HP Open Source Security for OpenVMS
Volume 1: Common Data Security Architecture

CDSA Version 2.2 for OpenVMS
based on the Intel Version 2.0 Release 3 Reference Platform

OpenVMS I64 Version 8.2 or higher
OpenVMS Alpha Version 7.3-2 or higher

This manual supersedes *HP Open Source Security for OpenVMS Common Data Security Architecture*, Version 7.3-2

Manufacturing Part Number: BA554-90006
July 2006

Legal Notice

The HP OpenVMS documentation set is available on CD-ROM.

ZK6660

Contents

Contents

Tables

Tables

Figures

Figures

Preface

Intended Audience

This document is for application developers who want to use the Common Data Security Architecture (CDSA) to add security to their programs.

This is not a tutorial manual. The reader should already have a basic understanding of fundamental cryptographic terms and principles, as well as a broad overview of CDSA services and architecture.

Document Structure

This manual consists of the following chapters:

Chapter 1 contains a broad overview of CDSA.

Chapter 2 provides important information about installation and initialization of CDSA.

Chapter 3 describes Secure Delivery for OpenVMS, a new features included in CDSA Version 2.2. Secure Delivery creates digital signatures for files, so that the file and associated manifest can be delivered over an unsecured channel such as a web download.

Chapter 4 describes administrative and development utilities provided with CDSA.

Chapter 5 includes programming information and examples of using CDSA.

Following the chapters are three reference sections: the CDSA and MDSUTIL application programming interface functions (API functions), the Elective Module Manager (EMM) API functions, and the Human Recognition Service (HRS) API functions.

Following the reference sections are an appendix containing the open source notice and a glossary.

Related Documents

The following documents are recommended for further information:

- *HP Open Source Security for OpenVMS, Volume 2: HP SSL for OpenVMS.*
- *HP Open Source Security for OpenVMS, Volume 3: Kerberos.*
- DCL Help file for the API functions. (Enter the HELP CDSA command at the DCL prompt.)
- Release Notes for CDSA. For Versions 7.2-2 and higher, the release notes for CDSA can be found in SYS$HELP:CDSA022.RELEASE_NOTES.
- Intel CDSA documents, found in SYS$COMMON:[CDSA.DOCS]:
  - *Intel Common Data Security Architecture Application Developer's Guide*: CDSA$APP_DEV_GUIDE.PDF
  - *Intel Common Data Security Architecture Service Provider Developer's Guide*: CDSA$SP_DEV_GUIDE.PDF
  - *Intel Common Data Security Architecture Manifest Signing Tools User's Guide*: CDSA$MST_GUIDE.PDF
- *CDSA Technical Standard,* available from The Open Group at the following Web site:

  `http://www.opengroup.org/onlinepubs/009609799`

- *FIPS 186 Standard,* available from the following Web site:

  `http://www.itl.nist.gov/fipspubs/fip186.htm`

For additional information about HP OpenVMS products and services, see the following World Wide Web address:

`http://www.hp.com/go/openvms`

For additional information about CDSA, visit the following Web sites:

`http://sourceforge.net/projects/cdsa`
`http://www.intel.com/labs/archive/cdsa.htm`

Reader's Comments

HP welcomes your comments on this manual.

Please send comments to either of the following addresses::

Internet: openvmsdoc@hp.com

Postal Mail:
Hewlett-Packard Company
OSSG Documentation Group
ZKO3-4/U08
110 Spit Brook Road
Nashua, NH 03062-2698

How to Order Additional Documentation

For information about how to order additional documentation, visit the following World Wide Web address :

`http://www.hp.com/go/openvms/doc/order`

Conventions

The following conventions may be used in this manual:

Convention	Meaning
Ctrl/x	A sequence such as Ctrl/x indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 x	A sequence such as PF1 *x* indicates that you must first press and release the key labeled PF1 and then press and release another key (x) or a pointing device button.
Return	In examples, a key name in bold indicates that you press that key.
...	A horizontal ellipsis in examples indicates one of the following possibilities: – Additional optional arguments in a statement have been omitted. – The preceding item or items can be repeated one or more times. – Additional parameters, values, or other information can be entered.

Convention	Meaning
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In command format descriptions, parentheses indicate that you must enclose choices in parentheses if you specify more than one.
[ ]	In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement.
|	In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are optional; within braces, at least one choice is required. Do not type the vertical bars on the command line.
{ }	In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line.
bold type	Bold type represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason.
italic type	Italic type indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where (*dd*) represents the predefined par code for the device type).
UPPERCASE TYPE	Uppercase type indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
`Example`	This typeface indicates code examples, command examples, and interactive screen displays. In text, this type also identifies URLs, UNIX command and pathnames, PC-based commands and folders, and certain elements of the C programming language.
–	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes — binary, octal, or hexadecimal — are explicitly indicated.

1 Introduction to CDSA

This chapter provides an overview of key components of the Common Data Security Architecture (CDSA) and its set of integrity services.

1.1 What Is CDSA?

The Common Data Security Architecture (CDSA) is a multiplatform, industry-standard security infrastructure. Starting with Version 7.3-1, HP provides CDSA as part of the OpenVMS operating system. CDSA is compatible with OpenVMS Alpha Version 7.2-2 and higher, and OpenVMS I64 Version 8.2 and higher.

