a length field, the string “CSUM,” another NUL, a second length field, and finally the checksum. Figure 3 shows the bit layout of this header.

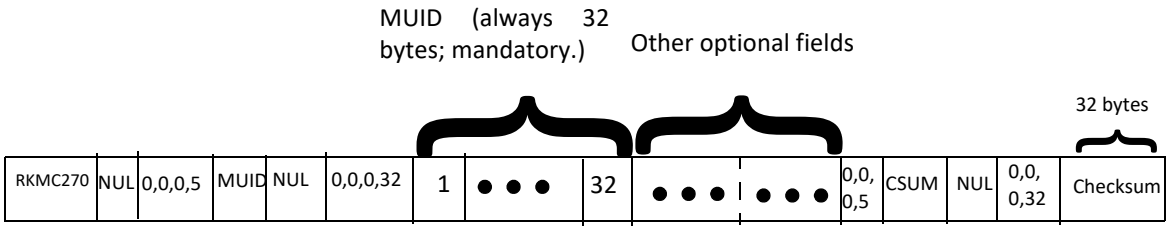
Figure 3: Header Version 2.1 Bit Diagram



DPM Version 2.7

Version 2.7 of the DPM header is identical to version 2.1, except that (a) the UUID is replaced by a 32-byte MUID, and the string “UUID” is replaced by the string “MUID.Figure 4 shows the bit layout of this header.

Figure 4: Header Version 2.7 Bit Diagram



Headers in the API

The `C_EncryptInit` method in the VAE API has bits that can force the key to use one of the above header types, or no header at all. Similarly the `C_DecryptInit` method must include the correct header used (if any) in its input parameters.

For `C_EncryptInit`, the bits that can be ORed to the encryption method value are one of:

```
CKM_THALES_V27HDR                       0x04000000
CKM_THALES_V21HDR                       0x02000000
CKM_THALES_V15HDR                       0x01000000
```

If a version 1.5 header is used, the value

```
CKM_THALES_BASE64                    0x08000000
```

can also be ORed with the method to indicate that the whole output (header and ciphertext) should be BASE64-encoded.

If one of these modifiers is used, the following bit must also be ORed with the method:

```
CKM_VENDOR_DEFINED (0x80000000)
```

For `C_DecryptInit`, the value which can be ORed to the method value is:

```
CKM_THALES_ALLHDR                                    0x07000000
```

and optionally:

```
CKM_THALES_BASE64                    0x08000000
```

If CKM_THALES_BASE64 is set, the header and the ciphertext after the header will be BASE64-decoded first.

The following table shows which header types are compatible with which encryption methods.

Table 2: Header Compatibility with Encryption Methods

	Format	Methods						
		ECB	CBC	CBC-PAD	CTR	FPE	FF1	GCM
C_Encrypt and C_Decrypt	V 1.5	No	Yes	Yes	No	No	No	No
	V1.5 Base-64	No	Yes	Yes	No	No	No	No
	V 2.1	Yes	Yes	Yes	Yes	No	No	No
	V 2.7	Yes	Yes	Yes	Yes	No	No	No
C_EncryptUpdate	V 1.5	No	Yes	Yes	Yes	No	No	No
	V1.5 Base-64	No	Yes	Yes	Yes	No	No	No
	V 2.1	No	Yes	Yes	Yes	No	No	No
	V 2.7	No	Yes	Yes	Yes	No	No	No
C_DecryptUpdate	V 1.5	No	No	No	Yes	No	No	No
	V1.5 Base-64	No	No	No	Yes	No	No	No
	V 2.1	No	No	No	Yes	No	No	No
	V 2.7	No	No	No	Yes	No	No	No

Versioning

In the VAE, a key's version refers to one of the three RSA headers, each of which is a different version of a similar format. Once a key has been created to support versioning, the API can append any or none of these headings to a given cipher text.

Of course, functions that decrypt the data must know which key version to expect.

NIST Key States

Thales e-Security uses stateful keys that are fully compatible with the NIST standard. For convenience, the descriptions of each key state are:

- Preactivation

  A key enters this state immediately after it becomes active, and normally stays there until ready for use. Preactivated keys that have been compromised or are not needed are destroyed instead.

- Active

  Keys in the active state can both encrypt and decrypt information. They are destroyed when their use period ends, or enter the compromised state when they are suspected of being compromised.

  The active state has two valid sub-states: protect-only and process-only.

- Suspended

  Keys can be suspended and reactivated later for several possible reasons. For example, the entity associated with a key could take a leave of absence, or there needs to be an investigation into whether the key was compromised. Suspended keys are not allowed to encrypt data, but they could still decrypt data that was locked before the suspension took place.

  The suspended state has only one valid sub-state, which is process-only.

- Deactivated

  Keys are most often deactivated at the end of their use periods. Deactivated keys are not allowed to encrypt data, but they could still decrypt data that was locked before the deactivation. Deactivated keys should be destroyed as soon as they are no longer needed.

  The deactivated state has only one valid sub-state, which is process-only.

- Compromised

  A compromised key must never be used to encrypt information, and it should be destroyed when no longer needed.

  The compromised state has only one valid sub-state, which is process-only.

- Destroyed

  The key itself has been deleted, although metadata relating to the key might still be preserved for record keeping. Such practice is important if it's later discovered that the key was compromised before it was deleted.

NOTE: Key states are not supported for asymmetric keys.

Opaque Objects

In the Thales/Vormetric environment, opaque objects are used for importing previously unsupported size keys, creating keys with unsupported algorithms, and importing certificates associated with a key. The object itself is not necessarily a key.

Exporting Opaque Objects from the DSM

Opaque objects can be exported back to the VAE from the DSM. The maximum object size is 4KB.

Cached Keys

One valuable feature of the VAE is its ability to cache keys in local memory, instead of relying exclusively on a key cache within the DSM. This local cache speeds up performance on database transactions and small amounts of data. The VAE uses openSSL to perform the encryption locally.

NOTE: For information on how to enable a local key cache, see See "Encryption Use Cases" on page 39.

GCM Support for Symmetric Keys

AES-GCM includes additional input and output parameters to the known CBC and CTR used to verify the plain text result of a decryption use GCM. Create AES-GCM keys on the DSM. The VAE receives and register keys to and the DSM. When implementing GCM consider the following usage restrictions:

- GCM support is only available in the custom Thales Java wrapper. It is not available in the traditional `sun.security.pkcs11` wrapper.
- GCM mode is in C.
- GCM mode is not supported for C#.

- GCM mode is supported in JDK 9 and higher versions.
- GCM works only with `CacheOnHost` keys.
- GCM is not supported for headers.
- GCM only support IV of 12 bytes.

Following is an example of GCM usage:

```
typedef struct CK_GCM_PARAMS {
    CK_BYTE_PTR       pIv;
    CK_ULONG          ulIvLen;
    CK_ULONG          ulIvBits;
    CK_BYTE_PTR       pAAD;
    CK_ULONG          ulAADLen;
    CK_ULONG          ulTagBits;
} CK_GCM_PARAMS;
```

Export Asymmetric Keys

Export an asymmetric key in a wrapping key passed to `C_WrapKey`. The format must be set to `PEM` in `CK_MECHANISM Mechanism`::

```
public static final long CKM_THALES_PEM_FORMAT = 0x00100000L;
```

NOTE: The mechanism is same for `C_UnwrapKey`.

If the wrapping key is symmetric key use the `CKA_THALES_DEFINED` and one of the following wrapping mechanisms:

- `CKM_AES_CBC_PAD`
- `CKM_RSA_PKCS`

In addition to exporting an symmetric key by itself, the following options are supported:

- Wrap a symmetric key with an asymmetric key
- Wrap a symmetric key with a symmetric key
- Wrapping an asymmetric key a symmetric key

When exporting an asymmetric key, or exporting symmetric key with asymmetric key as the wrapper include the following additional `CKA_THALES_DEFINED` mechanism configuration:

```
public static final long CKA_THALES_DEFINED  = 0x40000000L
```

NOTE: In this case the wrapping mechanism is handled in the VAE library and not the DSM.

All the constants should be ORed together to form the mechanism passed in. Refer to the sample code for examples.

`C_WrapKey` example:

```
CK_DEFINE_FUNCTION(CK_RV, C_WrapKey)
  (
    CK_SESSION_HANDLE hSession, /* the session's handle */
    CK_MECHANISM_PTR pMechanism, /* the wrapping mechanism */
    CK_OBJECT_HANDLE hWrappingKey, /* wrapping key */
    CK_OBJECT_HANDLE hKey, /* key to be wrapped */
    CK_BYTE_PTR pWrappedKey, /* gets wrapped key */
    CK_ULONG_PTR pulWrappedKeyLen /* gets wrapped key size */
)
```

`C_UnwrapKey` example:

```
CK_DEFINE_FUNCTION(CK_RV, C_UnwrapKey)
  (
    CK_SESSION_HANDLE hSession, /* session's handle */
    CK_MECHANISM_PTR pMechanism, /* unwrapping mech. */
    CK_OBJECT_HANDLE hUnwrappingKey, /* unwrapping key */
    CK_BYTE_PTR pWrappedKey, /* the wrapped key */
    CK_ULONG ulWrappedKeyLen, /* wrapped key len */
    CK_ATTRIBUTE_PTR pTemplate, /* new key template */
    CK_ULONG ulAttributeCount, /* template length */
    CK_OBJECT_HANDLE_PTR phKey /* Gets new handle */
)
```

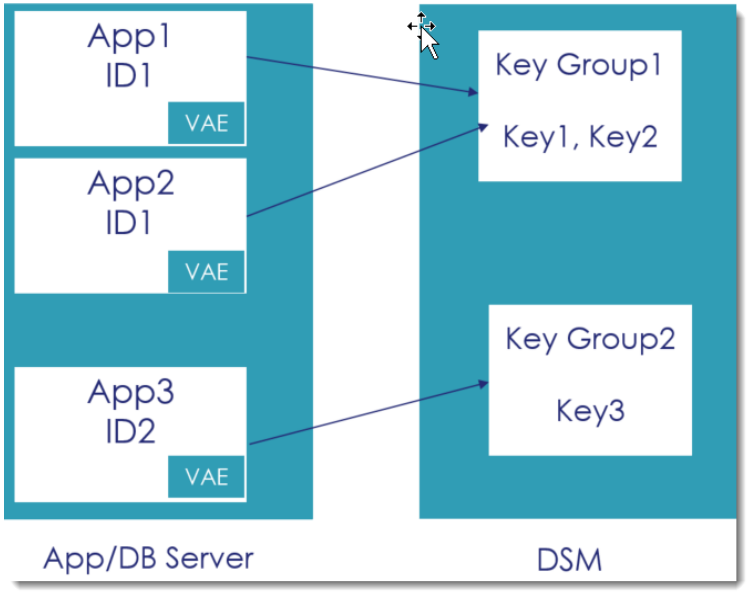
Identity-Based Key Access

Before using the Identities feature on the VAE, the DSM admin must setup keys and key groups on the domain.

NOTE: For more information about the Identities feature see: "Providing Identity-Based Key Access Credentials" on page 35 and the *Data Security Manager (DSM) Installation and Configuration Guide.*

The VAE provides credentials (username and password) to support Identity-Based Key Access. Permission based domain access is regulated by Key Groups. Keys are assigned to key groups and can be included in multiple key groups to establish exclusive access to the keys. The association between identities and key groups is many to many.

Figure 5: Identity Domain Mapping Example



NOTE: For backward compatibility legacy `C_Login` calls still work. In this case the Identity User and Identity Password are not passed to the DSM.

To create a key automatically in a key group, use the following command::

```
CK_DEFINE_FUNCTION(CK_RV, C_Login)
  (
    CK_SESSION_HANDLE hSession, /* session's handle */
CK_USER_TYPE userType, /* the user’s type*/
CK_CHAR_PTR pPin, /* the user’s PIN*/
    CK_ULONG ulPinLen, /* the length of the PIN*/
)
```

The pPin takes the <registered pin>:<identity name>:identity password>:<default key group>, with <default key group> being optional.

PKCS#11

Encryption and Decryption in the VAE follow the PKCS (Public Key Cryptography Standards) interface, developed by RSA Laboratories. PKCS is actually a group of 15 separate standards, in which #11 is an API called *Cryptoki* that defines a generic interface to cryptographic tokens.

PINs

An important part of the PKCS standard is that a PIN be used to log in, enabling the user to access the API functions for encryption and key management. When a VAE and DSM are used together, this PIN is created during machine registration.

The PIN is used for encrypting the private certificate file that establishes a secure channel between the VAE and the DSM. The PIN is hashed for verification of login. It is specific to only the server on which it was created, which makes it necessary to re-register if the VAE or the DSM is migrated.

Header Files

The header files used by the Cryptoki API contain a comprehensive set of object and function definitions.

Object types include:

- Data objects, including custom data types
- Storage objects
- Hardware Objects

- Key Objects
- Certificate Objects
- Domain Parameter Objects
- Mechanism Objects

Types of defined functions include:

- Slot and Token management functions
- Session management functions
- Message Digesting functions
- Encryption and Decryption functions
- Key Management functions
- Signing functions
- Callback functions

Vormetric Application Encryption Installation

3

This chapter describes how to install and configure Vormetric Application Encryption (VAE) on Windows and Linux systems. It contains the following sections:

Overview

Installing the key agent installs the shared library that enables application encryption. The library (`libvorpkcs11.so` on Linux or `vorpkcs11.dll` on Windows) provides functions that the customer can call to do application encryption and key management.

Assumptions

The IP address, routing configuration, and DNS addresses for the hosts where Vormetric Encryption Agents are installed allow connectivity to all DSMs.

NOTE: Since the key cache on the VAE agent is only in memory and not written to disk, the VAE (key) agent must connect to the DSM upon a reboot to retrieve necessary keys. The challenge-response/generate password option (in which the user can type in a password to unlock the agent when the DSM is off-line) is available only for VTE agents and is NOT supported with the VAE (key) agent.

Because we don't support the host password for challenge response/generate password or a manual pass phrase with the VAE (key) agent, there is no work-around for an offline DSM situation. For continuous access, make sure you have at least 1 healthy DSM online in your HA DSM configuration to serve up the MEK to decrypt table/tablespace keys.

Installation Plan

Following are the high-level steps for installing and configuring VAE.

1. If fingerprint verification is used, confirm with the DSM Security Administrator that the host where you will install the agent has been added to the DSM.
2. Collect configuration information.
3. Set up the host name resolution method.
4. Open firewall ports, if applicable.
5. Choose the installation method (Silent or Interactive).
6. Decide if you want to enable the anti-cloning functionality.
7. Install the agent software.
8. Verify the installation.

Agent Install Checklist

Use this table to verify prerequisites and collect the information needed for the installation.

Table 3: VAE Agent Installation Checklist

Checklist item	Status
Obtain the Installation file from Vormetric support. The format for the VAE Agent file name is: `vee-key-<product-version-build-system>.exe` **Examples:** Windows: `vee-key-6.0.2-win64.exe` Linux: `vee-key-6.0.2-rh7-x86_64.bin`	
Fully Qualified Domain Name (FQDN) of the DSM	

Table 3: VAE Agent Installation Checklist (Continued)

Checklist item	Status
IP address or Fully Qualified Domain Name (FQDN) of the host	
Subnet mask of the host	
Root password for the host	
If using Shared Secret Registration, get from the **DSM Security Administrator**: 1) The shared secret password 2) Domain 3) Host group if applicable 4) A description for the host.	
If using the Fingerprint Registration ask the **DSM Administrator** to add the host to the DSM database and check the **Registration Allowed** check box. After checking the fingerprint, select the **Communication Enabled** check box.	
Host system clock set to the correct time zone	

Before You Begin

- The Vormetric Data Security Manager (DSM) must be installed and configured before the VAE Agent is installed.
- Check the Release Notes to confirm hardware and operating system requirements and software compatibility requirements for the DSM before you begin.
- Do not install the VAE Agent on network-mounted volumes like NFS.

Determine Your Agent Registration Method

Protected hosts can be registered with the DSM using either the *Fingerprint method* or the default *Shared Secret method*.

- **The Fingerprint method**—requires the DSM Security Administrator to add the FQDN or IP address of each protected host to the DSM database before registering the agent.

  After registration, the installer of the agent passes the CA certificate fingerprint to the DSM Security Administrator to verify that the protected host and DSM share valid certificates.

  If you choose the Fingerprint method, ask the DSM Security Manager to add the FQDN or IP address of the protected host to the DSM database before registering that host.

- **The Shared Secret method**—requires the DSM Security Administrator to create a *shared secret*—a case-sensitive string of characters—for auto-registering a domain or host group.

  Agent Installers use the shared secret to add and register protected hosts to the DSM for a domain or host group. The DSM Security Administrators can skip adding host names or IP addresses to the DSM database, because this is done automatically using the Shared Secret Registration method, and there is no need to verify that the protected host and DSM share valid certificates. Multiple protected hosts can be added dynamically with a single shared secret password during the agent installation and registration process.

If you choose the Shared Secret method, ask the DSM Security Manager to create a shared secret for the domain or host group in which the new protected host will reside. Then, get the shared secret and the validity period (one hour, day, week, or month) and register within that period.

There is a "**Require that hosts are first added**" checkbox in DSM Shared Secret creation page. If this box is checked, the hosts must be manually added to the DSM database.

Host Name Resolution

You can map a host name to an IP address using the Domain Name System (DNS). DNS is the most preferred method of host name resolution.

You can also modify the *hosts* file on the DSM or identify a host using only the IP address.

- If you use DNS to resolve host names, use the FQDN for the host names.
- If you do NOT use a DNS server to resolve host names, do the following on all of the DSMs and the protected hosts:
  - **Modify the *hosts* file on the DSM:** To use names like *serverx.domain.com,* enter the host names and matching IP addresses in the `/etc/hosts` file on the DSM using the `host` command under the *network* menu.

    For example:

    Request:

```
network$ host add jblaster-dev1 10.3.42.12
SUCCESS: add host
```

    Response:

```
network$ host sh
name=localhost6.localdomain6 ip=::1
name=linux32-48215.sacdbackup.com ip=10.3.48.215
name=jblaster-dev1 ip=10.3.42.12
SUCCESS: show host
```

IMPORTANT: You must do this on *each* DSM, since entries in the **hosts** file are not replicated across DSMs.