CDSA provides a stable, standards-based programming interface that enables applications to access operating system security services. With CDSA, you can create cross-platform, security-enabled applications. Security services, such as cryptography and other public key operations, are available through a dynamically extensible interface to a set of add-in modules. These modules can be supplemented or changed as business needs and technologies evolve.

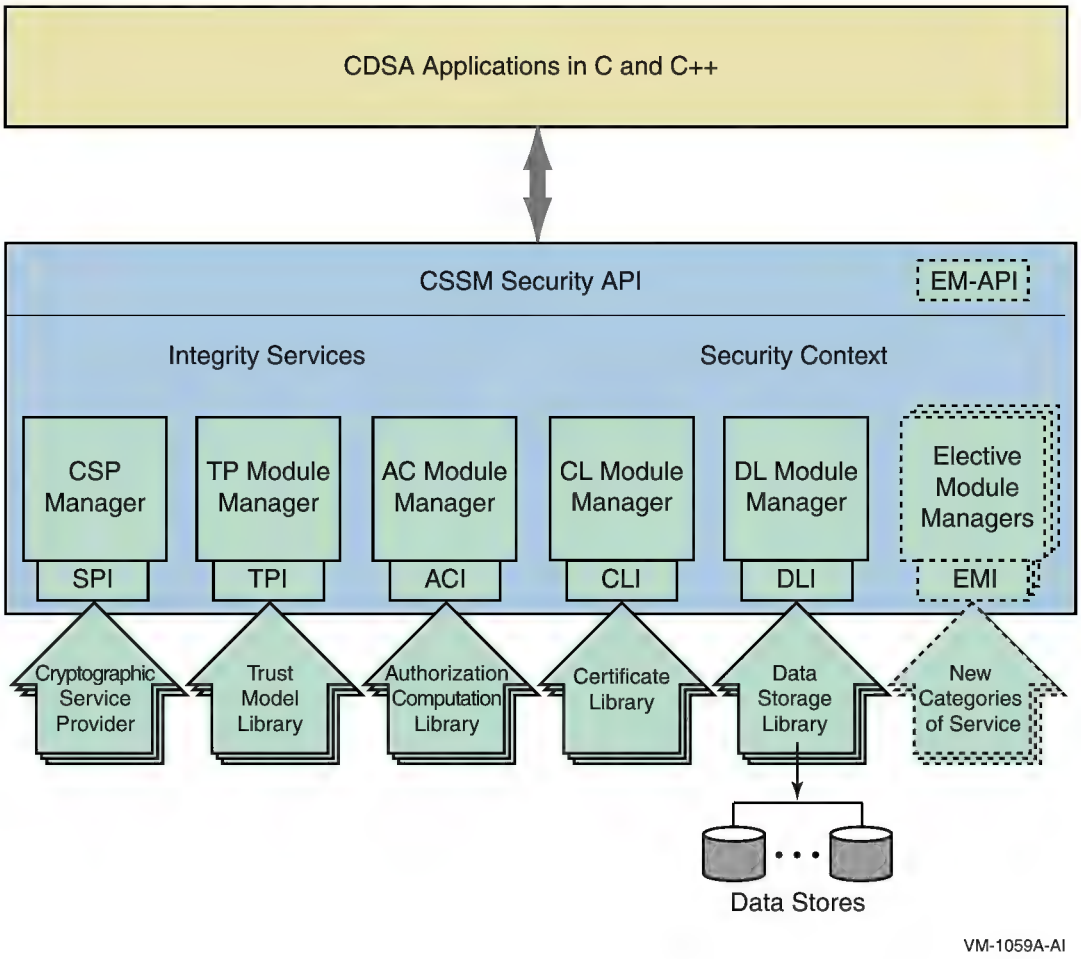
CDSA is security middleware that provides flexible mix-and-match solutions across a variety of applications and security services. CDSA insulates you from the issues of incorporating security into applications, freeing you to focus on the applications themselves. The security underpinnings are transparent to the user.

CDSA was originally developed by Intel® Architecture Labs and was released to the OpenSource community in May 2000. HP's CDSA implementation is based on the Intel V2.0 Release 3 reference platform, which implements CDSA V2.0 with Corrigenda, as defined in The Open Group's Technical Standard C914, May 2000.

1.2 CDSA Overview

The CDSA layered architecture is shown in Figure 1-1 on page 16.

Figure 1-1 CDSA Layered Architecture



Applications call the Common Security Services Manager (CSSM), which implements the CDSA APIs. The CSSM also implements the CDSA integrity services and security contexts. The managers for each of the CDSA add-ins are also part of CSSM. CSSM is described in more detail in Section 1.2.1.

In addition to the CSSM, CDSA includes the following:

- Service provider modules - See Section 1.2.2
- Elective module Managers (EMMs) - See Section 1.2.3
- Module directory Services (MDS) - See Section 1.2.4

Chapter 5, "CDSA Programming Concepts," on page 49 provides sample C programs that illustrate the use of CDSA.

For additional information about CDSA, see the web links listed in the Preface.

1.2.1 Common Security Services Manager (CSSM)

The Common Security Services Manager (CSSM) is the heart of CDSA. It is a shared library (in SYS$SHARE:CDSA$INCSSM300_SHR.EXE) to which applications can link to obtain security services. It defines both the API and the service provider interface (SPI) for add-in security service modules. CSSM includes a set of core services that are common to all categories of security services. These services perform functions such as:

- Dynamic attach of an add-in security module
- Enforced integrity, authentication, and exemption verification when dynamically attaching services
- Secure linkage checks on calls to service provider modules
- General integrity services

Applications call functions in the CSSM API, which is fully specified by the *CDSA Technical Standard* (located at http://www.opengroup.org/onlinepubs/009609799/). API function names are prefaced with CSSM_ and are sometimes followed by the designation of the module that will actually handle the request. For instance, applications call `CSSM_DL_DbOpen()` to direct a DL module to open a data store. The associated SPI for this module is `DL_DbOpen()`. (The SPI interface is not directly callable by CDSA applications.)

An application begins by initializing its connection to CSSM using the `CSSM_Init()` routine. The application can use Module Directory Services (MDS) to inquire about available modules and their supported functionality (see an MDS example in Section 5.4.3) or it can directly access a specific service provider by using its global unique identifier (GUID). The application loads the desired module using the `CSSM_ModuleLoad()` routine and then attaches to it using the `CSSM_ModuleAttach()` routine.

The CSSM is implemented as a sharable image on OpenVMS. Header files (in CDSA_SYSDIR:[INCLUDES]*.H) define the CSSM API.

1.2.2 Service Provider Modules

There are several types of add-ins for CDSA, each supporting a different security task:

- Cryptographic Service Provider (CSP) modules (see Section 1.2.2.1)
- Trust Policy (TP) modules (see Section 1.2.2.2)
- Authorization Computation (AC) modules (see Section 1.2.2.3)
- Certificate Library (CL) modules (see Section 1.2.2.4)
- Data Storage Library (DL) modules (see Section 1.2.2.5)

On OpenVMS, service providers are implemented as sharable images.

1.2.2.1 Cryptographic Service Providers (CSPs)

The Cryptographic Service Providers (CSPs) are add-in modules to the Common Security Services Manager (CSSM). CSPs perform cryptographic operations and securely store cryptographic keys for the applications that call them through the CSSM API. A CSP can be in the form of software, hardware, or both.

Applications call these CSPs to provide authentication, data integrity, data and communication privacy, and nonrepudiation of messages to users.

CSPs implement the following cryptographic algorithms, among others, in one or more modes:

- Bulk encryption algorithm in modes AES, DES, Triple DES, DESX, RC2, RC4, and RC5
- Digital signature algorithm in modes RSA and DSS

- Key negotiation algorithm in modes Diffie-Hellman and DSA
- Cryptographic hash algorithm in modes MD4, MD5, and SHA1

CSPs also provide the following services:

- Unique identification number: hard coded or random generated
- Random number generator: attended and unattended
- Encrypted data: symmetric keys and private keys
- Secure key storage
- Custom facilities unique to the CSP

The CSP module manager administers the CSPs that are installed on the local system. It defines a common API to access all of the Cryptographic Service Providers that can be attached and used by any caller in the system.

The specific security services API functions that are defined by the CSP module manager include the following service categories:

SignData
VerifyData
DigestData
EncryptData
DecryptData
GenerateKeyPair
GenerateRandom
WrapKey
UnwrapKey

CDSA on OpenVMS provides CSPs based on OpenSSL and RSA BSAFE:

- OpenSSL CSP
  - — Message authentication based on MD5 and SHA1
  - — Symmetric encryption based on DES, Triple DES, and AES
- RSA BSAFE CSP
  - — Message authentication based on MD5 and SHA1
  - — Symmetric encryption based on DES, triple DES, DESX, and RC2, RC4, and RC5.
  - — Asymmetric encryption based on RSA, DSA, and Diffie-Hellman

The following sections discuss these topics:

- Establishing a session to use a CSP (see Section 1.2.2.1.1)
- Defining the security context (see Section 1.2.2.1.2)
- Using keys (see Section 1.2.2.1.3)

1.2.2.1.1 Establishing a Session An application establishes a session to select a particular CSP. Once attached, the application can initiate a cryptographic login session with the CSP. The application requests additional credentials, such as a passphrase or PIN, to gain access to specific keys and services managed by the CSP.

Within a module attach session or a cryptographic login session, an application creates, uses, and discards cryptographic contexts. A cryptographic context carries the parameters required to perform a cryptographic service. The cryptographic context can be used for the following:

- A one-step cryptographic operation in which only one call is needed to obtain the result.
- A cryptographic session of a multistaged cryptographic service, in which an initialization call is followed by one or more update calls, ending with a completion (final) call. For most cryptographic operations, the result is available after the final function completes its execution. An exception is staged encryption/decryption, in which each update call generates a portion of the result.

Depending on the class of cryptographic operations, individualized attributes are available for the cryptographic context. In addition to specifying an algorithm when creating the context, the application can also initialize a session key, pass an initialization vector, or pass padding information to complete the description of the session. A successful return value from the create function indicates that the desired CSP is available.