- **Modify the *hosts* file on the protected hosts:** Enter the DSMhost names and matching IP addresses in the `/etc/hosts` file on the protected host. *You must do this on EACH protected host making sure to add an entry for all DSM nodes (if using HA).*

Using the CLI

NOTE: The Vormetric Data Security Manager (DSM) is sometimes referred to in the code as a "Security `Server`" or a "`Data Security Server`" or sometimes as just the "Server".

Access the CLI menu as follows:

1. Start the serial console application (for example, HyperTerminal).
2. If the login prompt is not displayed, press the **Enter** key to wake up the connection.
3. Log on to the appliance. The default System Administrator name and password are `cliadmin` and `cliadmin123`.

Initial Setup

The following steps describe how to install the VAE Agent for the first time. For Supported OSs and DSM compatibility information, refer to the Release Notes.

To Install the VAE Agent on Windows

NOTE: The minimum length for the PIN is 8 characters and maximum length is 63 characters. The PIN cannot contain the $ symbol.

1. Log on to the host as a Windows user with administrative privileges.
2. Copy the installation file onto the Windows system.
3. Double-click the installation file. The *Welcome* window opens. Verify the version of VAE you are installing.
4. Click **Next**. The License Agreement appears.

5. Accept the License Agreement and then click **Next**. The *Destination Folder* window opens.
6. Click **Next** to accept the default folder. *The Ready to Install the Program* screen opens.

NOTE: If you have a Vormetric Transparent Encryption (VTE) (file system) agent already installed on the system, you cannot change the Destination Folder.

7. Click **Install**. The agent software installs. This may take a few minutes.
8. When the install is finished, the *Install Shield Wizard Completed* screen opens. **Register Vormetric Key Agent (64 bit) now** is selected.

NOTE: If you have a specific plan to register the agent later, clear the check box for **Register Vormetric Key Agent (64 bit) now**, click **Finish**, and then skip the rest of the steps in this section.

9. Click **Finish** to start the registration process. The *Register Host* window opens. Click **Next**. Verify the correct agent type is selected.
10. Click **Next**. Type the name of the DSM. This name must match the **Server name** on the **Dashboard** of the DSM.
11. Click **Next**. Enter the name of the machine hosting the agent. You can type the name or select it from the drop-down menu. This name must match the host name added to the DSM database by the Security Administrator.
12. Click **Next.** Enter a password. This is the PIN that will authenticate this device in the PKCS #11 applications you write.

IMPORTANT: Do not use the colon (:) character in the password sequence.

NOTE: If you are in a cluster environment, we recommend that you enter the same password for all cluster nodes.

13. Click the **Enable hardware association** box to enable the cloning prevention function.
14. Click **Register**. The *Register Host* window displays the VAE Agent version and a pop-up window displays the signer CA certificate fingerprint.

At this stage of the installation, the host administrator and DSM Security Administrator must exchange information to confirm that the agent host and DSM share valid certificates.

15. **Host Admin:** Send the fingerprint to the DSM Security Administrator and wait for confirmation.

16. **DSM Security Admin**:
17. a: Log onto the DSM Management Console and navigate to the domain where the host was added.
18. b: Click the **Dashboard** tab.
19. c: Match the fingerprint from the Host Admin with the appropriate **fingerprint** on the Dashboard.
20. d: Advise the Host Admin of the results.
21. **Host Admin:** If the fingerprints match, type 'Y' and then press **Enter**.
22. **Host Admin:** The fingerprint for the certificate is displayed. Pass this fingerprint to the DSM Security Admin.
23. **DSM Security Admin**:
    a. Click the **Host** tab.
    b. Click the name of the host where you just installed the agent. The **Edit host** screen opens.
    a. Match the fingerprint from the Host Admin with the fingerprint in the **Certificate Fingerprint** column.
    b. Advise the Host Admin of the results.
24. **Host Admin:** If the fingerprints match, click **OK**. A message that the installation was a success is displayed.
25. **DSM Security Admin**: On the DSM, select the **Communication Enabled** check box for the host.

NOTE: After installation, the PIN or password must be provided by the application developer when prompted during `C_Login`. ***Do not forget this password.*** To change the password, you must re-register the agent.

To Verify the Installation on Windows

Verify the installation by checking agent processes.

1. In the system tray, right-click the Vormetric icon.
2. Select Status. Review the information in the Vormetric Status window to confirm the correct agent or agents are installed and registered.

To Install the VAE Agent on Linux/UNIX

NOTE: Password minimum length for PIN is 8 characters and maximum length is 63 characters.

1. Log on to the host where you will install the VAE Agent. You must have root access.
2. Copy or mount the installation file to the host system.
3. Start the installation. At the prompt, type:

```
./vee-key-<product-version-build-system>.bin
```

Example:

```
./vee-key-6.1.0-rh7-x86_64.bin
```

4. Type '**y**' and press **Enter** to accept the Vormetric License Agreement.

   The installation proceeds.

NOTE: The Vormetric Data Security Manager (DSM) is sometimes referred to in the code as a "`Security Server`" or a "`Data Security Server`" or sometimes as just the "`Server`".

```
Welcome to the Vormetric Key Agent
Registration Program.
Agent Type: Vormetric Key Agent
Agent Version: [latest version]
In order to register the Vormetric Key Agent
with a Vormetric Data Security Server:
    1) you must know the host name of the machine running the
       Security Server (the host name is displayed on the
       Dashboard window of the Management Console), and
    2) unless you intend to use the 'shared secret' registration
method,
       the agent's host machine must be pre-configured on the
       Security Server as a host with the 'Reg. Allowed'
       checkbox enabled for this agent type on the Hosts window
       of the Management Console
```

```
Would you like to register to the Security Server using a
registration shared secret (S) or using fingerprints (F)? (S/F)
[S]:
What is the registration shared secret?
Please enter the domain name for this host: key_encrypt_domain
Please enter the host group name for this host, if any:
Please enter a description for this host: example
Shared secret    : ********
Domain name      : key_encrypt_domain
Host Group       : (none)
Host description : example
Are the above values correct?  (Y/N) [Y]:
```

5. At the following prompt, type '**y**' or press **Enter** to proceed with registration, or type '**n**' if you specifically plan to register later.

```
Do you want to continue with agent registration? (Y/N) [Y]: Y
```

The following dialog displays:

```
Please enter the primary Security Server host name: DSM Appliance-dg-
15208.com
You entered the host name dsm-dg-15208.com
Is this host name correct? (Y/N) [Y]: Y
Please enter the host name of this machine, or select from the
following list. If using the “fingerprint” registration method, the
name you provide must precisely match the name used on the "Add Host"
page of the Management Console.
[1] h55119.dg.com
[2] 10.3.55.119
Enter a number, or type a different host name or IP address in
manually:
What is the name of this machine? [1]:
You selected "h55119.dg.com".
```

6. Follow the prompts:
   a. Enter the fully qualified host name of the primary DSM, and then press **Enter.**
   b. Verify the host name, and then press **Enter**. A list of host names appears.
   c. From the list, type the number of the host name of the local machine, or manually enter the host name, and then press **Enter**. This name must match the name of the host that was added to the DSM by the Security Administrator.

Example:

```
Please enter the host name of this machine, or select from the
following list. The name you provide must precisely match the name
used on the “Add Host” page of the Management Console.
[1] host1.example.com
[2] sys41017-priv.example.com
[3] sys41017-vip.example.com
[4] 10.3.41.127
Enter a number, or type a different host name or IP address in
manually:
What is the name of this machine? [1]: 1
Generating certificate...done.
Signing certificate...done.
```

7. At the following prompt, press **Enter** to enable cloning prevention functionality.

Example:

```
It is possible to associate this installation with the hardware of
this machine. If selected, the agent will not contact the DSM or
use any cryptographic keys if any of this machine's hardware is
changed. This can be rectified by running this registration program
again.
Do you want to enable this functionality?  (Y/N) [Y]:
```

8. The CA certificate fingerprint is displayed.

   At this stage of the installation, the host administrator and DSM Security Administrator must exchange information to confirm that the agent host and DSM share valid certificates. This is done to verify that nobody is intercepting and modifying traffic between the DSM and agent. This is a security feature.

9. Enter the password for the VAE (Key Agent) library and confirm your entry.

```
Please enter a password for the Key Agent library.
Accesses to this library will be protected by this password.
NOTE: If using Oracle RAC, passwords must be the same on all nodes.
Please enter password  :
Enter again to confirm :
Successfully registered the Vormetric Key Agent with the primary
Vormetric Data Security Server on dsm-dg-15208.com.
```

10. **Host Admin:** Send the fingerprint to the DSM Security Administrator and wait for confirmation.

11. **DSM Security Admin**:

a. Log on to the DSM Management Console and navigate to the domain where the host was added.

b. Click the **Dashboard** tab.

c. Match the fingerprint from the Host Admin with the **EC CA fingerprint** on the Dashboard.

d. Advise the Host Admin of the results.

12. **Host Admin:** If the fingerprints match, type 'y' and then press **Enter**. "Installation success." is displayed.

NOTE: After installation, the PIN or password must be provided by the application developer when prompted during `C_Login`. ***Do not forget this password.*** To change the password, you must re-register the agent.

Verify the Linux Installation

The best way to check if everything is correctly installed:

1. Register the agent against a DSM.
2. Run sample code such as `create_key`.
3. Check that the key is on the DSM by using the DSM GUI after running the `create_key` sample.

Modify the Key Cache

You can modify two settings in the key cache on the DSM.

By default, keys are cached on the host, with a time-to-live value of 44640 minutes (31 days).

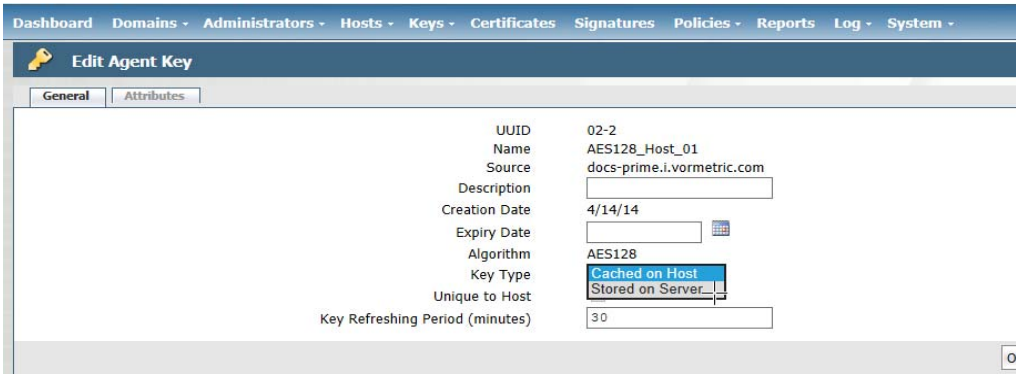
- You can choose to store the key on the DSM or cache the key on the host.
- You can also modify how long the key stays in the local key cache before it is re-fetched from the DSM.

NOTE: There is also the option of "Storing Keys on the Server by Default" by editing the agent.conf file, which takes precedence over a DSM setting.

To Modify the Key Cache on the DSM

1. Log on to the DSM as an administrator of type Security Administrator.
2. Click the **Domains** tab, and switch to the domain where the key is installed.
3. Click **Keys > Agent Keys**, and then click on the name of the key you want to modify. If you have many keys, search for a key on the Management Console using the **Keyname contains:** field. The *Edit Agent Key* window opens.

Figure 6: Edit Agent Key window



4. In the **Key Type** drop down, select **Cached on Host** or **Stored on Server** (DSM).
5. In the **Key Refreshing Period (minutes)** field, enter the number of minutes you want the key to exist in the local key cache before it is re-fetched from the DSM. Then click **OK**.

Certificate Renewal

The certificate that is registered between the VAE and the DSM expires in one year. However, renewal of the certificate happens automatically when the library in either makes a `C_Initialize()` call or a re-initialization takes place. The library checks for the validity of the agent certificate, and if it is found to expire within eight (8) days, a certificate signing request (CSR) is sent to the DSM to obtain a new certificate.

The directory in which the agent certificate is stored is owned by root, so there could be an issue when an application linking against the pkcs11 VAE library is running under another user. In this case, it is necessary to run a cron job (on windows) to renew the certificate. The application can be as simple as calling `C_Initialize()` and `C_Finalize()` only. There is a sample program named

vpkcs11_sample_renew_cert within the bundle of sample programs. The crontab entry on Linux should look like the following:

```
0 12 * * 1 <path>/vpkcs11_sample_renew_cert
```

And on Windows running the **at** command:

```
At 10:00:00AM /every:Monday c:\<path>\vpkcs11_sample_renew_cert
```

Uninstalling

To uninstall VAE on Linux, run the **uninstall** command:

```
/opt/vormetric/DataSecurityExpert/agent/pkcs11/bin/uninstall
```

To uninstall VAE on Windows:

1. Navigate to **Control Panel** > **Programs** > **Programs and Features**
2. Select the **Vormetric Key Agent**
3. Right-click to choose **uninstall**

4

Using the VAE API

This chapter is for application developers. It describes how to use the Vormetric Application Encryption (VAE) APIs to integrate VAE functionality with your applications. It contains the following sections:

Sample Code

VAE provides code examples for C and Java. The sample code demonstrates the following functionality:

- Create a symmetric key
- Search for and delete a key
- Encrypt and decrypt a single message
- Encrypt and decrypt a file
- Import a wrapped key into the DSM
- Export a wrapped key from the DSM
- Create an asymmetric key pair and sign
- Export and import asymmetric keys

Download and extract the most recent samples and documentation from:

https://support.vormetric.com

Place the files in the following directory:

```
<install directory>/vormetric/DataSecurityExpert/agent/pkcs11/
```

C sample files are located at:

```
.../samples/c_samples
```

Java sample files are located at:

```
.../samples/java_samples
```

NOTE: Refer to the README file in the samples directory for more information.

Compiling and Running Sample Code in c_samples

To compile and run the sample code provided in `c_samples`, use `gmake`. (Previously, the `make` command was used on Linux). Use the following commands.

To clean all the files:

```
gmake clean
```

To compile:

```
gmake
```

To run all samples:

```
gmake PASSWORD=<your password here> run
```

Location of Libraries, Samples, and Logs

The default location of the library files on Linux systems is:

```
/opt/vormetric/DataSecurityExpert/agent/pkcs11/lib
```

Linux code samples:

```
<install directory>/vormetric/DataSecurityExpert/agent/pkcs11/samples
```

On a Windows 32-bit machine, find the sample and library files as follows:

- Samples:

`C:\Program Files\Vormetric\DataSecurityExpert\agent\pkcs11\samples`

- Library installation directory:

`C:\Program Files\Vormetric\DataSecurityExpert\agent\pkcs11\bin`

On a Windows 64-bit machine, find the sample and library files as follows:

Table 4: Samples

Sample	Directory Path
32-bit samples	`C:\Program Files(x86)\Vormetric\DataSecurityExpert\agent\pkcs11\samples`
32-bit library installation directory	`C:\Program Files(x86)\Vormetric\DataSecurityExpert\agent\pkcs11\bin`
64-bit samples	`C:\Program Files\Vormetric\DataSecurityExpert\agent\pkcs11\samples`
64-bit library installation directory	`C:\Program Files\Vormetric\DataSecurityExpert\agent\pkcs11\bin`

The Windows logging file uses the following path:

```
%\ProgramData\Vormetric\DataSecurityExpert\agent\log
```

Using Java 9 and Higher with VAE

For environments using Java version 9 and later a custom Thales Java wrapper is available.

IMPORTANT: Make sure you are using VAE version 6.2.0 or newer.

1. Update the work environment CLASSPATH configuration to replace `sun.security.pkcs11.wrapper` with the new Java wrapper named `pkcs11-wrapper-6.2.0-<build-no>.jar`.
2. Replace each Sun wrapper import statement with the new Java wrapper import name, as shown in Table 5 - “Import Replacements”.
3. Compile and run the implementation code.

Table 5: Import Replacements

Legacy	Replacement
sun.security.pkcs11.wrapper.CK_ATTRIBUTE	com.vormetric.pkcs11.wrapper.CK_ATTRIBUTE
sun.security.pkcs11.wrapper.CK_MECHANISM	com.vormetric.pkcs11.wrapper.CK_MECHANISM
sun.security.pkcs11.wrapper.CK_SLOT_INFO	com.vormetric.pkcs11.wrapper.CK_SLOT_INFO
sun.security.pkcs11.wrapper.CK_VERSION	com.vormetric.pkcs11.wrapper.CK_VERSION
sun.security.pkcs11.wrapper.PKCS1	com.vormetric.pkcs11.wrapper.PKCS11
sun.security.pkcs11.wrapper.PKCS11Exception	com.vormetric.pkcs11.wrapper.PKCS11Exception
sun.security.pkcs11.wrapper.PKCS11Constants*	com.vormetric.pkcs11.wrapper.PKCS11Constants*

NOTE: See “Example With Thales Java Wrapper” on page 53 for a complete code sample.

Verifying Successful API Initialization

Each time the application linked to the VAE library starts up, the VAE library checks itself to be sure that the cryptographic functions are working correctly and the library integrity has not been compromised. This self-test runs when the cryptographic library is loaded, before any `pkcs11` library function calls.

To check whether the API passed this test, view the log files:

- Linux: `/var/log/vorpkcs11/selftest.log`
- Windows: `C:\ProgramData\Vormetric\DataSecurityExpert\agent\log\vorpkcs11_startup.log`

If the self-test is successful, the following messages appear in the log file for each startup of the application running the library:

Linux:

```
VAE Library (45771): Tue Sep 20 14:40:21 2016 [SUCCESS] selftest passed!
VAE Library (45771): Tue Sep 20 14:40:21 2016 [SUCCESS] integrity check
passed!
```

Windows:After this verification step, the application can access the cryptographic functions.

```
VAE Library: Tue Sep 20 14:40:21 2016 [SUCCESS] selftest passed!
VAE Library: Tue Sep 20 14:40:21 2016 [SUCCESS] integrity check passed!
```

If the verification fails, the VAE library logs a message and exits with status 255. The messages logged my either be related to an integrity failure or a Know Answer Test (KAT) failure.