Functions are also provided to manage the created context. The cryptographic context contains most or all of the input parameters required for an operation. Some cryptographic service functions accept input parameters in addition to the CSP handle and the context handle. These input parameters always take precedence over any duplicate or conflicting parameters in the cryptographic context. When a context is no longer required, the application calls a DeleteContext function. Resources allocated for that context can then be reclaimed by the operating system.

1.2.2.1.2 Defining a Security Context The application's associated security context defines parameter values for the low-level variables that control the details of cryptographic operations. For example, an application issuing a request to the EncryptData call can reference a security context that defines the following parameters:

- The algorithm to be used (such as DES)
- Algorithm-specific parameters (such as key length)
- The object on which the operation is conducted (such as a set of buffers)
- The cryptographic variables (such as the key)

Most applications use predefined, default contexts. Typically, a distinct context is used for encrypting, hashing, and signing. For an initialized application, these contexts change little, if at all, during the application's execution or between executions. This allows the application developer to implement security by manipulating certificates, using previously defined security contexts, and maintaining a high-level view of security operations.

1.2.2.1.3 Using Keys In CDSA, there are two main types of cryptographic algorithms that use keys:

- **Asymmetric algorithms** use one key to encrypt and a second key to decrypt. They are often called public-key algorithms. One key is called the public key and the other is called the private key or secret key. RSA (Rivest-Shamir-Adelman) is the most commonly used public-key algorithm. It can be used for encryption and for signing.
- **Symmetric algorithms** use a single secret key for encryption and decryption. Both the sender and receiver must know the secret key. Well-known symmetric functions include DES (Data Encryption Standard) and IDEA. DES was endorsed by the U.S. Government as a standard in 1977. It's an encryption block cipher that operates on 64-bit blocks with a 56-bit key. It is designed to be implemented in hardware, and works well for bulk encryption. IDEA (International Data Encryption Algorithm) uses a 128-bit key.

Every CSP implements its own secure, persistent storage and management of private keys. To support chains of trust across application domains, CSPs support importing and exporting of public and private keys among remote and possibly foreign systems. To transfer keys, the CSP must be able to convert one key format into any other key format and to secure the transfer of private and symmetric keys.

Each CSP is responsible for securely storing the private keys it generates or imports from other sources. Additional storage-related operations include retrieving a private key when given its corresponding public key and wrapping private keys as key blobs for secure exportation to other systems.

On an OpenVMS Alpha system, the CSP stores private key files in EAYCSP.PRI and MAF_BSAFE.PRI. The protections on the key files are OWNER:READ,WRITE,DELETE. The key files are user-specific and are stored in the [.CDSA.PKD] subdirectory in the user's login directory.

Public Key Infrastructure (PKI)

The Public Key Infrastructure (PKI) is the state-of-the-art method, ultimately to be applied worldwide, for secure and confidential electronic transactions. It employs public and private keys.

The two PKI algorithms in widespread use are:

- RSA-based algorithms
- DSA-based algorithms

For RSA-based algorithms, CDSA uses the PKCS#1 standard for key representation. For DSA-based algorithms, no organization has published a standard. CDSA's representation of the DSA key is based on the DSA algorithm definitions in the FIPS 186 standard. (See the Preface for web links to this and other standards.)

A DSA public key is represented as a BER-encoding of a sequence list that contains the following:

```
PrimeModulus; /* p */
PrimeDivisor; /* q */
OrderQ; /* g */
PublicKey; /* y */
```

A DSA private key is represented as a BER-encoded sequence list that contains the following:

```
PrimeModulus; /* p */
PrimeDivisor; /* q */
OrderQ; /* g */
PrivateKey; /* x */
```

These key components are defined as follows by FIPS 186 and FIPS 186a:

- PrimeModulus. This is the public prime modulus.

  p = A prime modulus, where $2^{L-1} < p < 2^{L}$ for 512 <= L <= 1024, and L is a multiple of 64.

- PrimeDivisor. Another public prime number dividing (p-1).

  q = A prime divisor of p-1, where $2^{159} < q < 2^{160}$

- OrderQ. This public number has order q mod p.

  g = $h^{(p-1)/q}$ mod p, where h is any integer with 1 < h < p-1, such that $h^{(p-1)}$ /q mod p > 1.

- PrivateKey. The private key.

  x = a pseudorandomly generated integer with 0 < x < q.

- PublicKey. The public key.

  y = g^{x} mod p.

A DSA-wrapped private key is defined by the PKCS#8 specification. The PKCS#8 standard specifies the wrapped key format resulting from encoding an algorithm object identifier (OID) with an encoded private key.

1.2.2.2 Trust Policy (TP) Modules

Trust Policy modules allow applications to request security services that require "policy review and approval" as the first step in performing the operation. Approval can be based on the identity, integrity, and authorization represented in a group of digital certificates.

Trust Policy modules implement policies defined by authorities and institutions. Policies define the level of trust required before certain actions can be performed. Three basic action categories exist for all certificate-based trust domains:

- Actions on certificates
- Actions on certificate revocation lists
- Domain-specific actions (such as issuing a check or writing a file)

The Trust Policy function can invoke certificate and data storage library functions to carry out the mechanics of the approved action.

1.2.2.3 Authorization Computation (AC) Modules

Authorization Computation modules define a general authorization evaluation service that computes whether a set of credentials and samples are authorized to perform a specific operation on a specific object. AC modules implement an authorization evaluation mechanism based on caller inputs. Callers provide:

- The assumptions forming the basis of the caller's policy
- The request for which authorization is being checked
- The credentials, samples, and exhibits that could demonstrate authorization to perform the request

The Authorization Computation engine determines whether the request is authorized based on the assumptions and caller credentials. The AC module can provide other services related to authorization computations through the `CSSM_AC_PassThrough()` function.

1.2.2.4 Certificate Library (CL) Modules

The Certificate Library API allows applications to manipulate memory-resident certificates and certificate revocation lists. Operations must include creating, signing, verifying, and extracting field values from certificates. Each add-in certificate library incorporates knowledge of certificate data formats, and how to manipulate that format.

The CSSM Certificate API defines the generic operations that should be supported by every CL module. Each module can choose to implement only those operations required to manipulate a specific certificate data format, such as X.509, SDSI, etc.

The implementation of these operations is intended to be semantic-free. Semantic interpretation of certificate values is designed to be implemented in Trust Policy modules, layered services, and applications.

The Certificate Library module provided on OpenVMS systems can manipulate X509V3 certificates and SPKI (Simple Public Key Infrastructure) certificates.

1.2.2.5 Data Storage Library (DL) Modules

The Data Storage Library allows applications to search and select stored data objects, and to query meta-information about each data store (such as its name, date of last modification, size of the data store, and so on).

Data Storage Library modules provide stable storage for security-related data objects. These objects can be certificates, certificate revocation lists, cryptographic keys, integrity and authentication credentials, policy objects, or application-specific objects. Stable storage can be provided by one of the following:

- Commercially-available database management system product
- Native file system
- Custom hardware-based storage device
- Remote directory services (e.g., LDAP)
- In-memory storage

Each Data Storage Library module can choose to implement only those operations required to provide persistence under its selected model of service.

The Data Storage Library module currently provided on OpenVMS uses OpenVMS flat files.

1.2.3 Elective Module Managers (EMMs)

The CDSA architecture includes several extensibility mechanisms. Elective module managers support the dynamic addition of entire new categories of service. Prior to requesting services from an add-in service provider module, the application attaches to an instance of the service provider. For elective module managers, the CSSM transparently attaches the associated module manager if it is not already loaded. Once the manager is loaded, the APIs defined by that module are available to the application.

This process is transparent to the add-in module as well as to the application. Therefore, an add-in module vendor should not need to modify their module implementation to work with an elective module manager versus a basic module manager.

1.2.4 Module Directory Services (MDS)

The Module Directory Services provide facilities to describe and locate executable objects and their associated signed manifest integrity credentials.

MDS consists of a database and a set of access methods. It is used primarily to support secure loading and the use of add-in software modules. It is a system-wide service available to all processes. MDS defines a basic object directory schema to name and locate software components and the signed manifest credentials associated with those software components. Each software component in the object directory is uniquely named by a globally unique identifier (GUID). CDSA defines an additional set of schemas to store CDSA-specific security attributes of all CDSA components. CDSA components use the MDS-managed data to do the following:

- Discover other available CDSA components
- Learn about the capabilities and properties of other CDSA components
- Locate the executables for CDSA components
- Locate the signed manifest credentials associated with a CDSA software component

New schemas can be defined to store the properties and capabilities of elective CDSA modules as they are defined. CDSA applications can also define MDS schemas and use MDS services. CDSA components use MDS managed data to support CDSA's software authentication and integrity checking procedure, known as bilateral authentication.

Chapter 5, “CDSA Programming Concepts,” on page 49 provides an example of how to use MDS.

1.3 Maintaining CDSA Integrity

As the foundation of the security framework, CSSM provides a set of integrity services that can be used by CSSM, module managers, add-in modules, and applications to verify their own integrity, and the integrity, identity, and authorizations of other components in the CDSA environment.

CSSM's set of self-contained security services establishes a security perimeter around CDSA. These services incorporate techniques to protect against malicious attacks. Because application and add-in security service modules are dynamic components in the system, CSSM uses and requires the use of a strong verification mechanism to screen all components as they are added to the CSSM environment.

Applications can extend CSSM's security perimeter to include themselves by using bilateral authentication, integrity verification, and authorization checks during dynamic binding.

The establishment of integrity between two dynamically loaded, executable objects proceeds in three phases:

- Self-check
- Bilateral authentication
- Secure linkage check

1.3.1 Self-Check

In the first phase, the self-check phase, the software module checks its own digital signature. The Embedded Integrity Services Library (EISL) defines a statically linked library procedure to perform self-check.

1.3.2 Bilateral Authentication

In the second phase, bilateral authentication routines in the EISL offer support for securely loading, verifying, and linking to partner software modules. The process of bilateral authentication begins in the MDS registry, where each program can find the credentials as well as the object code of all other CDSA modules.

Verification of other modules can be done prior to loading, or, if a module is already loaded, it can be verified in memory. Verification prior to loading prevents activating file viruses in infected modules. Verification in memory prevents stealth viral attacks where the file is healthy, but the loaded code is infected.