Linux failure messages:

```
VAE Library (45771): Tue Sep 20 14:40:21 2016 [FAILURE] selftest failed!
VAE Library (45771): Tue Sep 20 14:40:21 2016 [TERMINATE] exiting due to
failure!
VAE Library (45771): Tue Sep 20 14:40:21 2016 [FAILURE] integrity check
failed!
VAE Library (45771): Tue Sep 20 14:40:21 2016 [TERMINATE] exiting due to
failure!
```

Windows error messages:

```
VAE Library: Tue Sep 20 14:40:21 2016 [FAILURE] selftest failed!
VAE Library: Tue Sep 20 14:40:21 2016 [TERMINATE] exiting due to failure!
VAE Library: Tue Sep 20 14:40:21 2016 [FAILURE] integrity check failed!
VAE Library: Tue Sep 20 14:40:21 2016 [TERMINATE] exiting due to failure!
```

Providing Identity-Based Key Access Credentials

The VAE API is used to provide credentials (username and password) to facilitate identity based access. The DSM validates the identity by verifying username and password hash.

Use the VAE "C_Login" API to pass in the username and password pair. The user is expected to pass the following parameters using a colon delimiter:

- `vaepin`

- `Identity User`
- `Identity Password`

If the `Identity User` and `Identity Password` are not provided. the VAE accepts the `vaepin` and domain access is not restricted by identity/key group.

IMPORTANT: the user parameter cannot include the following characters: “**:**”, “**!**”, “**@**”, “**#**”, “**$**”, “**%**”, “**^**”, “**&**”, “*****”, “**(**“ “**)**”.

NOTE: For more details see See also “C_Login” on page 73.

NOTE: See also “Identity-Based Key Access” on page 14.

Creating a Key in a Key Group

Use the VAE “C_Login” API create the Key Group with the following syntax:

```
Pin:Identity:password:group
```

Restricting Encryption Key Access

To restrict older VAE or VKM clients from accessing key on the DSM, the DSM admin must move all keys into a key group that requires identity enforcement. This solution is effective for all versions of the VAE or VKM before 6.1.0.

Metadata Logging and Sample Code

This section shows you how to use metadata logging with the Vormetric Application Encryption (VAE) library.

Metadata logging is used to pass extra information (such as user name, user login ID or any other user-specified information), associated with the user who makes a particular function call.

Two calls are required for a corresponding function to create metadata logging. In the first call, the input buffer contains the user metadata to be logged. The corresponding buffer length passed must be zero. The second call follows the regular PKCS #11 calling conventions by passing in the allocated buffer and its corresponding length.

C Sample

The following example is for `vpkcs11_sample_metadata_logging.c`:

```
encryptAndDecrypt ()
. . .
CK_BYTE* cipherText = NULL_PTR;
CK_ULONG cipherTextLen = 0;
CK_ULONG metaDataLen = 256;
CK_CHAR metadata[] = "META: This is a test metadata/Encryption: user:
tester: hostname" ;
CK_CHAR metadata2[] = "META: This is a test metadata/Decryption: user:
tester: hostname" ;

/* For C_Decrypt */
CK_BYTE* decryptedText = NULL_PTR;
CK_ULONG decryptedTextLen = 0;
/* General */
CK_RV rc = CKR_OK;
. . .
cipherText = (CK_BYTE *)calloc( 1, sizeof(CK_BYTE) * metaDataLen );
if(cipherText != NULL)
{
sprintf((char*)cipherText, "%s", (char *)metadata);
}
printf ("Plain Text length: %d\n", (int)sizeof( plainText));
/* first call C_Encrypt by pass in metadata and obtain cipherText buffer
size upon CKR_OK return */
rc = FunctionListFuncPtr->C_Encrypt(
hSession,
plainText, sizeof (plainText),
cipherText,&cipherTextLen
);
if (rc != CKR_OK){
printf ("C_Encrypt failed\n");
return rc;
```

```
}
else
{
printf ("C_Encrypt succeed, cipherTextLen is : %ld \n",
(long)cipherTextLen);
cipherText = (CK_BYTE *)calloc( 1, sizeof(CK_BYTE) * cipherTextLen );
}
. . .
```

Java Sample

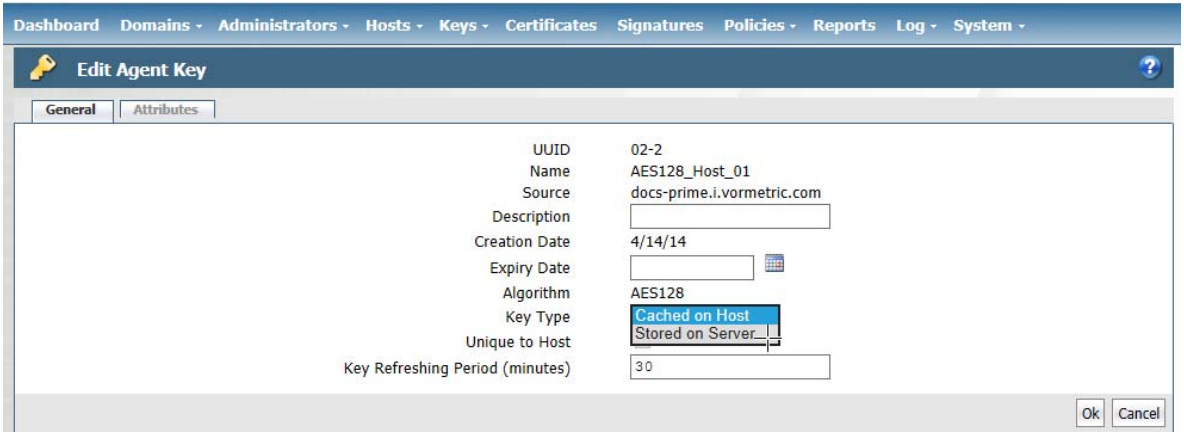
Metadata samples for Java can be found in the following files:

- `EncryptDecryptMetaData.java`
- `EncryptDecryptFileMeta.java`

Troubleshooting

By default, keys are cached on the host. If you don't want them cached on the host, log on to the DSM Console, switch to the domain with the desired host, click the **Keys** tab and select the key name from the **Name** list. In the **Key Type** drop down menu in the **Edit Agent Key** window, select **Stored on Server.**

Figure 7: Edit Agent Key window



5 Encryption Use Cases

This chapter describes the general work flows of several different encryption applications, followed by a walk-through of how the encryption and decryption cycle works with the VAE and DSM.

Signing

One important use for encryption, in addition to securing users' data, is in the creation of digital signatures. Just like handwritten signatures, digital signatures are used to indicate the authenticity of the source of a message, but digital signing is much more difficult to forge.

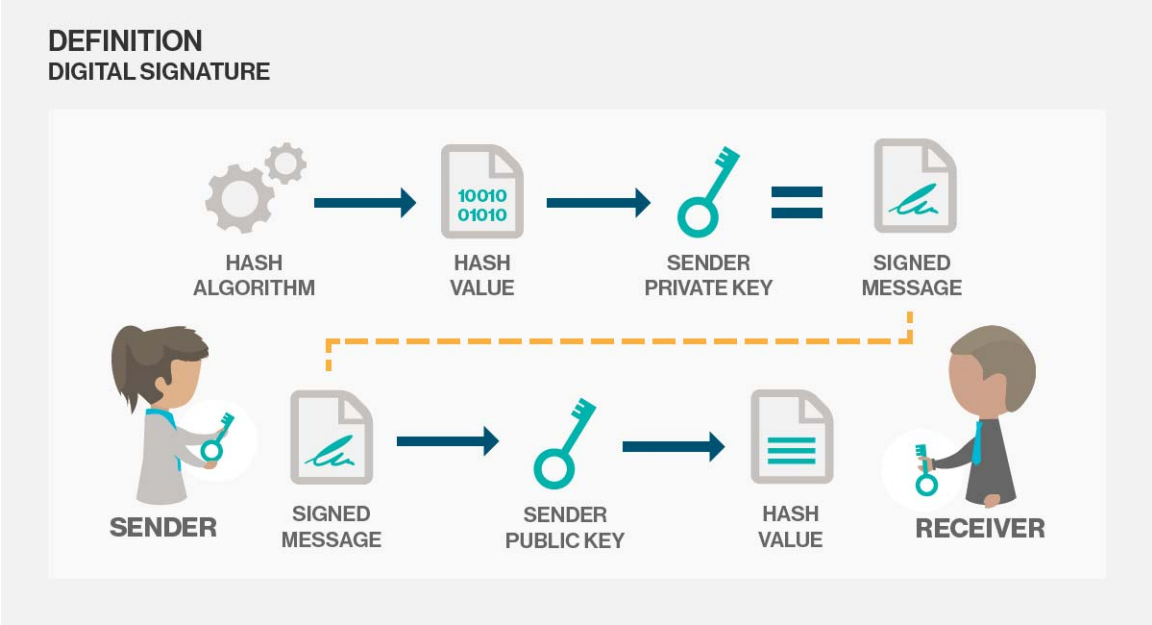
Creating a signature is an asymmetric encryption process that begins with two generated keys, one public and one that is private to the sender. The two keys are mathematically linked to allow verification by the receiver. The next step is to create a hash of at least part of the data to be sent. (For a more in-depth discussion of hashing, refer to the next section.) In practice, the data to be hashed is usually not the actual message; it's simply a large random number. The content of the message may or may not be encrypted separately.

In either case, the sender encrypts this hashed data with the private key, resulting in the digital signature. Because the value of the hash is always unique to the data, any change in the data results in a different hash value. This feature makes it nearly impossible to compromise either the sender's identity or the message itself.

The receiver can validate the message by decrypting the hashed data with the sender's public key. If the original hash algorithm is known, the receiver can simply compare the computed hash value with the encrypted hash value in the message. If they match, it proves that the message has not changed since it was signed. If not, it shows that the message was either tampered with or encrypted with a key that is not linked to the sender's public key.

Figure 8 illustrates a typical signature workflow.

Figure 8: Digital signature workflow



Hashing

Even though hashing is very different from encryption, it still plays an important role in many different encryption use cases, including many VAE applications. The fundamental difference between hashing and encryption is that encryption is designed to be reversible. Hashing is designed not to be.

As detailed in the previous section on Signing, hashing is very useful for generating a signature. However, there are several other common use cases. For example, one of the most common applications for hashing is to compare two values that, for security reasons, the user does not want to store internally. Passwords are the most obvious case, of course. If you have already stored a hashed copy of the user's password in your system's database, all that is required is to run any future password inputs through the same hash algorithm. You are not interested in what the password was: you are interested only in whether the input matches the one that was stored.

Most other uses for hashing also involve data integrity. The CRC32 algorithm, for example, is a hash that is often used for checking for whether compressed data has been corrupted. It's also possible to use hashing to check a file system for pirated data.

NOTE: It is very important for a hashing algorithm to avoid a situation where two different inputs return the same hash value: this phenomenon is called a collision. Well-designed hash algorithms minimize the probability of collisions, usually based on input size.

FPE

Format Preserving Encryption (FPE) is designed to create an encrypted output (ciphertext) that has the same length and character set as the input (called plaintext). The most common use of FPE is for financial data, such as credit card numbers, where there is legacy software that is unable to deal with encrypted data with different lengths or character sets.

Although a variety of FPE algorithms have been used, VAE relies on the FF1 and FF3 algorithms. For complete details of these, refer to *NIST Special Publication 800-38G, Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption*.

NOTE: A character set for FPE is entirely user defined. You could define a character set with as few as 2 characters ('Y' and 'N' or '0' and '1'), as large as 65535, or just 62 ('0'-'9' and 'A'-'Z' and 'a'-'z'). It is up to the user to decide how large the character set should be. However, for VAE, input characters not present in the character set result in an error.

A character set should be chosen carefully. If it is too small, the user gets an error if a character that is not part of the character set appears in the input. If the character set is too large, the input is basically a random sample of the supplied character set!

Caution: When using variable-length character sets, make sure to allocate enough memory for the worst-case ciphertext, which can be larger than the plaintext.

Storing Keys on the Server by Default

VAE-created keys are cached on the host or stored on the server. If nothing is explicitly defined, Vormetric caches the key on the host by default, for performance reasons.

In previous releases, you could decide to store individual keys on the server two ways:

- By API attribute: Use the `CKA_THALES_CACHED_ON_HOST` API attribute in a template and set it to 0, or
- From the DSM console: Click the **Keys** tab and select the key name from the **Name** list. From the **Edit Agent Keys** window, in the **Key Type** drop-down menu, select **Stored on Server.**

You also have the option to set key storage on the server as a global default.
To do so:

1. Access the `agent.conf` file.
   Linux:
   `/opt/vormetric/DataSecurityExpert/agent/pkcs11/etc/agent.conf`
   Windows:
   `C:\ProgramData\Vormetric\DataSecurityExpert\agent\pkcs11\etc\agent.con`
2. Set the flag `"Key_StoredOnServer" = 1`
3. Set the flag `"Key_Cache_Time"` = a value in minutes for the default cache time.

NOTE: Template values override the `agent.conf` values. Therefore, if you add the `Key_StoredOnServer` flag to the `agent.conf` file, it can be overridden for individual keys by setting `CKA_THALES_CACHED_ON_HOST` API attribute to 1, or through the DSM GUI.

Automated Key Versioning

VAE supports “Automated Key Versioning”. Encryption key versioning allows developers to simplify their applications by requesting automated key version advances, changing the key material automatically and placing the current version of the key in the header of the encrypted data. When the application requests decryption, VAE reads the header, retrieves the correct version of the key from the DSM, and then decrypts the data. This is transparent to the application, enabling the developer to refer to the key by its name/handle, not its version or corresponding material.

You can enable Automated Key Versioning when you create a key with `C_GenerateKey`. See Option 2, below.

NOTE: Automated Key Versioning does not pertain to VAE’s support for RSA DPM Encryption Header versions.)

The DSM administrator can also implement Automated Key Versioning in the DSM UI.

NOTE: The DSM user interface refers to “Automatic Key Rotation,” which VAE calls “Automated Key Versioning.”)

What is a versioned key

A versioned key is a standard cryptographic key that is versioned, with a version number and other metadata placed as a header in front of encrypted data. Versioned keys enable key material to changed (sometimes referred to as ‘rotation’) without changing the associated key metadata (such as key name, key size or algorithm). However, new key material and a new version identifier are created.

How do versioned keys work

A developer can use the APIs of the VAE library to accomplish the following functions:

Encryption: A header containing the key identifier is placed in front of the encrypted data: `C_Encrypt.init, C_EncryptUpdate, C_EncryptFinal`

Decryption: The header in front of the encrypted data is parsed and is used to detect the correct key to decrypt data: `C_Decrypt.init, C_DecryptUpdate, C_DecryptFinal`

Key creation: A standard cryptographic key is created with its own metadata, including version, name, size, algorithm, etc.
`C_GenerateKey`

Key deletion: Key must be in a pre-activated, deactivated, or compromised state before the key can be deleted using the key ID. Deleting the base key deletes all versions.
`C_DestroyObject`

Key Import/Export: Key can be imported into DSM with the VAE APIs and each version can be imported sequentially. Key can be retrieved by wrapping it with another symmetric key.
`C_CreateObject/C_WrapKey/C_UnwrapKey`

NOTE: To import an existing keypair, you must import the private key. The DSM regenerates the public key. If only the public key is imported, only the public key is on the DSM.

Automated key versioning: At requested time intervals, new cryptographic key materials and version ID are created. The original key name, algorithm, and key size remain unchanged.
`C_GenerateKey`

- Automated key versioning can be requested in the DM graphical user interface.
- Automated key versioning can be requested through Vormetric Application Encryption API calls.

Data rekeying: A variety of API calls can be used to enable data rekeying
`C_FindObjects.init, C_FindObjects, C_FindObjectsFinal,` as well as encryption/decryption APIs

- Detect the version (ID) of the key used for encryption
- Decrypt the data using that key
- Re-encrypt data with the latest version of the key
  Use Case: rekeying a column in the database:
  - Decrypting all data with the old key (previous versions) when read from the database, and
  - Encrypting it with the new key (latest version) when data is written/saved in the database.
  - These operations can be executed LIVE and without stopping the database or any system With Automated Key Versioning, when a new key version is created, data that has never been read and written will not be rekeyed or re-encrypted with the new key. However, it is possible to implement an active/live-data-rekeying mechanism by writing a “refresh utitlity” that reads (decrypts) data from the database in batches, and writes (encrypts)

them back to the databases. Batch Data Transformation, another product from Thales eSecurity, is another option for implementing this behavior.

Sample Scenario

- Key is generated; version 1 is created
- When data is encrypted, the data is tagged with a header containing version 1
- When data is decrypted, the application detects that version 1 was used for encryption, and the correct key is used to decrypt
- At the specified interval, automatic key versioning creates version 2 of the key material
- Now new data is encrypted with the new version 2 key
- When data is decrypted, the application will parse the correct version and decrypt the data. Note that the version may be 1 or 2.

Implementing Automated Key Versioning

There are two simple ways to enable automated key versioning in VAE 6.0. The minimum version lifespan is 1 day and there is no effective limit on the number of key versions supported in a versioning rotation schedule.

Option 1: In the DSM GUI

A DSM administrator can add new key(s), check the “Automatic key rotation” checkbox, and set the version lifespan in the DSM UI.

NOTE: The VAE term “automated key versioning” and the DSM term “automatic key rotation” refer to the same thing.

1. In the DSM UI, select Keys > Agent Keys > Keys and click **Add.**
   The *Add Agent Key* window is displayed.
2. Enter the following values:
   Name: create a name
   Template: Default_SQL_Symmetric_Key_Template (required)
   Algorithm: AES 128 or AES 256
   Key Type: Cached on Host (recommended)
   Key Creation Method: Generate
   Key Refresh Period: (refers to Cached on Host setting—not important for key rotation)
   Automatic Key Rotation: checked
   Key Version Lifespan (days): any positive number between 1 – 10,000 (inclusive)

3. Click OK.

NOTE: Once the key is created with automatic key rotation checked and the lifespan set, the key cannot be reverted to non-auto-rotation status.