1.3.3 Secure Linkage Check

Once verified, programs can use the verified in-memory representation of the credentials to perform validity checks of addresses to provide secure linkage to modules. The addresses of both the callers and the procedures to be called can be verified using the Secure Linkage Check facility.

2 Installation and Initialization

This chapter provides important information about CDSA installation and initialization.

NOTE You must have the SYSPRV and CMKRNL privileges to initialize CDSA. Users of CDSA applications do not need SYSPRV, but you will likely need SYSPRV to develop CDSA signed applications and plugins

Table 2-1 lists the currently supported versions of CDSA, and the installation and configuration requirements for the versions of OpenVMS that support CDSA.

Table 2-1 CDSA Installation and Configuration Summary

OpenVMS Version	CDSA Version
Version 8.3 and higher	1. CDSA is automatically installed and configured.
Version 8.2	1. CDSA Version 2.1 is automatically installed. 2. Execute this command to configure CDSA: @SYS$STARTUP:CDSA$UPGRADE
Version 7.3-2	1. CDSA Version 2.0 is automatically installed. 2. Execute this command to configure CDSA: @SYS$STARTUP:CDSA$UPGRADE

NOTE CDSA Version 1.0 is not supported on OpenVMS Version 7.3-2 or higher.

2.1 Installation of CDSA on OpenVMS Alpha and I64 Version 8.3 and higher

If you install or upgrade to OpenVMS I64 or OpenVMS Alpha Version 8.3 or higher, CDSA Version 2.2 is automatically installed and configured.

2.2 Installation of CDSA on OpenVMS Alpha and I64 Version 8.2

If you install or upgrade to OpenVMS I64 or OpenVMS Alpha Version 8.2, CDSA Version 2.1 is automatically installed. Before you can use CDSA Version 2.1, however, you must execute the following command to initialize CDSA:

```
$ @SYS$STARTUP:CDSA$UPGRADE
```

Note that you must have the SYSPRV and CMKRNL privileges to execute this procedure. This command automatically calls `CDSA$INITIALIZE()`.

2.3 Installation of CDSA on OpenVMS Alpha Version 7.3-2

If you install or upgrade to OpenVMS Alpha Version 7.3-2, CDSA Version 2.0 is automatically installed. Before you can use CDSA Version 2.0, however, you must execute the following command to initialize CDSA:

```
$ @SYS$STARTUP:CDSA$UPGRADE
```

Note that you must have the SYSPRV and CMKRNL privileges to execute this procedure. This command automatically calls `CDSA$INITIALIZE()`.

2.4 Installation of CDSA Version 2.2 on OpenVMS Versions Earlier than Version 8.3

Although older versions of CDSA are pre-installed on OpenVMS Version 7.3-2 and Version 8.2, you can install and run CDSA Version 2.2.

2.4.1 CDSA Version 2.2 Setup and Initialization

If you want to run CDSA Version 2.2 on OpenVMS Version 8.2 or 7.3-2, you must manually install the CDSA Version 2.2 kit, which is included on the OpenVMS Version 8.3 media. Use the following command to install CDSA Version 2.2 on an OpenVMS Version 8.2 or 7.3-2 system:

```
$ PRODUCT INSTALL CDSA /SOURCE=disk:[directory]

The following product has been selected:
        CPQ AXPVMS CDSA V2.2            Layered Product

Do you want to continue? [YES]

Configuration phase starting ...

You will be asked to choose options, if any, for each selected product and for
any products that may be installed to satisfy software dependency requirements.

CPQ AXPVMS CDSA V2.2
```

```
Do you want the defaults for all options? [YES]

Do you want to review the options? [NO]

Execution phase starting ...

The following product will be installed to destination:
       CPQ AXPVMS CDSA V2.2             DISK$SYSTEM:[VMS$COMMON.]

Portion done:
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%

The following product has been installed:
       CPQ AXPVMS CDSA V2.2             Layered Product

CPQ AXPVMS CDSA V2.2
$
```

Before you can use CDSA Version 2.2. you must perform the following manual procedure, for which you must have SYSPRV privileges. Execute the following command to initialize CDSA Version 2.0:

```
$ @SYS$STARTUP:CDSA$UPGRADE
```

This procedure automatically runs CDSA$INITIALIZE. It is not necessary to rerun any initialization procedure when the system is rebooted; therefore, you do not need to add the initialization to the OpenVMS startup procedures.