Option 2: In the application code

Alternatively, a developer can copy sample code into their application and set the attribute values there. To do so:

1. Access the following two files: `vpkcs11_sample_create_key.c` and `vpkcs11_sample_helper.c`
2. Within `vpkcs11_ sample_helper.c`, find the function `CK_RV createKey(char *keyLabel, char *keyAlias, int gen_action, CK_ULONG ulifespan, int key_size)`
3. Within the `aesKeyTemplate` section of that helper file, find the attribute types:

```
{CKA_THALES_KEY_VERSION_ACTION, &keyVersionAction,
sizeof(keyVersionAction) } and
{CKA_THALES_KEY_VERSION_LIFE_SPAN, &ulifespan,
sizeof(ulifespan) }
```

NOTE:
`CKA_THALES_KEY_VERSION_ACTION` and `CKA_THALES_KEY_VERSION_LIFE_SPAN` are defined in `vpkcs11_sample_helper.h`.

Actual Values:

```
CKA_THALES_KEY_VERSION_ACTION                0x40000082
CKA_THALES_KEY_VERSION_LIFE_SPAN             0x40000083
```

4. Use editor to copy/paste/edit the two attribute types into your application, as part of generating the key (function `C_GenerateKey`).
5. Set the attribute types as follows:
   - `keyVersionAction` = **0** [meaning base key creation]
   - `ulifespan =` [**any positive number** between 1-10,000 (measured in days) for rotation schedule.]

NOTE: To manually rotate versioned keys, set the `keyVersionAction= 1` and remove the `CKA_THALES_KEY_VERSION_LIFE_SPAN` attribute from the template.)

NOTE: Once the key is created with the `ulifespan` attribute type set, the key cannot be reverted to non-auto-rotation status.

NOTE: For more information see the DSM *Installation and Configuration Guide.*

Complete Walk-Through

This section examines sample code driving the process used in the VAE to create a key and use it to encrypt data. These samples are in Java, but the VAE library also supports code in C.

The most important parts of this code are the calls to the PKCS library, beginning with `'C_'`.

Create and Import a Key

Key creation starts with creating the PKCS11 instance, after which the following steps are programically executed:

1. A PCKS11 session is created.
2. The new session is activated
3. A key object is created with the required attributes, and a file containing this object is created on the DSM.

Key Creation Java Sample

Step 1 - Create the Instance and Session

In the Java sample, creating the instance and session in these files is done with the `startUp` method, as follows:

```
public static Vpkcs11Session startUp(String libPath, String pin)
    {
        try
        {
            Vpkcs11Session session = new Vpkcs11Session();
          /* Initialization of the PKCS11 instance, open session
and login */
            session.p11 = PKCS11.getInstance(libPath,
"C_GetFunctionList", null, false);
            long[] slots = session.p11.C_GetSlotList (false);
            session.sessionHandle = session.p11.C_OpenSession
(slots[0], 0,  null, null);
            System.out.println ("Session successfully opened.
Handle: " + session.sessionHandle);
          session.p11.C_Login (session.sessionHandle, CKU_USER,
pin.toCharArray());
            System.out.println ("Successfully Logged in");
        return session;
      }
      catch (PKCS11Exception e)
      {
        System.out.println (e.getMessage());
        e.printStackTrace();
      }
      catch (Exception e)
      {
        System.out.println ("Exception thrown.");
        System.out.println (e.getMessage());
      }
      return null;
    }
```

Step 2 - Create the Key

Creating the key is done with the `createKey` method:

```
public static long createKey(Vpkcs11Session session, String keyName)
    {
        long keyID = 0;
        /* Create an AES 256 key on the DSM without pass in key value */
        try {
            CK_MECHANISM mechanism = new CK_MECHANISM (CKM_AES_KEY_GEN);

            CK_ATTRIBUTE[] attrs = new CK_ATTRIBUTE[]
                    {
                            new CK_ATTRIBUTE (CKA_LABEL, keyName),
                            new CK_ATTRIBUTE (CKA_CLASS, CKO_SECRET_KEY),
                            new CK_ATTRIBUTE (CKA_KEY_TYPE, CKK_AES),
                            new CK_ATTRIBUTE (CKA_VALUE_LEN, 32),
                            new CK_ATTRIBUTE (CKA_TOKEN, true),
                            new CK_ATTRIBUTE (CKA_ENCRYPT, true),
                            new CK_ATTRIBUTE (CKA_DECRYPT, true),
                            new CK_ATTRIBUTE (CKA_SIGN, false),
                            new CK_ATTRIBUTE (CKA_VERIFY, false),
                            new CK_ATTRIBUTE (CKA_WRAP, true),
                            new CK_ATTRIBUTE (CKA_UNWRAP, true),
                            new CK_ATTRIBUTE (CKA_EXTRACTABLE, false),
                           new CK_ATTRIBUTE (CKA_ALWAYS_SENSITIVE, false),
                           new CK_ATTRIBUTE (CKA_NEVER_EXTRACTABLE, true),
                            new CK_ATTRIBUTE (CKA_SENSITIVE, true),
 };

            System.out.println ("Before generating Key. Key Handle: " +
keyID);
            keyID = session.p11.C_GenerateKey (session.sessionHandle,
mechanism, attrs);
           System.out.println ("Key successfully Generated. Key Handle: "
+ keyID);
        }
        catch (PKCS11Exception e)
        {
            e.printStackTrace();
        }
        return keyID;
    }
```

This code bundle also has a `createKeyObject` method, which is similar to `createKey` but it also passes CKA_VALUE as one of the key attributes, and uses `C_GenerateObject` instead of `C_GenerateKey` to obtain the key handle.

Finally, the code logs out of the session, closes it, and destroys the PKCS11 instance.

Troubleshooting

If a CKR_DATA_LEN_RANGE error is thrown, do the following:

1. Append these two lines in the agent.conf file:

```
logger_threshold_P11 = TRACE
appender_threshold_File_Appender = TRACE
```

2. Rre-run the test program.
3. Navigate to `/var/log/vormetric/pkcs11_USERNAME.log` (where USERNAME is the login name, for instance "root" or "joe"), to view the log file.

Encrypt and Decrypt

Once the PKCS11 session is active and the key is created, the actual encryption is straightforward to code.

The following method takes a predefined text string, encrypts the string, prints the byte length of the result, and then decrypts the string to create a result that matches the original text.

NOTE: `C_EncryptInit` and `C_DecryptInit` must be run before the actual encryption and decryption calls.

```
public static void main ( String[] args)
    {
        String pin = null;
        String libPath = null;

        switch (args.length)
        {
            case 2:
            {
                if ( !args[0].equals("-p")) usage();
                pin = args[1];
                break;
            }
            case 4:
            {
                if ( !args[0].equals("-p")) usage();
                pin = args[1];
                if ( !args[2].equals("-m")) usage();
                libPath = args[3];
                break;
            }
            default:
                usage();
                break;
        }

try
{
            Vpkcs11Session session =
Vpkcs11_sample_helper.startUp(Vpkcs11_sample_helper.getPKCS11LibPath(libPath),
pin);

            long keyID = Vpkcs11_sample_helper.findKey(session, keyName) ;

            if (keyID == 0)
            {
                System.out.println ("the key is not found, creating it..." );
                keyID = Vpkcs11_sample_helper.createKey(session, keyName);
                System.out.println ("Key successfully Created. Key Handle: " +
keyID);
            }
```

```
/* encrypt, decrypt with key */
byte[] outText = new byte[120];
byte[] decryptedText = new byte [120];
int encryptedDataLen = 0;

session.p11.C_EncryptInit (session.sessionHandle, encMech, keyID);
System.out.println ("C_EncryptInit succeed");

System.out.println ("plaintext = " + plainText);
encryptedDataLen = session.p11.C_Encrypt (session.sessionHandle,
plainText.getBytes(), 0, plainText.length(), outText, 0, outText.length);
System.out.println ("C_Encrypt success. encrypted data len = " + encryptedDataLen);

session.p11.C_DecryptInit (session.sessionHandle, encMech, keyID);
System.out.println ("C_DecryptInit success");
session.p11.C_Decrypt (session.sessionHandle, outText, 0, encryptedDataLen,
decryptedText, 0, decryptedText.length);

String decryptedTextStr = new String (decryptedText);
System.out.println ("C_Decrypt succeed.");
System.out.println ("Plain Text = " + plainText + " Decrypted Text = " +
decryptedTextStr);

/* Delete the key */
session.p11.C_DestroyObject (session.sessionHandle, keyID);
System.out.println ("Successfully deleted key");

            Vpkcs11_sample_helper.closeDown(session);
    }
catch (PKCS11Exception e)
    {
System.out.println (e.getMessage());
e.printStackTrace();
    }
catch (Exception e)
    {
System.out.println ("Exception thrown.");
System.out.println (e.getMessage());
    }
    }
}
```

Example With Thales Java Wrapper

The following code sample shows the `GenerateKey` class with the Thales Java Wrapper implemented:

NOTE: For more information see "Using Java 9 and Higher with VAE" on page 33.

```
public void testGenerateKey_AES128() {
                    String keyname = "QA_PKCS11_" + Utils.randomNumberGenerator(12);
                    System.out.println(" key name : " + keyname);
                    long sessionHandle = 0;
                    CK_ATTRIBUTE[] attrs = new CK_ATTRIBUTE[] {
                        new CK_ATTRIBUTE(CKA_LABEL, keyname),
                            new CK_ATTRIBUTE(CKA_CLASS, CKO_SECRET_KEY),
                            new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_AES),
                            new CK_ATTRIBUTE(CKA_VALUE_LEN, 16 L),
                            new CK_ATTRIBUTE(CKA_TOKEN, true),
                            new CK_ATTRIBUTE(CKA_ENCRYPT, true),
                            new CK_ATTRIBUTE(CKA_DECRYPT, true),
                            new CK_ATTRIBUTE(CKA_SIGN, false),
                            new CK_ATTRIBUTE(CKA_VERIFY, false),
                            new CK_ATTRIBUTE(CKA_WRAP, true),
                            new CK_ATTRIBUTE(CKA_UNWRAP, true),
                            new CK_ATTRIBUTE(CKA_EXTRACTABLE, false),
                            new CK_ATTRIBUTE(CKA_ALWAYS_SENSITIVE, false),
                            new CK_ATTRIBUTE(CKA_NEVER_EXTRACTABLE, true),
                            new CK_ATTRIBUTE(CKA_SENSITIVE, true)
                    };

                    char[] pin = password.toCharArray(); //Ssl12345#
                    try {
                        long[] slotList = p11.C_GetSlotList(false);
                        assertNotNull(slotList);
                        int size = slotList.length;
                        assertTrue(size > 0);
                        System.out.println("=== Islot ID = " + slotList[0]);
                        sessionHandle = p11.C_OpenSession(slotList[0], 0, null, null);
                        System.out.println("=== sessionHandle = " + sessionHandle);
                        p11.C_Login(sessionHandle, CKU_USER, pin);
                        System.out.println("==== log in pass ====");
                        CK_MECHANISM mechamism = new CK_MECHANISM(CKM_AES_KEY_GEN);
                        long keyHandle = p11.C_GenerateKey(sessionHandle, mechamism, attrs);
                        System.out.println("key handle is: " + keyHandle);
```

```
                } catch (PKCS11Exception e) {
                System.out.println("=== PKCS11Exception is thrown ===");
                String errMsg = e.getMessage();
                System.out.println("=== error message = " + errMsg);

            } finally {
                try {
                    p11.C_Logout(sessionHandle);
                    System.out.println("=== Successfully log out ===");
                    p11.C_CloseSession(sessionHandle);
                    System.out.println("=== Successfully close session ===");
                } catch (PKCS11Exception e) {

                }
            }
        }

        }
```

Running Samples with the Java Wrapper

Use the following steps to run the sample code with the Java Wrapper:

1. Remove the previous /build/ folder, otherwise the recompile attempts were not actually creating new output files.
2. Use the following command for ant compiler:

```
ant "C:\Program
Files\Vormetric\DataSecurityExpert\agent\pkcs11\java_samples\pkc
s11
-wrapper-6.2.0-27.jar" compile
```

This command renders the following output:

```
Buildfile: c:\Program
Files\Vormetric\DataSecurityExpert\agent\pkcs11\java_sampl
es\build.xml
compile:
[mkdir] Created dir: c:\Program
Files\Vormetric\DataSecurityExpert\agent\pkc
s11\java_samples\build\classes
[javac] c:\Program
Files\Vormetric\DataSecurityExpert\agent\pkcs11\java_samp
les\build.xml:51: warning: 'includeantruntime' was not set,
defaulting to build.
sysclasspath=last; set to false for repeatable builds
[javac] Compiling 16 source files to c:\Program
Files\Vormetric\DataSecurity
Expert\agent\pkcs11\java_samples\build\classes
[javac] Note: Some input files use unchecked or unsafe operations.
[javac] Note: Recompile with -Xlint:unchecked for details.
BUILD SUCCESSFUL
Total time: 5 seconds
```

3. Manually then copy the jar file to the /build/jar folder.
4. Create a key from sample code:

```
c:\Program
Files\Vormetric\DataSecurityExpert\agent\pkcs11\java_samples>
java -cp build/classes/;build/jar/pkcs11-wrapper-6.2.0-27.jar
com.vormetric.pkcs11.sample.CreateKey -p Vormetric456! -m
"C:\Program
Files\Vormetric\DataSecurityExpert\agent\pkcs11\bin\vorpkcs11.dl
l" -k "YourKeyLabel"
```

Use “;” instead of a “:” between items in the class path on Windows.

Add the “-m” string with the path to the vorpkcs11.dll file.

This produces the following output:

```
Start CreateKey ...
Loading the Vormetric PKCS11 library from : C:\Program
Files\Vormetric\DataSecur
ityExpert\agent\pkcs11\bin\vorpkcs11.dll
WARNING: sun.reflect.Reflection.getCallerClass is not supported.
This will impact performance.
Session successfully opened. Handle: 1
Successfully Logged in
The key not found, creating it...
Current End Date: year: 2018 month: 12 day: 21
Before generating Key. Key Handle: 0
Successfully Generated key. Key Label: YourKeyLabel. Key Handle:
536871014
Key successfully Generated. Key Handle: 536871014
Successfully logged out.
Successfully closed session.
  End CreateKey.
```

API Reference

A

This API reference is organized by the following functional groups:

General Purpose Functions

C_Initialize

Initializes the pkcs11 library.

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

NOTE: Only the applications written in C will call C_Initialize. Java and .NET versions do not. For more examples, see the sample code.

NOTE: We do not support application level or OS level mutexes at this time.

Table 6: Output Parameters

Parameter	Description
CKR_ARGUMENTS_BAD	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_OK	The function is executed successfully.

C_Finalize

Called to indicate that an application is finished with using the `pkcs11` library.

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

Table 7: Input Parameters

Parameter	Description
`CK_VOID_PTR pReserved`	Must be NULL.

Table 8: Return Values

Parameter	Description
`CKR_ARGUMENTS_BAD`	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_OK`	The function is executed successfully.

C_GetInfo

Provides manufacturer and version information about the library.

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

Table 9: Input Parameters

Parameter	Description
`CK_INFO_PTR pInfo`	Pointer to a `CK_INFO` structure to receive the information.

Table 10: Output Parameters

Parameter	Description
C_GetInfo	Fills in the CK_INFO structure with relevant information about the library.
CK_INFO	Provides general information about Cryptoki. Definitions are outlined in Table 11

Table 11: Cryptoki Responses

Response	Description
cryptokiVersion	Cryptoki interface version number, for compatibility with future revisions of this interface.
manufacturerID	ID of the Cryptoki library manufacturer. Must be padded with the blank character (' '). Should ***not*** be null-terminated.
flags	Bit flags reserved for future versions. Must be zero for this version.
libraryDescription	Character-string description of the library. Must be padded with the blank character (' '). Should ***not*** be null-terminated.
libraryVersion	Cryptoki library version number.

Table 12: Return Values

Parameter	Description
CKR_ARGUMENTS_BAD	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_OK	The function is executed successfully.

C_GetFunctionList

Allows the application (caller of the Application Encryption Library) to find the list of APIs that are supported and their addresses.

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

NOTE: Only the applications written in C will call `C_GetFunctionList`. Java and .NET versions do not. For more examples, see the sample code.

Table 13: Input Parameters

Parameter	Description
`CK_FUNCTION_LIST_PTR_PTR ppFunctionList;`	Pointer to a value that will receive a pointer to `CK_FUNCTION_LIST` structure that contains function pointers for all the API routines in the library.

Table 14: Output Parameters

Parameter	Description
`ppFunctionList`	Filled in with the address of the list of function pointers from the PKCS#11 library.

Table 15: Return Values

Parameter	Description
`CKR_ARGUMENTS_BAD`	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	Insufficient memory to perform the requested function.
`CKR_OK`	The function is executed successfully.

Slot and Token Management Functions

C_GetSlotList

Provides list of available slots. It can be a single default slot. In C, call this function twice to get the actual slot list. The first time, it returns the number of available slots. Allocate memory to the available slots, and then the second call returns the actual slot list.

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

NOTE: Only the applications written in C will call `C_GetSlotList`. Java and .NET versions do not. For more examples, see the sample code.

First Call to Retrieve Number of Slots

First call to `C_GetSlotList(CK_FALSE)` to retrieve the number of slots.

Table 16: Input Parameters

	Description
`CK_BOOL tokenPresent`	Must be CK_FALSE.
`CK_SLOT_ID_PTR pSlotList`	
`CK_ULONG_PTR pulCount`	Pointer to an unsigned long to hold slot count.

Second Call to Retrieve Slot List

Second call to `C_GetSlotList` to retrieve the actual slot list.

Table 17: Input Parameters

Parameter	Description
`CK_BOOL tokenPresent`	CK_TRUE
`CK_SLOT_ID_PTR`	Pointer to the space allocated to hold the slot based on the count retrieved above.

Table 17: Input Parameters

Parameter	Description
`CK_ULONG_PTR pulCount`	Pointer to a CK_ULONG holding the number of slots allocated above.

Table 18: Output Parameters

	Description
`pulCount`	Points to the number of slots available. It is always 1 for our library.

Table 19: Return Values

	Description
`CKR_ARGUMENTS_BAD`	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.
`CKR_BUFFER_TOO_SMALL`	
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
`CKR_FUNCTION_FAILED`	
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	
`CKR_OK`	The function is executed successfully.

C_GetSlotInfo

Provides information about a particular slot in the system.

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

Table 20: Input Parameters

Parameter	Description
`CK_SLOT_ID slotID`	Must be 0.
`CK_SLOT_INFO_PTR pInfo`	Pointer to a CK_SLOT_INFO object.

Table 21: Output Parameters

Parameter	Description
`CK_SLOT_INFO_PTR`	The structure where `CK_SLOT_INFO_PTR` points will be filled in with relevant information about the token.

Table 22: Return Values

Parameter	Description
`CKR_ARGUMENTS_BAD`	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	Insufficient memory to perform the requested function.
`CKR_OK`	The function is executed successfully.
`CKR_SLOT_ID_INVALID`	The specified slot ID is not valid.

C_GetTokenInfo

Provides information about a specific token.

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

Table 23: Input Parameters

Parameter	Description
`CK_SLOT_ID slotID`	Must be slot 0.
`CK_TOKEN_INFO_PTR pInfo`	Pointer to a CK_TOKEN_INFO object.

Table 24: Output Parameters

Parameter	Description
`CK_TOKEN_INFO_PTR`	The structure where `CK_TOKEN_INFO_PTR` points will be filled in with relevant information about the token.

Table 25: Return Values

Parameter	Description
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	Insufficient memory to perform the requested function.
`CKR_OK`	The function is executed successfully.
`CKR_SLOT_ID_INVALID`	The specified slot ID is not valid.
`CKR_TOKEN_NOT_PRESENT`	The token was not present in its slot at the time that the function was invoked.
`CKR_TOKEN_NOT_RECOGNIZED`	The Cryptoki library and/or slot does not recognize the token in the slot.
`CKR_ARGUMENTS_BAD`	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.

C_GetMechanismList

Provides a list of mechanisms supported.

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

Table 26: Input Parameters

Parameter	Description
`CK_SLOT_ID slotID`	Must be slot 0.
`CK_MECHANISM_TYPE_PTR pMechanismList`	Pointer to memory to hold a list of `CK_MECHANISM_TYPE`.
`CK_ULONG_PTR pulCount`	Number of `CK_MECHANISM_TYPE` objects the memory above can hold.

Table 27: Output Parameters

Parameter	Description
`CK_MECHANISM`	The `CK_MECHANISM` structure where `CK_MECHANISM_TYPE_PTR` points will be filled in with the list of supported mechanism types.

Table 28: Return Values

Parameter	Description
`CKR_BUFFER_TOO_SMALL`	The output of the function is too large to fit in the supplied buffer.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.

Table 28: Return Values

Parameter	Description
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_OK	The function is executed successfully.
CKR_SLOT_ID_INVALID	The specified slot ID is not valid.
CKR_TOKEN_NOT_PRESENT	The token was not present in its slot at the time that the function was invoked.
CKR_TOKEN_NOT_RECOGNIZED	The Cryptoki library and/or slot does not recognize the token in the slot.
CKR_ARGUMENTS_BAD	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.

GetMechanismInfo

Provides information about a mechanism possibly supported by the token.

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

Table 29: Input Parameters

Parameter	Description
CK_SLOT_ID slotID	Must be 0.
CK_MECHANISM_TYPE type	Mechanism type to retrieve info for.
CK_MECHANISM_INFO_PTR pInfo	Pointer to a CK_MECHANISM_INFO structure.

Table 30: Output Parameters

Parameter	Description
CK_MECHANISM_INFO	The CK_MECHANISM_INFO structure where pInfo points is filled in with information about a particular mechanism.

Table 31: Return Values

Parameter	Description
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_MECHANISM_INVALID	An invalid mechanism was specified to the cryptographic operation.
CKR_OK	The function is executed successfully.
CKR_SLOT_ID_INVALID	The specified slot ID is not valid.
CKR_TOKEN_NOT_PRESENT	The token was not present in its slot at the time that the function was invoked.
CKR_TOKEN_NOT_RECOGNIZED	The Cryptoki library and/or slot does not recognize the token in the slot.
CKR_ARGUMENTS_BAD	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.

Session Management Functions

C_OpenSession

Starts a cryptographic session with a specific token.

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

Supported Functionality

- Only read/write sessions are supported. Maximum number of read/write sessions is 1000.
- No read-only sessions are supported.
- Each thread must run in its own session for multi-threaded applications.

Table 32: Input Parameters

Parameter	Description
`CK_SLOT_ID slotID`	Must be 0.
`CK_FLAGS flags`	Must be 0.
`CK_VOID_PTR  pApplication`	String to describe session name, or NULL.
`CK_NOTIFY Notify`	NULL
`CK_SESSION_HANDLE_PTR phSession`	Pointer to a session ID.

Table 33: Output Parameters

Parameter	Description
`CK_SESSION_HANDLE_PTR phSession`	The session ID pointer (`CK_SESSION_HANDLE_PTR phSession`) will be filled in with the session handle.

Table 34: Return Values

Parameter	Description
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_SLOT_ID_INVALID	The slot ID is out of range.
CKR_ARGUMENTS_BAD	Already initiated.

C_CloseSession

Closes a cryptographic session.

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

Table 35: Input Parameters

Parameter	Description
CK_SESSION_HANDLE hSession	Session to close, identified by the session handle.

Table 36: Return Values

Parameter	Description
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_ARGUMENTS_BAD	Already initiated.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.

Table 36: Return Values

Parameter	Description
CKR_OK	The function is executed successfully.
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.

C_CloseAllSessions

Closes all sessions on a slot. C programming language only.

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

Table 37: Input Parameters

Parameter	Description
CK_SLOT_ID slotID	Must be 0.

Table 38: Return Values

Parameter	Description
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_OK	The function is executed successfully.
CKR_ARGUMENTS_BAD	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.

C_GetSessionInfo

Provides information about a session.

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

Table 39: Input Parameters

Parameter	Description
CK_SESSION _HANDLE hSession	Session handle.
CK_SESSION_INFO_PTR pInfo	Pointer to a CK_SESSION_INFO object.

Table 40: Output Parameters

Parameter	Description
CK_SESSION_INFO_PTR	The structure where CK_SESSION_INFO_PTR points will be filled in with relevant information about the session.

Table 41: Return Values

Parameter	Description
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_OK	The function is executed successfully.
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.
CKR_ARGUMENTS_BAD	Already initiated.

C_Login

Passes user credentials to login into a token. The credentials can be PIN only for all domain access, include a user name and password to secure the session access based on key group permissions. Credentials are entered during Key Agent registration.

All domain access example:

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

Identity-based key access example:

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

Table 42: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session to log the user into.
`CK_USER_TYPE userType`	CKU_USER (We support only CKU_USER)
`CK_CHAR_PTR pPin`	Password used to register the key agent with the Data Security Manager. For identity based access the value must contain the following: • `vaepin` • `user` • `userpassword`

Table 42: Input Parameters

Parameter	Description
`CK_ULONG ulPinLen`	Length of the PIN.

Table 43: Return Values

Parameter	Description
`CKR_ARGUMENTS_BAD`	Already initiated.
`CKR_PIN_INVALID`	The specified PIN has invalid characters in it. This is the default error when a check for `loginforIdentity` fails.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
`CKR_FUNCTION_REJECTED`	This error is returned when the identity check fails.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	Insufficient memory to perform the requested function.
`CKR_OK`	The function is executed successfully.
`CKR_PIN_INCORRECT`	The specified PIN is incorrect.
`CKR_SESSION_CLOSED`	The session was closed during the execution of the function.
`CKR_SESSION_HANDLE_INVALID`	The specified session handle was invalid at the time that the function was invoked.
`CKR_USER_PIN_NOT_INITIALIZED`	Indicates that the normal user's PIN has not yet been initialized.
`CKR_USER_TYPE_INVALID`	An invalid value was specified.

C_Logout

Logs a user out from a token. You can call `C_logout()` multiple times in a row.

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

Table 44: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle that you want to log out of.

Table 45: Return Values

Parameter	Description
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	Insufficient memory to perform the requested function.
`CKR_OK`	The function is executed successfully.
`CKR_SESSION_CLOSED`	The session was closed during the execution of the function.
`CKR_SESSION_HANDLE_INVALID`	The specified session handle was invalid at the time that the function was invoked.

Object Management Functions

C_WrapKey

Supports key export on the DSM where a symmetric key is wrapped with another symmetric key on the DSM and then exported. This function supports metadata logging.

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

Table 46: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_MECHANISM_PTR pMechanism`	The mechanism pointer only supports CKM_AES_CBC_PAD mode. It requires a 16 byte IV. The structure looks like this: `{CKM_AES_CBC_PAD,iv,16};`
`CK_OBJECT_HANDLE hWrappingKey`	Symmetric key used to wrap the key.
`CK_OBJECT_HANDLE hKey`	Symmetric key to export.
`CK_BYTE_PTR pWrappedKey`	Points to an array to put the wrapped key bytes.
`CK_ULONG_PTR ulWrappedKeyLen`	Length of the above array.
`CKM_THALES_PEM_FORMAT`	Export an asymmetric key in a format.
`CKA_THALES_DEFINED`	Export an asymmetric key, or asymmetric key with a custom Thales vendor-defined constant.
`CKM_AES_CBC_PAD`	Export an asymmetric or symmetric key with a symmetric key wrapper. The wrapping mechanism needs to be `CKM_RSA_PKCS.`

Table 47: Constants for CK_MECHANISM

Parameter	Description
CKM_THALES_PEM_FORMAT	Export an asymmetric key in a format.
CKA_THALES_DEFINED	Export an asymmetric key, or asymmetric key with a custom Thales vendor-defined constant.
CKM_AES_CBC_PAD	Export an asymmetric or symmetric key with a symmetric key wrapper. The wrapping mechanism needs to be CKM_RSA_PKCS.

Table 48: Output Parameters

Parameter	Description
CK_BYTE_PTR pWrappedKey	The array where CK_BYTE_PTR pWrappedKey points will be filled in with the wrapped key bytes.

Table 49: Return Values

Parameter	Description
CKR_ARGUMENTS_BAD	Already initiated.
CKR_BUFFER_TOO_SMALL	The output of the function is too large to fit in the supplied buffer.
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_FUNCTION_CANCELED	The function was canceled in mid-execution.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.

Table 49: (Continued)Return Values

Parameter	Description
CKR_KEY_HANDLE_INVALID	The specified key handle is not valid.
CKR_KEY_NOT_WRAPPABLE	Unable to wrap the secret key.
CKR_KEY_SIZE_RANGE	The supplied key‘s size is outside the range of key sizes.
CKR_KEY_UNEXTRACTABLE	The specified private or secret key can’t be wrapped because its CKA_EXTRACTABLE attribute is set to CK_FALSE.
CKR_MECHANISM_INVALID	An invalid mechanism was specified to the cryptographic operation.
CKR_MECHANISM_PARAM_INVALID	Invalid parameters were supplied to the mechanism specified to the cryptographic operation.
CKR_OK	The function is executed successfully.
CKR_OPERATION_ACTIVE	There is already an active operation (or combination of active operations) which prevents Cryptoki from activating the specified operation.
CKR_PIN_EXPIRED	The specified PIN has expired, and the requested operation cannot be carried out.
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.
CKR_USER_NOT_LOGGED_IN	The desired action cannot be performed because the appropriate user (or an appropriate user) is not logged in.
CKR_WRAPPING_KEY_HANDLE_INVALID	Indicates that the key handle specified to be used to wrap another key is not valid.
CKR_WRAPPING_KEY_SIZE_RANGE	Indicates that although the requested wrapping operation could in principle be carried out, the token is unable to actually do it because the supplied wrapping key’s size is outside the range of key sizes that it can handle.
CKR_WRAPPING_KEY_TYPE_INCONSISTENT	Indicates that the type of the key specified to wrap another key is not consistent with the mechanism specified for wrapping.

C_UnwrapKey

Used in conjunction with `C_WrapKey`, where a symmetric key is wrapped with another symmetric key on the DSM and then exported.

`C_UnwrapKey` completes the `C_WrapKey` feature, allowing the exported key to be imported to a different DSM (or the same DSM with a different name/label) and ready for future use from the destination.

When exporting and importing the key into a new DSM the key attributes are not preserved. Only the key material is imported. The new template specifies the key attributes.

IMPORTANT: For wrapped keys with set to Store on Server, the unwrap template must include the CKA_THALES_CACHED_ON_HOST attribute set to false.

Reference the code sample `vpkcs11_sample_import_key.c` to use `C_UnwrapKey`.

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

Table 50: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle
`CK_MECHANISM_PTR pMechanism`	The mechanism pointer supports `CKM_AES_CBC_PAD` mode for unwrapping. It requires a 16 byte IV. The structure looks like this: `{CKM_AES_CBC_PAD,iv,16};`
`CK_OBJECT_HANDLE hUnwrappingKey`	The unwrapping key. Do a FIND to obtain the `hUnwrappingKey` handle.
`CK_BYTE_PTR pWrappedKey`	Points to an array to hold the wrapped key bytes as input

Table 50: Input Parameters (Continued)

Parameter	Description
`CK_ULONG_PTR ulWrappedKeyLen`	Length of the above array
`CK_ATTRIBUTE_PTR pTemplate`	Pointer to the new key template such as the `ImportKeyTemplate` portion of `vpkcs11_sample_import_key.c`
`CK_ULONG ulAttributeCount`	Template length (number of attributes)
`CK_OBJECT_HANDLE_PTR phKey`	Returns pointer to a new handle

Table 51: Template Attributes

Template Attribute Name	Type	Value
`CKA_LABEL`	`CK_UTF8CHAR`	Required. Key name; also displayed on the DSM.
`CKA_APPLICATION`	`CK_UTF8CHAR`	Description of the application generating the key name, can be NULL. (Optional)
`CKA_CLASS`	`CK_OBJECT_CLASS`	Required. Must be `CKO_SECRET_KEY`.
`CKA_KEY_TYPE`	`CK_KEY_TYPE`	Required. Must be `CKK_AES`.
`CKA_VALUE_LEN`	`CK_ULONG`	Optional: length of the key imported, passed in bytes. 16 or 32
`CKA_TOKEN`	`CK_BBOOL`	Required. Must be true.
`CKA_ENCRYPT`	`CK_BBOOL`	Default true.
`CKA_DECRYPT`	`CK_BBOOL`	Default true.
`CKA_SIGN`	`CK_BBOOL`	Default false.
`CKA_VERIFY`	`CK_BBOOL`	Default false.
`CKA_WRAP`	`CK_BBOOL`	Default true.
`CKA_UNWRAP`	`CK_BBOOL`	Default false.

Table 51: Template Attributes (Continued)

Template Attribute Name	Type	Value
CKA_EXTRACTABLE	CK_BBOOL	Default false.
CKA_ALWAYS_SENSITIVE	CK_BBOOL	Default false.
CKA_NEVER_EXTRACTABLE	CK_BBOOL	Default true.
CKA_SENSITIVE	CK_BBOOL	Default true.
CKA_THALES_CACHED_ON_HOST	CK_BBOOL	Optional. If attribute is not used, the keys are cached on the host by default. Insert the attribute and set to 0 to store a key on the server. (Equivalent to choosing "Store key on server" in the DSM GUI.) To make storing on the server a default for all keys, use a flag in the `agent.conf` file. See "Storing Keys on the Server by Default" on page 42. This value must be set to false for keys that are stored on server and not cached on host.
CKA_THALES_KEY_CACHED_TIME	CK_ULONG	Optional, in minutes. Between 1 and 44640 by default.

Table 52: Output Parameters

Parameter	Description
CK_OBJECT_HANDLE_PTR phKey	Returns pointer to a new handle that was imported into either the new DSM or the old DSM with a new key name.

C_CreateObject

Imports key bytes into the DSM and creates a symmetric key on the DSM. Maximum length of the key name is 64 characters. There is no minimum length.

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

Table 53: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_ATTRIBUTE pTemplate`	Same template as `C_GenerateKey`, with the addition of `CKA_VALUE` and `CKA_VALUE_LEN`, which hold the key bytes and the length of the key bytes, respectively.
`CK_ULONG ulCount`	Number of attributes in `pTemplate`.
`CK_OBJECT_HANDLE phObject`	Generated key handle.

Table 54: Template Attributes

Template Attribute Name	Type	Value
`CKA_LABEL`	`CK_UTF8CHAR`	Key Name. Also displayed on the Data Security Manager.
`CKA_APPLICATION`	`CK_UTF8CHAR`	Description of the application generating the key name, can be NULL.
`CKA_CLASS`	`CK_OBJECT_CLASS`	Must be `CKO_SECRET_KEY`.
`CKA_KEY_TYPE`	`CK_KEY_TYPE`	Must be `CKK_AES`.
`CKA_VALUE`	`CK_BYTE`	Key bytes.

`CKA_VALUE_LEN`	`CK_ULONG`	Length of key bytes. Pass a long value to the `CK_ATTRIBUTE` constructor when creating the `CKA_VALUE_LEN` attribute. For example:
		`new CK_ATTRIUBTE(CKA_VALUE_LEN, 32L)`
		or:
		`long len = 32;`
		`new CK_ATTRIBUTE(CKA_VALUE_LEN, len)`

`CKA_TOKEN`	`CK_BBOOL`	Required. Must be true.
`CKA_ENCRYPT`	`CK_BBOOL`	Optional, recommend true.
`CKA_DECRYPT`	`CK_BBOOL`	Optional, recommend true.
`CKA_SIGN`	`CK_BBOOL`	Optional

Table 54: Template Attributes (Continued)

Template Attribute Name	Type	Value
CKA_VERIFY	CK_BBOOL	Optional
CKA_WRAP	CK_BBOOL	Optional
CKA_UNWRAP	CK_BBOOL	Optional
CKA_EXTRACTABLE	CK_BOOL	Optional
CKA_ALWAYS_SENSITIVE	CK_BBOOL	Optional
CKA_NEVER_EXTRACTABLE	CK_BBOOL	Optional
CKA_SENSITIVE	CK_BBOOL	Optional

Table 55: Output Parameters

Parameter	Description
CK_OBJECT_HANDLE hGenKey	This parameter is filled in with the handle of the newly generated key.