The CDSA$UPGRADE procedure can take a few minutes, depending on your processor and disk speeds. When the procedure is run interactively, you will see system messages similar to the following:

```
CDSA-I-Init, CDSA has previously been initialized on this system.
CDSA-I-Init, Re-initializing CDSA.

CDSA-I-Init, Initializing CDSA
MDS installed successfully.
Module uninstalled successfully.
Module uninstalled successfully.
Module uninstalled successfully.
Module uninstalled successfully.
Module uninstalled successfully.
Module uninstalled successfully.
Module uninstalled successfully.
Module uninstalled successfully.
CDSA-I-Init, CDSA Initialization complete

CDSA-I-Init, Initializing Secure Delivery
Install completed successfully.
Install completed successfully.
Module installed successfully.
Module installed successfully.
CDSA-I-Init, Secure Delivery Initialization complete
$
```

2.4.2 Warning Against Uninstalling CDSA from OpenVMS Alpha Version 7.3-1 or Higher

The POLYCENTER Software Installation utility command PRODUCT REMOVE is not supported for CDSA on OpenVMS Alpha Version 7.3-1 or higher, even though there is an apparent option to remove CDSA. (This option is due to the use of the POLYCENTER Software Installation utility in the installation.) CDSA is installed together with the operating system and is tightly bound with it. An attempt to remove it from Version 7.3-1 or higher would not work cleanly and could create other undesirable side effects. An attempt to remove CDSA results in the following message:

```
%PCSI-E-HRDREF, product CPQ AXPVMS CDSA Vx.x is
referenced by DEC AXPVMS OPENVMS V7.3-2
-PCSI-E-HRDRF1, the two products are tightly bound
by this software dependency
```

2.5 Post-Installation Tasks

Once you have installed CDSA, you should perform the tasks described in the following sections.

2.5.1 Defining CDSA Symbols

To define symbols for CDSA developers, add the following command to the SYS$MANAGER:SYLOGIN.COM file on the system where CDSA development work is being done:

```
$ @SYS$MANAGER:CDSA$SYMBOLS.COM
```

NOTE The file SYS$MANAGER:CDSA$SYMBOLS.COM does not exist for CDSA Version 1.0, so it is not present on an OpenVMS Version 7.3-1 system unless a CDSA Version 2.0 or higher kit has subsequently been installed.

If this command is not defined at the system level in SYLOGIN.COM, individual CDSA developers should add it to their personal LOGIN.COM file so that they can use the symbols.

2.5.2 Backing up the CDSA Database

HP recommends that you back up the CDSA database and registry files on a regular basis and when any major changes to the data are planned. For example:

```
$ BACKUP CDSA_SYSDIR:[CDSAFFDB]*.* -
_$ disk:[directory...]CDSA_DB_BACKUP.BCK/SAV
$ BACKUP CDSA_SYSDIR:[REGISTRY…]*.* -
_$ disk:[directory...]CDSA_REGISTRY_BACKUP.BCK/SAV
```

3 Secure Delivery

3.1 Introduction

This chapter provides an overview of Secure Delivery on OpenVMS and describes how to invoke its components using CDSA. Secure Delivery creates digital signatures for files, so that the file and associated manifest can be delivered over an unsecured channel such as a web download.

Support for Secure Delivery is included in CDSA beginning with OpenVMS Version 8.3.

NOTE Kits included on the OpenVMS Version 8.3 distribution media are signed using Secure Delivery. On OpenVMS I64, SIP (System Integrated Product) or layered product kits that are installed **during or after** the OpenVMS upgrade are validated. On OpenVMS Alpha, only SIP or layered product kits that are installed **after** the OpenVMS upgrade are validated.

Kits created before the secure delivery process was enabled in OpenVMS Version 8.3 can be installed on OpenVMS Version 8.3. These kits are marked as unsigned, rather than as a validated kit in the PCSI history file. Products installed before Version 8.3 have a blank validation status in the PCSI history.

For more information, see Section 3.2.

Secure Delivery uses public key and digital signature technology to implement a system that provides OpenVMS users with the ability to authenticate and validate the files they download from OpenVMS and third-party OpenVMS vendors.

Secure Delivery enhances CDSA by creating a **manifest** of a target file so that the file and its accompanying manifest can be delivered together over an unsecured Internet link or media format, such as a CD or DVD. After the files are in place on the target system, the manifest can be used to authenticate the originator and validate the contents of the target file. If the target file (or the manifest) has been tampered with in any way, the validation process will fail. If the certificates used to sign the file have been revoked, the validation will fail.

See the Glossary for definitions of terms used in this chapter.

3.2 PCSI and Secure Delivery

The POLYCENTER Software Installation utility (PCSI) is a software installation and management tool for OpenVMS systems. It can package, install, remove, and manage software products. It can also save information about software products such as system requirements and installation options.

Beginning in OpenVMS Version 8.3, PCSI checks for the existence of a manifest for kits that are being installed. If a manifest is not found, PCSI issues a warning and asks whether or not to proceed. If a manifest is found but does not match the kit, the installation is ended. The PCSI database contains an indication as to whether a kit used Secure Delivery on installation. For more information, see Section 3.2.1.

The PCSI utility validates kits (when a manifest is present in the source directory) for the following commands:

PRODUCT CONFIGURE
PRODUCT COPY
PRODUCT EXTRACT {FILE | PDF | PTF | RELEASE_NOTES}
PRODUCT INSTALL
PRODUCT LIST
PRODUCT RECONFIGURE
PRODUCT REGISTER PRODUCT

In OpenVMS Version 8.3, kit validation checking can be turned off by specifying the /OPTIONS=NOVALIDATE_KIT qualifier to the PRODUCT command.

For more information about PCSI, see the *System Management Utilities Reference Manual: M-Z* and the *POLYCENTER Software Installation Utility Developer's Guide*.

Examples 3-1, 3-2, 3-3. and 3-4 show validation output from the PRODUCT INSTALL command.