C_DestroyObject

Deletes a key.

Warning! Use this function VERY CAREFULLY. Deleted keys cannot be recovered and all data encrypted by that key will be lost.

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

Table 56: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_OBJECT_HANDLE hObject`	Handle of the key to be deleted.

Table 57: Return Values

Parameter	Description
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to `C_Initialize`.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	Insufficient memory to perform the requested function.
`CKR_OBJECT_HANDLE_INVALID`	The specified object handle is not valid.
`CKR_OK`	The function is executed successfully.
`CKR_PIN_EXPIRED`	The specified PIN has expired, and the requested operation cannot be carried out.
`CKR_SESSION_CLOSED`	The session was closed during the execution of the function.
`CKR_SESSION_HANDLE_INVALID`	The specified session handle was invalid at the time that the function was invoked.
`CKR_SESSION_READ_ONLY`	The specified session was unable to accomplish the desired action because it is a read-only session.
`CKR_TOKEN_WRITE_PROTECTED`	The requested action could not be performed because the token is write-protected.

C_FindObjectsInit

Initializes a search for a token and session objects that match a template. We only support searching for a key by the CKA_LABEL, which corresponds to the key name on the DSM. The search template must have CKA_LABEL as its required attribute.

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

Table 58: Input Parameters

Parameter	Description
CK_SESSION_HANDLE hSession	Session handle.
CK_ATTRIBUTE_PTR pTemplate	Attribute(s) to search for.
CK_ULONG ulCount	Attribute template count.

The following code shows various attributes that can be searched for:

```
CK_ATTRIBUTE findKeyTemplatePass[] =
{
{CKA_LABEL, ksid, ksid_len},
{CKA_CLASS, &keyType, sizeof(keyType)}
};
switch(keyidType)
{
case keyIdLabel:
findKeyTemplatePass[0].type = CKA_LABEL;
        break;
```

```
    case keyIdUuid:
        findKeyTemplatePass[0].type = CKA_THALES_OBJECT_UUID;
        break;

     case keyIdMuid:
          findKeyTemplatePass[0].type = CKA_THALES_OBJECT_MUID;
          break;

      case keyIdImport:
           findKeyTemplatePass[0].type = CKA_THALES_OBJECT_IKID;
           break;

      case keyIdAlias:
            findKeyTemplatePass[0].type = CKA_THALES_OBJECT_ALIAS;
            break;
}
```

Table 59: Return Values

Parameter	Description
CKR_ARGUMENTS_BAD	Already initiated.
CKR_ATTRIBUTE_TYPE_INVALID	An invalid attribute type was specified in a template.
CKR_ATTRIBUTE_VALUE_INVALID	An invalid value was specified for a particular attribute in a template.
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_OK	The function is executed successfully.
CKR_OPERATION_ACTIVE	There is already an active operation (or combination of active operations) which prevents Cryptoki from activating the specified operation.
CKR_PIN_EXPIRED	The specified PIN has expired, and the requested operation cannot be carried out.

Table 59: (Continued)Return Values

Parameter	Description
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.
CKR_KEY_FUNCTION_NOT_PERMITTED	The Identity check failed.

C_FindObjects

Finds an object on the DSM. The only attribute currently supported is KEY_LABEL, which corresponds to the key name on the DSM. Returns at most 1 key.

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

NOTE: The DSM honors only the public key name. To query for a public key, pass in the public key name and CKO_PUBLIC_KEY. To query for the private key, pass in the public key name and CKO_PRIVATE_KEY.

NOTE: Searching for a key's handle with C_FindObject may not produce accurate results immediately in the same session. To ensure accuracy, wait until the key cache expires.

Table 60: Input Parameters

Parameter	Description
CK_SESSION_HANDLE hSession	Session handle.
CK_OBJECT_HANDLE_PTR phObject	Pointer to the buffer to put the object handles.
CK_ULONG ulMaxObjectCount	The maximum number of objects to put into the buffer.

Table 60: Input Parameters

Parameter	Description
`CK_ULONG_PTR pulObjectCount`	Pointer to CK_ULONG where the number of objects found is returned.

Table 61: Output Parameters

	Description
`CK_OBJECT_HANDLE_PTR phObject`	Returns a single key that matches the CKA_LABEL.
`CK_ULONG_PTR pulObjectCount`	

Table 62: Return Values

Parameter	Description
`CKR_ARGUMENTS_BAD`	Already initiated.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	Insufficient memory to perform the requested function.
`CKR_OK`	The function is executed successfully.
`CKR_OPERATION_NOT_INITIALIZED`	There is no active operation of an appropriate type in the specified session.
`CKR_SESSION_CLOSED`	The session was closed during the execution of the function.
`CKR_SESSION_HANDLE_INVALID`	The specified session handle was invalid at the time that the function was invoked.
`CKR_KEY_FUNCTION_NOT_PERMITTED`	The Identity check failed.

C_FindObjectsFinal

Terminates a search for a token and session objects.

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

Table 63: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.

NOTE: If searching for a versioned key on its alias, the alias must not contain spaces or non-alphanumeric characters.

Table 64: Return Values

Parameter	Description
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	Insufficient memory to perform the requested function.
`CKR_OK`	The function is executed successfully.
`CKR_OPERATION_NOT_INITIALIZED`	There is no active operation of an appropriate type in the specified session.
`CKR_SESSION_CLOSED`	The session was closed during the execution of the function.

Table 64: (Continued)Return Values

Parameter	Description
`CKR_SESSION_HANDLE_INVALID`	The specified session handle was invalid at the time that the function was invoked.
`CKR_KEY_FUNCTION_NOT_PERMITTED`	The Identity check failed.

C_GetAttributeValue

Gets attribute value of a specific Security Object. Argument specifies the Security Object ID.

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

Table 65: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_OBJECT_HANDLE hObject`	Object to get the attribute for.
`CK_ATTRIBUTE_PTR pTemplate`	Template containing the attributes to search for. Points to the `CK_ATTRIBUTE` structure.
`CK_ULONG ulCount`	Number of attributes in the template.

Table 66: Output Parameters

Parameter	Description
`CK_ATTRIBUTE_PTR pTemplate`	This parameter is filled in with the values for the attributes searched for. There are five custom attributes: • `CKA_LABEL` • `CKA_THALES_OBJECT_UUID` • `CKA_THALES_OBJECT_MUID` • `CKA_THALES_OBJECT_IKID` • `CKA_THALES_OBJECT_ALIAS`.

Table 67: Return Values

Parameter	Description
`CKR_ARGUMENTS_BAD`	Already initiated.
`CKR_ATTRIBUTE_SENSITIVE`	An attempt was made to obtain the value of an attribute of an object which cannot be satisfied because the object is either sensitive or un-extractable.
`CKR_ATTRIBUTE_TYPE_INVALID`	An invalid attribute type was specified in a template.
`CKR_BUFFER_TOO_SMALL`	The output of the function is too large to fit in the supplied buffer.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
`CKR_FUNCTION_FAILED`	The function did not execute.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	Insufficient memory to perform the requested function.
`CKR_OBJECT_HANDLE_INVALID`	The specified object handle is not valid.
`CKR_OK`	The function is executed successfully.
`CKR_SESSION_CLOSED`	The session was closed during the execution of the function.

Table 67: Return Values (Continued)

Parameter	Description
`CKR_SESSION_HANDLE_INVALID`	The specified session handle was invalid at the time that the function was invoked.

C_SetAttributeValue

Sets attribute value of a specific Security Object. The argument specifies the Security Object ID.

```
CK_DEFINE_FUNCTION(CK_RV, C_SetAttributeValue)(
CK_SESSION_HANDLE hSession,
CK_OBJECT_HANDLE hObject,
CK_ATTRIBUTE_PTR pTemplate,
CK_ULONG ulCount,
CK_ATTRIBUTE setAttrsTemplate[],
CK_ATTRIBUTE setAttrsTemplateSymm[]
);
```

NOTE: `C_SetAttributeValue` does not work if setting the PROCESS_START state for a key that is in the preactive state.

Table 68: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_OBJECT_HANDLE hObject`	Object handle.
`CK_ATTRIBUTE_PTR pTemplate`	Attribute template for the attributes to set. There area five custom attributes: • `CKA_LABEL` • `CKA_THALES_OBJECT_UUID` • `CKA_THALES_OBJECT_MUID` • `CKA_THALES_OBJECT_IKID` • `CKA_THALES_OBJECT_ALIAS`

Table 68: Input Parameters (Continued)

Parameter	Description
`CK_ULONG ulCount`	Number of attributes in the template.

`CK_ATTRIBUTE setAttrsTemplate[]`	Contains = {
	`{CKA_THALES_KEY_STATE, &state, sizeof(KeyState) }`
	};
	
	`typedef enum {`
	`KeyStatePreActive = 0,`
	`KeyStateActive = 1,`
	`KeyStateSuspended = 2,`
	`KeyStateDeactivated = 3,`
	`KeyStateCompromised = 4,`
	`KeyStateDestroyed = 5`
	`} KeyState;`
	Must not be used together with `setAttrsTemplateSymm[]`. The NIST diagram of key states is included below this table.

`CK_ATTRIBUTE setAttrsTemplateSymm[]`	Contains `{CKA_THALES_KEY_VERSION_ACTION, &keyVersionAction, sizeof(keyVersionAction) }`. Can accept only 2 as the parameter (2 = migration from non-versioned to versioned key.) Must not be used together with `setAttrsTemplate[]`.

Figure 9: NIST key states

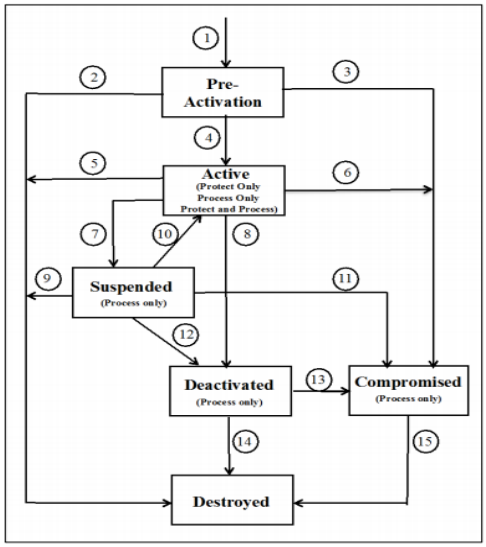


Table 69: Return Values

Parameter	Description
CKR_ARGUMENTS_BAD	Already initiated.
CKR_ATTRIBUTE_READ_ONLY	An attempt was made to set a value for an attribute which may not be set or modified.
CKR_ATTRIBUTE_TYPE_INVALID	An invalid attribute type was specified in a template.
CKR_ATTRIBUTE_VALUE_INVALID	An invalid value was specified for a particular attribute in a template.
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_OBJECT_HANDLE_INVALID	The specified object handle is not valid.
CKR_OK	The function is executed successfully.

Table 69: Return Values (Continued)

Parameter	Description
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.
CKR_SESSION_READ_ONLY	The specified session was unable to accomplish the desired action because it is a read-only session.
CKR_TEMPLATE_INCONSISTENT	The template specified for creating an object has conflicting attributes.
CKR_TOKEN_WRITE_PROTECTED	The requested action could not be performed because the token is write-protected.
CKR_USER_NOT_LOGGED_IN	The desired action cannot be performed because the appropriate user (or an appropriate user) is not logged in.

C_GenerateKey

Generates a symmetric key. Maximum length of the key name is 64 characters. There is no minimum length.

By default, keys are stored on the host or on the DSM. If you don't want them stored on the host, log on to the DSM Console, click the **Keys** tab and select the key name from the **Name** list. From the **Edit Agent Keys** window, in the **Key Type** drop-down menu, select **Stored on Server.**

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

Supported Functionality

AES-128 and AES-256 symmetric encryption keys are supported.

Table 70: Input Parameters

CK_SESSION_HANDLE hSession	
CK_MECHANISM_PTR pMechanism	{CKM_AES_KEY_GEN, NULL_PTR, 0};
CK_ULONG ulCount	
CK_OBJECT_HANDLE_PTR phKey	
CKA_VORM_CACHED_ON_HOST	
CKA_VORM_CACHED_TIME	
CK_ATTRIBUTE_PTR pTemplate	

Table 71: Template Attributes

Attribute Name	Type	Description
CKA_LABEL	CK_UTF8CHAR	Key name; also displayed on the Data Security Manager
CKA_APPLICATION	CK_UTF8CHAR	Description of the application generating the key name, can be NULL.
CKA_CLASS	CK_OBJECT_CLASS	Must be CKO_SECRET_KEY.
CKA_KEY_TYPE	CK_KEY_TYPE	Must be CKK_AES.
CKA_VALUE_LEN	CK_ULONG	Key size.
CKA_TOKEN	CK_BBOOL	Required. Must be true.
CKA_ENCRYPT	CK_BBOOL	Optional. Default true.
CKA_DECRYPT	CK_BBOOL	Optional. Default true.

Table 71: Template Attributes (Continued)

Attribute Name	Type	Description
CKA_SIGN	CK_BBOOL	Optional. Default false.
CKA_VERIFY	CK_BBOOL	Optional. Default false.
CKA_WRAP	CK_BBOOL	Optional. Default true.
CKA_UNWRAP	CK_BBOOL	Optional. Default false.
CKA_EXTRACTABLE	CK_BBOOL	Optional. Default false.
CKA_ALWAYS_SENSITIVE	CK_BBOOL	Optional. Default false.
CKA_NEVER_EXTRACTABLE	CK_BBOOL	Optional.
CKA_SENSITIVE	CK_BBOOL	Optional.
CKA_THALES_CACHED_ON_HOST	CK_BBOOL	Optional. If attribute is not used, the keys are cached on the host by default. Insert the attribute and set to 0 to store a key on the server. (Equivalent to choosing "Store key on server" in the DSM GUI.) To make storing on the server a default for all keys, use a flag in the `agent.conf` file. See "Storing Keys on the Server by Default" on page 42.
CKA_THALES_KEY_CACHED_TIME	CK_ULONG	In minutes. Default is 44640 (31 days).
CKA_THALES_DATE_KEY_DEACTIVATION	CK_DATE	Epoch time as CK_DATE.
CKA_THALES_KEY_VERSION_ACTION	CK_VERSIONACTION	Can be omitted; or takes 0, 1, or 3 as a parameter. 0 creates a base version of a versioned key, 1 rotates a new version for a versioned key, and 3 indicates that the key is non-versioned. See "Automated Key Versioning" on page 43
CKA_THALES_KEY_VERSION_LIFE_SPAN	CK_ULIFESPAN	Must be greater than 0. Specifies the time until automatic key rotation. Omit this attribute if no key rotation is desired. See "Automated Key Versioning" on page 43

NOTE: If automatic key rotation is set, the key will rotate 24 hours *before* the specified rotation period. This behavior is expected.

NOTE: State changes are supported only by explicitly setting the new state, never by changing the dates.

Table 72: Output Parameters

	Description
`CK_OBJECT_HANDLE_PTR phKey`	The structure where `CK_OBJECT_HANDLE_PTR phKey` points is filled in with the newly created key handle.

Table 73: Return Values

Parameter	Description
`CKR_ARGUMENTS_BAD`	Already initiated.
`CKR_ATTRIBUTE_READ_ONLY`	An attempt was made to set a value for an attribute which may not be set or modified.
`CKR_ATTRIBUTE_TYPE_INVALID`	An invalid attribute type was specified in a template.
`CKR_ATTRIBUTE_VALUE_INVALID`	An invalid value was specified for a particular attribute in a template.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
`CKR_FUNCTION_CANCELED`	The function was canceled in mid-execution.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	Insufficient memory to perform the requested function.
`CKR_MECHANISM_INVALID`	An invalid mechanism was specified to the cryptographic operation.
`CKR_MECHANISM_PARAM_INVALID`	Invalid parameters were supplied to the mechanism specified to the cryptographic operation.
`CKR_OK`	The function is executed successfully.

Table 73: Return Values

Parameter	Description
CKR_OPERATION_ACTIVE	There is already an active operation (or combination of active operations) which prevents Cryptoki from activating the specified operation.
CKR_PIN_EXPIRED	The specified PIN has expired, and the requested operation cannot be carried out.
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.
CKR_SESSION_READ_ONLY	The specified session was unable to accomplish the desired action because it is a read-only session.
CKR_TEMPLATE_INCOMPLETE	The template specified for creating an object is incomplete, and lacks some necessary attributes.
CKR_TEMPLATE_INCONSISTENT	The template specified for creating an object has conflicting attributes.
CKR_TOKEN_WRITE_PROTECTED	The requested action could not be performed because the token is write-protected.
CKR_USER_NOT_LOGGED_IN	The desired action cannot be performed because the appropriate user (or an appropriate user) is not logged in.

C_GenerateKeyPair

Generates an asymmetric key pair. Arguments specify RSA type and length of the key pair.

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

Supported Functionality

RSA-1024 and RSA-2048 asymmetric key pairs are supported.

Table 74: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_MECHANISM_PTR pMechanism`	Key generation mechanism must have the following values: `{ CKM_RSA_PKCS_KEY_PAIR_GEN, NULL_PTR, 0 };`

`CK_ATTRIBUTE_PTR pPublicKeyTemplate`	`{CKA_LABEL, publicKeyLabel, sizeof(publicKeyLabel) },/* Keyname */`
	`{CKA_CLASS, &pubkey_class, sizeof(pubkey_class)},`
	`{CKA_ENCRYPT, &bTrue, sizeof(bTrue)},`
	`{CKA_VERIFY, &bTrue, sizeof(bTrue)},`
	`{CKA_WRAP, &bTrue, sizeof(bTrue)},`
	`{CKA_TOKEN, &bTrue, sizeof(bTrue)},`
	`{CKA_PUBLIC_EXPONENT, publicExponent, sizeof(publicExponent)},`
	`{CKA_MODULUS_BITS, &modulusBits, sizeof(modulusBits)}`

`CK_ULONG ulPublicKeyAttributeCount`	Size of the key template `publicKeyTemplate`.

`CK_ATTRIBUTE_PTR privateKeyTemplate`	`{CKA_LABEL, privateKeyLabel, sizeof(publicKeyLabel) },/* Keyname*/`
	`{CKA_CLASS, &privkey_class, sizeof(privkey_class)},`
	`{CKA_TOKEN, &bTrue, sizeof(bTrue)},`
	`{CKA_PRIVATE, &bTrue, sizeof(bTrue)},`
	`{CKA_SENSITIVE, &bTrue, sizeof(bTrue)},`
	`{CKA_DECRYPT, &bTrue, sizeof(bTrue)},`
	`{CKA_SIGN, &bTrue, sizeof(bTrue)},`
	`{CKA_UNWRAP, &bTrue, sizeof(bTrue)}`

`CK_ULONG ulPrivateKeyAttributeCount`	Size of the key template `privateKeyTemplate`.
`CK_OBJECT_HANDLE_PTR phPublicKey`	Filled in with the newly created public key handle.
`CK_OBJECT_HANDLE_PTR phPrivateKey`	Filled in with the newly created private key handle.

Table 75: Output Parameters

Parameter	Description
CK_OBJECT_HANDLE_PTR phPublicKey	CK_OBJECT_HANDLE_PTR phPublicKey is filled in with the object handle of the public key.
CK_OBJECT_HANDLE_PTR phPublicKey	CK_OBJECT_HANDLE_PTR phPublicKey is filled in with the object handle of the private key.

Table 76: Return Values

Parameter	Description
CKR_ARGUMENTS_BAD	Already initiated.
CKR_ATTRIBUTE_READ_ONLY	An attempt was made to set a value for an attribute which may not be set or modified.
CKR_ATTRIBUTE_TYPE_INVALID	An invalid attribute type was specified in a template.
CKR_ATTRIBUTE_VALUE_INVALID	An invalid value was specified for a particular attribute in a template.
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_DOMAIN_PARAMS_INVALID	Invalid or unsupported domain parameters were supplied to the function.
CKR_FUNCTION_CANCELED	The function was canceled in mid-execution.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_MECHANISM_INVALID	An invalid mechanism was specified to the cryptographic operation.
CKR_MECHANISM_PARAM_INVALID	Invalid parameters were supplied to the mechanism specified to the cryptographic operation.
CKR_OK	The function is executed successfully.
CKR_OPERATION_ACTIVE	There is already an active operation (or combination of active operations) which prevents Cryptoki from activating the specified operation.

Table 76: Return Values

Parameter	Description
CKR_PIN_EXPIRED	The specified PIN has expired, and the requested operation cannot be carried out.
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.
CKR_SESSION_READ_ONLY	The specified session was unable to accomplish the desired action because it is a read-only session.
CKR_TEMPLATE_INCOMPLETE	The template specified for creating an object is incomplete, and lacks some necessary attributes.
CKR_TEMPLATE_INCONSISTENT	The template specified for creating an object has conflicting attributes.
CKR_TOKEN_WRITE_PROTECTED	The requested action could not be performed because the token is write-protected.
CKR_USER_NOT_LOGGED_IN	The desired action cannot be performed because the appropriate user (or an appropriate user) is not logged in.

Digest and MAC Functions

C_DigestInit

Initializes a digest operation.

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

Table 77: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.

`CK_MECHANISM_PTR pMechanism`	Which cryptographic hash function to use. One of the following values:
	`{ CKM_SHA256, NULL_PTR, 0 },`
	`{ CKM_SHA384, NULL_PTR, 0 },`
	`{ CKM_SHA512, NULL_PTR, 0 }`
	OR
	`{ CKM_SHA256_HMAC, NULL_PTR, 0 }`
	Other mechanisms include:
	CKM_MD5
	CKM_SHA_1
	CKM_SHA224
	CKM_MD5_HMAC
	CKM_SHA_1_HMAC
	CKM_SHA224_HMAC
	CKM_SHA384_HMAC
	CKM_SHA512_HMAC

Table 78: Return Values

Parameter	Description
`CKR_ARGUMENTS_BAD`	Already initiated.

Table 78: Return Values

Parameter	Description
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
`CKR_FUNCTION_CANCELED`	The function was canceled in mid-execution.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	Insufficient memory to perform the requested function.
`CKR_MECHANISM_INVALID`	An invalid mechanism was specified to the cryptographic operation.
`CKR_MECHANISM_PARAM_INVALID`	Invalid parameters were supplied to the mechanism specified to the cryptographic operation.
`CKR_OK`	The function is executed successfully.
`CKR_SESSION_HANDLE_INVALID`	The specified session handle was invalid at the time that the function was invoked.

C_Digest

Compute data digest in a single part. Must call `C_DigestInit` before calling `C_Digest`.

```
CK_DEFINE_FUNCTION(CK_RV, C_DigestInit)(
   CK_SESSION_HANDLE hSession,
   CK_BYTE_PTR pPart,
   CK_ULONG ulPartLen,
    CK_BYTE_PTR pDigest,
    CK_ULONG_PTR pulDigestLen
  );
```

Table 79: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.

Table 79: Input Parameters

Parameter	Description
`CK_BYTE_PTR pPart`	Pointer to the digest input data.
`CK_ULONG ulPartLen`	Length of the data.
`CK_BYTE_PTR pDigest`	Results in the message digest.
`CK_ULONG_PTR pulDigestLen`	The byte count of the digest.

Table 80: Return Values

Parameter	Description
`CKR_ARGUMENTS_BAD`	Already initiated.
`CKR_BUFFER_TOO_SMALL`	The output of the function is too large to fit in the supplied buffer.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to `C_Initialize`.
`CKR_FUNCTION_CANCELED`	The function was canceled in mid-execution.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	Insufficient memory to perform the requested function.
`CKR_MECHANISM_INVALID`	An invalid mechanism was specified to the cryptographic operation.
`CKR_MECHANISM_PARAM_INVALID`	Invalid parameters were supplied to the mechanism specified to the cryptographic operation.
`CKR_OK`	The function is executed successfully.
`CKR_OPERATION_NOT_INITIALIZED`	There is no active operation of an appropriate type in the specified session.
`CKR_SESSION_HANDLE_INVALID`	The specified session handle was invalid at the time that the function was invoked.

C_DigestKey

Specifies HMAC key for digest operation. Must call `C_DigestInit` before calling `C_DigestKey`.

```
CK_DEFINE_FUNCTION(CK_RV, C_DigestKey)(
    CK_SESSION_HANDLE hSession,
    CK_OBJECT_HANDLE hKey
  );
```

NOTE: This function does not work when key is set to `Store on Server`. The key must be set to `Cached on Host`.

NOTE: The key can be either AES or an opaque object.

NOTE: You can call `C_DigestKey` only once and directly after `C_DigestInit`.

Table 81: Input Parameter

	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_OBJECT_HANDLE hKey`	

Table 82: Return Values

Parameter	Description
`CKR_ARGUMENTS_BAD`	Already initiated.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to `C_Initialize`.
`CKR_FUNCTION_CANCELED`	The function was canceled in mid-execution.

Table 82: Return Values

Parameter	Description
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_KEY_FUNCTION_NOT_PERMITTED	
CKR_KEY_HANDLE_INVALID	The specified key handle is not valid.
-CKR_KEY_INDIGESTIBLE	The supplied key‘s size is outside the range of key sizes.
CKR_KEY_SIZE_RANGE	
CKR_MECHANISM_PARAM_INVALID	Invalid parameters were supplied to the mechanism specified to the cryptographic operation.
CKR_OK	The function is executed successfully.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.

C_DigestUpdate

Continues a multi-part digest operation. Must call `C_DigestInit` before calling `C_DigestUpdate`. Can optionally call `C_DigestKey` before calling `C_DigestUpdate`.

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

Table 83: Input Parameters

Parameter	Description
CK_SESSION_HANDLE hSession	Session handle.

Table 83: Input Parameters

Parameter	Description
`CK_BYTE_PTR pPart`	Pointer to the digest input data.
`CK_ULONG ulPartLen`	Length of the data.

Table 84: Return Values

Parameter	Description
`CKR_ARGUMENTS_BAD`	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to `C_Initialize`.
`CKR_FUNCTION_CANCELED`	The function was canceled in mid-execution.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	Insufficient memory to perform the requested function.
`CKR_KEY_FUNCTION_NOT_PERMITTED`	An attempt has been made to use a key for a cryptographic purpose that the key's attributes are not set to allow it to do.
`CKR_KEY_HANDLE_INVALID`	The specified key handle is not valid.
`CKR_KEY_SIZE_RANGE`	The supplied key's size is outside the range of key sizes.
`CKR_KEY_TYPE_INCONSISTENT`	The specified key is not the correct type of key to use with the specified mechanism.
`CKR_OK`	The function is executed successfully.
`CKR_OPERATION_NOT_INITIALIZED`	There is no active operation of an appropriate type in the specified session.
`CKR_SESSION_HANDLE_INVALID`	The specified session handle was invalid at the time that the function was invoked.

C_DigestFinal

Finishes a multi-part digest operation. This works only with the local key cache. Not supported if crypto is done remotely on the DSM.

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

Table 85: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_BYTE_PTR pDigest`	Results in the message digest.
`CK_ULONG_PTR pulDigestLen`	The byte count of the digest.

Table 86: Output Parameters

Parameter	Description
`CKR_ARGUMENTS_BAD`	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.
`CKR_BUFFER_TOO_SMALL`	The output of the function is too large to fit in the supplied buffer.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to `C_Initialize`.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.

Parameter (Continued)	**Description**
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_KEY_HANDLE_INVALID	The specified key handle is not valid.
CKR_KEY_SIZE_RANGE	The supplied key‘s size is outside the range of key sizes.
CKR_KEY_TYPE_INCONSISTENT	The specified key is not the correct type of key to use with the specified mechanism.
CKR_OK	The function is executed successfully.
CKR_OPERATION_NOT_INITIALIZED	There is no active operation of an appropriate type in the specified session.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.

Signing and Calculating MAC Functions

C_SignInit

Initializes a signature operation.

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

Table 87: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_MECHANISM_PTR pMechanism`	The mechanism should have the following parameters `;{ CKM_RSA_PKCS, NULL_PTR, 0 }`. Available mechanisms include: • HMAC-SHA 1 • CKM_SHA224_HMAC • CKM_SHA256_HMAC • CKM_SHA384_HMAC • CKM_SHA512_HMAC • RSA
`CK_OBJECT_HANDLE hKey`	Handle to the RSA private key. The key that this handle references can be symmetric, asymmetric, or an opaque object.

Table 88: Return Values

Parameter	Description
`CKR_ARGUMENTS_BAD`	Already initiated.

Table 88: Return Values

Parameter	Description
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_FUNCTION_CANCELED	The function was canceled in mid-execution.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_KEY_FUNCTION_NOT_PERMITTED	An attempt has been made to use a key for a cryptographic purpose that the key's attributes are not set to allow it to do.
CKR_KEY_HANDLE_INVALID	The specified key handle is not valid.
CKR_KEY_SIZE_RANGE	The supplied key's size is outside the range of key sizes.
CKR_KEY_TYPE_INCONSISTENT	The specified key is not the correct type of key to use with the specified mechanism.
CKR_MECHANISM_INVALID	An invalid mechanism was specified to the cryptographic operation.
CKR_MECHANISM_PARAM_INVALID	Invalid parameters were supplied to the mechanism specified to the cryptographic operation.
CKR_OK	The function is executed successfully.
CKR_OPERATION_ACTIVE	There is already an active operation (or combination of active operations) which prevents Cryptoki from activating the specified operation.
CKR_PIN_EXPIRED	The specified PIN has expired, and the requested operation cannot be carried out.
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.
CKR_USER_NOT_LOGGED_IN	The desired action cannot be performed because the appropriate user (or an appropriate user) is not logged in.

C_Sign

Signs data in a single part.

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

Table 89: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_BYTE_PTR pData`	Points to the data.
`CK_ULONG ulDataLen`	Length of data.
`CK_BYTE_PTR pSignature`	Pointer to buffer to receive the signed data.
`CK_ULONG_PTR pulSignatureLen`	Pointer to buffer length.

Table 90: Output Parameters

Parameter	Description
`CK_BYTE`	The `CK_BYTE` structure where `pSignature` points is filled in with the signed data.
`CK_ULONG`	The `CK_ULONG` structure where `pulSignatureLen` points is filled in with the signed data length.

Table 91: Return Values

Parameter	Description
`CKR_ARGUMENTS_BAD`	Already initiated.

Table 91: Return Values

Parameter	Description
CKR_BUFFER_TOO_SMALL	The output of the function is too large to fit in the supplied buffer.
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_DATA_INVALID	The plaintext input data to a cryptographic operation is invalid.
CKR_DATA_LEN_RANGE	The provided signature/MAC can be seen to be invalid solely on the basis of its length.
CKR_FUNCTION_CANCELED	The function was canceled in mid-execution.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_OK	The function is executed successfully.
CKR_OPERATION_NOT_INITIALIZED	There is no active operation of an appropriate type in the specified session.
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.
CKR_USER_NOT_LOGGED_IN	The desired action cannot be performed because the appropriate user (or an appropriate user) is not logged in.
CKR_FUNCTION_REJECTED	The signature request is rejected by the user.

C_VerifyInit

Initializes a verification operation.

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

Table 92: Input Parameters

Parameter	Description
`hSession`	Session handle.
`pMechanism`	Pointer to the structure that specifies the verification mechanism. Available mechanisms include: • HMAC-SHA 1 • CKM_SHA224_HMAC • CKM_SHA256_HMAC • CKM_SHA384_HMAC • CKM_SHA512_HMAC • RSA
`hKey`	Handle for the public key of the key pair used for verification.

Table 93: Return Values

Parameter	Description
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.

Table 93: (Continued)Return Values

Parameter	Description
CKR_KEY_FUCNTION_NOT_PERMITTED	An attempt has been made to use a key for a cryptographic purpose that the key's attributes are not set to allow it to do.
CKR_KEY_HANDLE_INVALID	The specified key handle is not valid.
CKR_MECHANISM_INVALID	An invalid mechanism was specified to the cryptographic operation.
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.
CKR_USER_NOT_LOGGED_IN	The desired action cannot be performed because the appropriate user (or an appropriate user) is not logged in.

C_Verify

Performs a single-part verification operation. To perform verification, the signature is verified with the key, the handle of which is provided in `C_Verify_Init` and compared with the `pData`.

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

Table 94: Return Values

Parameter	Description
CKR_OK	The function is executed successfully.
CKR_SIGNATURE_INVALID	The provided signature/MAC is invalid.
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to `C_Initialize`.

Table 94: (Continued)Return Values

Parameter	Description
`CKR_SIGNATURE_LEN_RANGE`	The provided signature/MAC can be seen to be invalid solely on the basis of its length.

Encryption Functions

C_EncryptInit

Initializes an encryption operation. `C_EncryptInit` must be called before `C_Encrypt` or `C_EncryptUpdate` is called:

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

NOTE: When using `CKM_AES_GCM`, the `pParameter` field of the `CK_MECHANISM` structure must point to the address of a `CK_GCM_PARAMS` structure, and the `ulParameterLen` field must equal the length of the structure.

Table 95: C_EncryptInit Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.

`CK_MECHANISM_PTR pMechanism`	The pointer to the structure that specifies the encryption mechanism.
	We support the following mechanisms:
	`CKM_AES_ECB`
	`CKM_AES_CBC`
	`CKM_AES_CBC_PAD`
	`CKM_AES_CTR`
	`CKM_THALES_FPE` (in Java, use: `0x80004001L`)
	`CKM_THALES_FF1` (in Java, use: `0x80004002L`)
	`CKM_RSA_PKCS`
	`CKM_AES_GCM`
	Note: For `CKM_THALES_FPE`, the IV contains the following field values:
	• tweak (8 bytes)
	• character set. ASCII or Unicode characters. The following Unicode encodings are supported: UTF-8, UTF-16, UTF-32, big-endian, or little-endian byte order. Be sure there are no duplicate characters in the character set.
	• radix (character set size). The size can be from 2 to 65535. For more information, see samples.
	For `CKM_THALES_FF1`, the IV contains the same 3 field values, except that the tweak is not fixed size: it can range from 0 to 2^{32} bytes.
	Note: FPE, CTR, and FF1 do not work when key is set to `Store on Server`. The key must be set to `Cached on Host`.

`CK_OBJECT_HANDLE hKey`	The handle to the encryption key to do the encryption.

NOTE: Not every header version supports every encryption mechanism. Refer to the table titled “Header Compatibility with Encryption Methods” earlier in this manual.

Table 96: Return Values

Parameter	Description
`CKR_FUNCTION_CANCELED`	The function was canceled in mid-execution.

Table 96: (Continued)Return Values

Parameter	Description
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_ARGUMENTS_BAD	Already initiated.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_KEY_FUNCTION_NOT_PERMITTED	An attempt has been made to use a key for a cryptographic purpose that the key's attributes are not set to allow it to do.
CKR_KEY_HANDLE_INVALID	The specified key handle is not valid.
CKR_KEY_SIZE_RANGE	The supplied key's size is outside the range of key sizes.
CKR_KEY_TYPE_INCONSISTENT	The specified key is not the correct type of key to use with the specified mechanism.
CKR_MECHANISM_INVALID	An invalid mechanism was specified to the cryptographic operation.
CKR_MECHANISM_PARAM_INVALID	Invalid parameters were supplied to the mechanism specified to the cryptographic operation.
CKR_OK	The function is executed successfully.
CKR_OPERATION_ACTIVE	There is already an active operation (or combination of active operations) which prevents Cryptoki from activating the specified operation.
CKR_PIN_EXPIRED	The specified PIN has expired, and the requested operation cannot be carried out.
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.
CKR_USER_NOT_LOGGED_IN	The desired action cannot be performed because the appropriate user (or an appropriate user) is not logged in.

C_Encrypt

Encrypts single part data. Must call `C_EncryptInit` before calling `C_Encrypt`.

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

Table 97: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_BYTE_PTR pData`	Pointer to data to be encrypted.
`CK_ULONG ulDataLen`	Length of the data to be encrypted.
`CK_BYTE_PTR pEncryptedData`	Pointer of the buffer to put the data. First call: Pointer to the metadata to be passed and logged on the application's behalf. Must start with "META:"; input parameter. Second call: Pointer to the resulting encrypted data; output parameter.
`CK_ULONG_PTR pulEncryptedDataLen`	Pointer to encrypted buffer length. First call: Pointer to encrypted buffer length. Encrypted buffer length is zero for metadata logging. Second call: actual buffer length used for encrypted data, written encrypted data length. For GCM this parameter contains the length of the tag in bytes, plus the length of the remaining ciphertext from encryption.

Table 98: Output Parameters

Parameter	Description
`pEncryptedData`	When input parameter `pEncryptedData` is NULL, this function returns a calculated output buffer length value pointed to by `pulEncryptedDataLen`. For C program only. For GCM the tag of length specified in the `ulTagBits` field for CK_GCM_PARAMS in bytes is appended to the end of the buffer.

Table 99: Return Values

Parameter	Description
CKR_OK	The function is executed successfully.
CKR_ARGUMENTS_BAD	Already initiated.
CKR_BUFFER_TOO_SMALL	The output of the function is too large to fit in the supplied buffer.
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_DATA_INVALID	The plaintext input data to a cryptographic operation is invalid.
CKR_DATA_LEN_RANGE	The plaintext input data to a cryptographic operation has a bad length.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_OPERATION_NOT_INITIALIZED	There is no active operation of an appropriate type in the specified session.
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.
CKR_PIN_INVALID	The specified PIN has invalid characters in it.
CKR_MECHANISM_INVALID	An invalid mechanism was specified to the cryptographic operation.

C_EncryptUpdate

Continues a multi-part encryption. This works only with the local key cache. Not supported if crypto is done remotely on the DSM.

```
CK_DEFINE_FUNCTION(CK_RV, C_EncryptUpdate)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR       pPart,
  CK_ULONG          ulPartLen,
  CK_BYTE_PTR       pEncryptedPart,
  CK_ULONG_PTR      pulEncryptedPartLen
  );
```

Table 100: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_BYTE_PTR pPart`	Pointer to the data to be encrypted.
`CK_ULONG ulPartLen`	Length of the data.
`CK_BYTE_PTR pEncryptedPart`	Pointer to the data buffer. First call: Pointer to the metadata to be passed and logged on the application's behalf. Must start with "META:"; input parameter. Second call: Pointer to the resulting encrypted data; output parameter.
`CK_ULONG_PTR pulEncryptedPartLen`	Pointer to encrypted buffer length. First call: Pointer to encrypted buffer length. Encrypted buffer length is zero for metadata logging. Second call: actual buffer length used for encrypted data, written encrypted data length.

Table 101: Output Parameters

Parameter	Description
`pEncryptedPart`	When input parameter `pEncryptedPart` is NULL, this function returns a calculated output buffer length value pointed to by `pulEncryptedPartLen`.

Table 102: Return Values

Parameter	Description
CKR_OK	The function is executed successfully.
CKR_ARGUMENTS_BAD	Already initiated.
CKR_BUFFER_TOO_SMALL	The output of the function is too large to fit in the supplied buffer.
CKR_CRYPTOKI_NOT_INITIALIZED	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to C_Initialize.
CKR_DATA_INVALID	The plaintext input data to a cryptographic operation is invalid.
CKR_DATA_LEN_RANGE	The plaintext input data to a cryptographic operation has a bad length.
CKR_FUNCTION_CANCELED	The function was canceled in mid-execution.

C_EncryptFinal

Finishes a multi-part encryption. This works only with the local key cache. Not supported if encryption is done remotely on the DSM.

```
CK_DEFINE_FUNCTION(CK_RV, C_EncryptFinal)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR       pLastEncryptedPart,
  CK_ULONG_PTR      pulEncryptedPartLen
);
```

Table 103: Input Parameters

Parameter	Description
CK_SESSION_HANDLE hSession	Session handle.

Table 103: Input Parameters

Parameter	Description
`CK_BYTE_PTR pLastEncryptedPart`	Pointer to any remaining data to be encrypted. First call: Pointer to the metadata to be passed and logged on the application's behalf. Must start with "META:"; input parameter. Second call: Pointer to the resulting encrypted data; output parameter.
`CK_ULONG_PTR pulLastEncryptedPartLen`	Buffer length pointer. First call: Pointer to encrypted buffer length. Encrypted buffer length is zero for metadata logging. Second call: actual buffer length used for encrypted data, written encrypted data length. For GCM, this parameter contains the length of the tag in bytes plus the length of the remaining ciphertext from encryption.

Table 104: Output Parameters

`Parameter`	Description
`pLastEncryptedPart`	When input parameter pLastEncryptedPart is NULL, this function returns a calculated output buffer length value pointed to by pulLastEncryptedPartLen. For C program only. For GCM, the tag of length is specified in the `ulTagBits` field for `CK_GCM_PARAMS` in bytes, at the end of the buffer.

Table 105: Return Values

Parameter	Description
`CKR_OK`	The function is executed successfully.
`CKR_ARGUMENTS_BAD`	Already initiated.
`CKR_BUFFER_TOO_SMALL`	The output of the function is too large to fit in the supplied buffer.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to `C_Initialize`.
`CKR_DATA_INVALID`	The plaintext input data to a cryptographic operation is invalid.

Table 105: (Continued)Return Values

Parameter	Description
CKR_DATA_LEN_RANGE	The plaintext input data to a cryptographic operation has a bad length.
CKR_FUNCTION_CANCELED	The function was canceled in mid-execution.

Decryption Functions

C_DecryptInit

Initializes a decryption option. `C_DecryptInit` must be called before `C_Decrypt` or `C_DecryptUpdate`.

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

NOTE: When using `CKM_AES_GCM`, the `pParameter` field of the `CK_MECHANISM` structure must point to the address of a `CK_GCM_PARAMS` structure, and the `ulParameterLen` field is the length of the structure.

Table 106: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.

Table 106: Input Parameters (Continued)

Parameter	Description
`CK_MECHANISM_PTR pMechanism`	Pointer to the mechanism for decryption. Must match the mechanism used to encrypt, including the correct IV if using CKM_AES_CBC or CKM_AES_CBC_PAD or the correct nonce and counter concatenated if using CKM_AES_CTR. We support the following mechanisms: CKM_AES_ECB CKM_AES_CBC CKM_AES_CBC_PAD CKM_AES_CTR CKM_THALES_FPE (in Java, use: 0x80004001L) CKM_THALES_FF1 (in Java, use: 0x80004002L) CKM_RSA_PKCS CKM_AES_GCM **Note:** For CKM_THALES_FPE, the IV contains the following field values: • tweak (8 bytes) • character set (ASCII characters 0-127) • radix (character set size). For more information see samples. For CKM_THALES_FF1, the IV contains the same field values, except that the tweak is not a fixed size: it can range from 0 to 2^{32} bytes.
`CK_OBJECT_HANDLE hKey`	Handle of the decryption key; (same as encryption key for symmetric encryption).
`CK_GCM_PARAMS ulEncryptedPartLen`	For GCM, this parameter must include the length of the tag in bytes, in addition to the ciphertext length.

Table 107: Return Values

Parameter	Description
`CKR_OK`	The function is executed successfully.
`CKR_ARGUMENTS_BAD`	Already initiated.
`CKR_BUFFER_TOO_SMALL`	The output of the function is too large to fit in the supplied buffer.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to `C_Initialize`.

Table 107: (Continued)Return Values

Parameter	Description
CKR_DATA_INVALID	The plaintext input data to a cryptographic operation is invalid.
CKR_DATA_LEN_RANGE	The plaintext input data to a cryptographic operation has a bad length.
CKR_FUNCTION_CANCELED	The function was canceled in mid-execution.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_KEY_FUNCTION_NOT_PERMITTED	An attempt has been made to use a key for a cryptographic purpose that the key's attributes are not set to allow it to do.
CKR_KEY_HANDLE_INVALID	The specified key handle is not valid.
CKR_KEY_SIZE_RANGE	The supplied key's size is outside the range of key sizes.
CKR_KEY_TYPE_INCONSISTENT	The specified key is not the correct type of key to use with the specified mechanism.
CKR_MECHANISM_INVALID	An invalid mechanism was specified to the cryptographic operation.
CKR_MECHANISM_PARAM_INVALID	Invalid parameters were supplied to the mechanism specified to the cryptographic operation.
CKR_OPERATION_ACTIVE	There is already an active operation (or combination of active operations) which prevents Cryptoki from activating the specified operation.
CKR_PIN_EXPIRED	The specified PIN has expired, and the requested operation cannot be carried out.
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.
CKR_USER_NOT_LOGGED_IN	The desired action cannot be performed because the appropriate user (or an appropriate user) is not logged in.

C_Decrypt

Decrypts encrypted data in a single part. The operation must have been initialized by a prior call to `C_DecryptInit`.

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

Table 108: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_BYTE_PTR pEncryptedData`	Pointer to the encrypted data to be decrypted.
`CK_ULONG ulEncryptedDataLen`	Length of the encrypted data to be decrypted. For GCM this parameter must also include the tag length, in bytes added to the ciphertext length.
`CK_BYTE_PTR pData`	Pointer to the buffer for the decrypted text. First call: pointer to the metadata to be passed and logged on application's behalf. Must start with "META:"; input parameter. Second call: pointer to the resulting decrypted data; (output parameter).
`CK_ULONG_PTR pulDataLen`	Pointer to the length of the buffer for the decrypted text. First call: Pointer to decrypted buffer length. Decrypted buffer length is zero for metadata logging. Second call: actual buffer length used for decrypted data; length of the written plaintext.
`CK_GCM_PARAMS`	The first bytes of length are specified in the `ulTagBits` field and `pEncryptedData` must contain the tag. Also for GCM, if the decryption can't be validated, the error `CKR_ENCRYPTED_DATA_INVALID` is returned.

Table 109: Output Parameters

Parameter	Description
`pData`	When input parameter `pData` is NULL, this function returns a calculated output buffer length value pointed to by `pulDataLen`. For C program only.

Table 110: Return Values

Parameter	Description
`CKR_OK`	The function is executed successfully.
`CKR_ARGUMENTS_BAD`	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.
`CKR_BUFFER_TOO_SMALL`	The output of the function is too large to fit in the supplied buffer.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to `C_Initialize`.
`CKR_DATA_INVALID`	The plaintext input data to a cryptographic operation is invalid.
`CKR_DATA_LEN_RANGE`	The plaintext input data to a cryptographic operation has a bad length.
`CKR_FUNCTION_CANCELED`	The function was canceled in mid-execution.
`CKR_MECHANISM_INVALID`	An invalid mechanism was specified to the cryptographic operation.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.
`CKR_HOST_MEMORY`	Insufficient memory to perform the requested function.
`CKR_OPERATION_NOT_INITIALIZED`	There is no active operation of an appropriate type in the specified session.
`CKR_SESSION_CLOSED`	The session was closed during the execution of the function.
`CKR_SESSION_HANDLE_INVALID`	The specified session handle was invalid at the time that the function was invoked.

Table 110: (Continued)Return Values

Parameter	Description
CKR_USER_NOT_LOGGED_IN	The desired action cannot be performed because the appropriate user (or an appropriate user) is not logged in.
CKR_MECHANISM_INVALID	An invalid mechanism was specified to the cryptographic operation.

C_DecryptUpdate

Decrypts multi-part encrypted data. The operation must have been initialized by a prior call to `C_DecryptInit`. This works only with the local key cache. Not supported if crypto is done remotely on the DSM.

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

Table 111: Input Parameters

Parameter	Description
CK_SESSION_HANDLE hSession	Session handle.
CK_BYTE_PTR pEncryptedData	Pointer to the encrypted data. For GCM, on the first call to `C_DecryptUpdate`, this parameter includes the first bytes of length specified in the `ulTagBits` field in bytes and `pEncryptedPart` must contain the tag.
CK_ULONG ulEncryptedDataLen	Length of the encrypted data. For GCM, his parameter must include the length of the tag in bytes, in addition to the ciphertext length.
CK_BYTE_PTR pData	Pointer to a buffer to write the decrypted data. First call: Pointer to the metadata to be passed and logged on the application's behalf. Must start with "META:"; input parameter. Second call: Pointer to the resulting encrypted data; output parameter.

Table 111: Input Parameters

Parameter	Description
`CK_ULONG_PTR pulDataLen`	Buffer length pointer. First call: Pointer to buffer length. Buffer length is zero for metadata logging. Second call: actual buffer length used for encrypted data, written encrypted data length.

Table 112: Output Parameters

Parameter	Description
`pData`	When input parameter `pData` is NULL, this function returns a calculated output buffer length value pointed to by `pulDataLen`. For C program only.

Table 113: Return Values

Parameter	Description
`CKR_OK`	The function is executed successfully.
`CKR_ARGUMENTS_BAD`	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.
`CKR_BUFFER_TOO_SMALL`	The output of the function is too large to fit in the supplied buffer.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to `C_Initialize`.
`CKR_DATA_INVALID`	The plaintext input data to a cryptographic operation is invalid.
`CKR_DATA_LEN_RANGE`	The plaintext input data to a cryptographic operation has a bad length.
`CKR_FUNCTION_CANCELED`	The function was canceled in mid-execution.
`CKR_FUNCTION_FAILED`	The requested function could not be performed, but detailed information about why not is not available in this error return.
`CKR_GENERAL_ERROR`	An unrecoverable error has occurred.

Table 113: (Continued)Return Values

Parameter	Description
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_OPERATION_NOT_INITIALIZED	There is no active operation of an appropriate type in the specified session.
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.
CKR_USER_NOT_LOGGED_IN	The desired action cannot be performed because the appropriate user (or an appropriate user) is not logged in.
CKR_MECHANISM_INVALID	An invalid mechanism was specified to the cryptographic operation.

C_DecryptFinal

Finishes a multi-part decryption. This works only with the local key cache. Not supported if crypto is done remotely on the DSM.

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

Table 114: Input Parameters

Parameter	Description
CK_SESSION_HANDLE hSession	Session handle.
CK_BYTE_PTR pLastPart	Pointer to a buffer to write the decrypted data. First call: Pointer to the metadata to be passed and logged on the application's behalf. Must start with "META:"; input parameter. Second call: Pointer to the resulting decrypted data; output parameter

Table 114: Input Parameters

Parameter	Description
`CK_ULONG_PTR pulLastPartLen`	Buffer length pointer. First call: Pointer to decrypted buffer length. Encrypted buffer length is zero for metadata logging. Second call: actual buffer length used for decrypted data, written encrypted data length.

Table 115: Output Parameters

Parameter	Description
`pLastPart`	`pLastPart` will be filled in with the decrypted text.
`pulLastPartLen`	`pulLastPartLen` will be filled in with the length of the decrypted text.
`pLastPart`	When input parameter `pLastPart` is NULL, this function returns a calculated output buffer length value pointed to by `pulLastPartLen`. For C program only.

Table 116: Return Values

Parameter	Description
`CKR_OK`	The function is executed successfully.
`CKR_ARGUMENTS_BAD`	Generic error code which indicates that the arguments supplied to the Cryptoki function were in some way not appropriate.
`CKR_BUFFER_TOO_SMALL`	The output of the function is too large to fit in the supplied buffer.
`CKR_CRYPTOKI_NOT_INITIALIZED`	Indicates that the function cannot be executed because the Cryptoki library has not yet been initialized by a call to `C_Initialize`.
`CKR_DATA_INVALID`	The plaintext input data to a cryptographic operation is invalid.
`CKR_ENCRYPTED_DATA_INVALID`	For GCM, if the decryption can't be validated.

Table 116: (Continued)Return Values

Parameter	Description
CKR_DATA_LEN_RANGE	The plaintext input data to a cryptographic operation has a bad length.
CKR_FUNCTION_CANCELED	The function was canceled in mid-execution.
CKR_FUNCTION_FAILED	The requested function could not be performed, but detailed information about why not is not available in this error return.
CKR_GENERAL_ERROR	An unrecoverable error has occurred.
CKR_HOST_MEMORY	Insufficient memory to perform the requested function.
CKR_OPERATION_NOT_INITIALIZED	There is no active operation of an appropriate type in the specified session.
CKR_SESSION_CLOSED	The session was closed during the execution of the function.
CKR_SESSION_HANDLE_INVALID	The specified session handle was invalid at the time that the function was invoked.
CKR_USER_NOT_LOGGED_IN	The desired action cannot be performed because the appropriate user (or an appropriate user) is not logged in.
CKR_MECHANISM_INVALID	An invalid mechanism was specified to the cryptographic operation.

Random Data Generation

C_GenerateRandom

Use this function to generate and return random data.

```
CK_RV GenerateRandom(struct ctx * c, CK_SESSION_HANDLE session,
     CK_BYTE_PTR * rand, CK_ULONG length)
{
*rand = calloc(length, sizeof(CK_BYTE));
if (*rand == NULL) {
return CKR_HOST_MEMORY;
}
CK_RV e = c->sym->C_GenerateRandom(session, *rand, length);
return e;
}
```

Table 117: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_BYTE_PTR randomdata`	Receives the random data.
`CK_ULONG length`	The number of byes to generate.

Table 118: Output Parameters

Parameter	Description
`RandomResult`	A randomized token is returned.

C_SeedRandom

Generate token random numbers by mixing additional seed material into the token random number generator.

```
 CK_RV SeedRandom(struct ctx * c, CK_SESSION_HANDLE session, CK_BYTE_PTR
seed,
 CK_ULONG seedlen)
{
CK_RV e = c->sym->C_SeedRandom(session, seed, seedlen);
return e;
}
```

Table 119: Input Parameters

Parameter	Description
`CK_SESSION_HANDLE hSession`	Session handle.
`CK_BYTE_PTR pSeed`	Set the seed value.
`CK_ULONG ulSeedLen`	The seed length