Example 3-1 Valid Manifest

```
$ PRODUCT INSTALL *

Performing product kit validation …

%PCSI-I-VALPASSED, validation of HP-I64VMS-TEST_THIS-0100--1.PCSI$COMPRESSED;1 succeeded
%PCSI-I-VALPASSED, validation of HP-I64VMS-TEST_THAT-0200--1.PCSI$COMPRESSED;1 succeeded

The following products have been selected:

HP-I64VMS-TEST_THIS V1.0                Layered Product
HP-I64VMS-TEST_THAT V2.0                Layered Product

Do you want to continue? [YES]
```

Example 3-2 Unsigned Kit

```
$ PRODUCT INSTALL *

%PCSI-I-CANNOTVAL, cannot validate HP-I64VMS-COBOL-0100--1.PCSI;1
-PCSI-I-NOTSIGNED, product kit was created without an associated manifest
%PCSI-I-CANNOTVAL, cannot validate HP-I64VMS-FORTRAN-0200--1.PCSI$COMPRESSED;1
-PCSI-I-NOTSIGNED, product kit was created without an associated manifest

The following products have been selected:

HP-I64VMS-COBOL V1.0  Layered Product
HP-I64VMS-FORTRAN V2.0  Layered Product

Do you want to continue? [YES]
```

Example 3-3 Missing Manifest

```
$ PRODUCT INSTALL TEST
...
%PCSI-W-NOVALDONE, cannot validate HP-I64VMS-TEST-0100--1.PCSI$COMPRESSED;1
-PCSI-W-NOMANFILE, associated manifest file was not found in source directory

Do you want to continue? [NO]
```

Example 3-4 Invalid Manifest

```
$ PRODUCT INSTALL TEST

Performing product kit validation …
%PCSI-E-VALFAILED, validation of PCSIBX$DKA0:[KRYCKA.SD]HP-I64VMS-TEST-0100--1.PCSI$COMPRESSED;1 failed
-PCSI-E-CDSA_TEXT, CSSM_ERRCODE_MODULE_MANIFEST_VERIFY_FAILED:
Modules manifest verification failed
%PCSI-E-S-OPFAIL, operation failed
%PCSIUI-E-ABORT, operation terminated due to an unrecoverable error condition
$
```

3.2.1 PCSI History File (Product Database)

The product database, or history file, is a set of binary files located in SYS$SYSDEVICE:[VMS$COMMON] with a .PCSI$DATABASE file extension.

The history file is the single source of information about operations performed on products that use PCSI. This information includes a history of operations performed, which products are installed, which files and other managed objects are owned by each product, software dependencies among products, and so forth.

The PCSI history file uses the following codes relating to Secure Delivery:

Val	Kit passed validation
Sys	Kit installed from Operating System media
(U)	Unsigned kit, not validated
(M)	Kit marked as signed, but no manifest found
(D)	Validation disabled by user
(C)	CDSA not loaded, unable to validate

Example 3-4 shows a partial output of a PCSI history file.

Example 3-5 PCSI History File (Partial Output)

```
PRODUCT                          KITTYPE Operation VAL   DATE
-------------------------------- ------- -------- --- -----------
HP I64VMS C S7.1-13              Full LP  Install (U) 03-NOV-2005
HP I64VMS CDSA T2.2-117          Full LP  Install Val 25-OCT-2005
HP I64VMS DECNET_PHASE_IV V8.3-B1B Full LP  Install Val 25-OCT-2005
HP I64VMS DWMOTIF_SUPPORT V8.3-B1B Full LP  Install Val 25-OCT-2005
HP I64VMS OPENVMS V8.3-B1B       Platform Install Val 25-OCT-2005
HP I64VMS VMS V8.3-B1B           Oper Sys Install Sys 25-OCT-2005
HP I64VMS CDSA V2.1-355          Full LP  Remove   -  25-OCT-2005
HP I64VMS DECNET_PHASE_IV V8.3-AX0 Full LP  Remove   -  25-OCT-2005
HP I64VMS DWMOTIF_SUPPORT V8.3-AX0 Full LP  Remove   -  25-OCT-2005
HP I64VMS OPENVMS V8.3-AX0       Platform Remove   -  25-OCT-2005
HP I64VMS VMS V8.3-AX0           Oper Sys Remove   -  25-OCT-2005
HP I64VMS BLISSI64 V1.12-67      Full LP  Install (U) 08-AUG-2005
...
HP I64VMS TCPIP V5.5-11          Full LP  Install     17-MAY-2005
HP I64VMS TDC_RT V2.1-69         Full LP  Install     17-MAY-2005
HP I64VMS VMS V8.2               Oper Sys Install     17-MAY-2005
-------------------------------- -------- ------- --- -----------
```

3.3 Fundamentals of Secure Delivery

The following sections discuss the fundamental parts of Secure Delivery, including CDSA architecture, the certificate, the manifest, and validation routines.

3.3.1 CDSA Architecture

Secure Delivery is built on the Common Data Security Architecture (CDSA), which is a multilayered security infrastructure that provides an integrated and dynamic set of security services to applications. CDSA provides a secure execution environment using two mechanisms, bilateral authentication and secure linkage.

3.3.1.1 Bilateral Authentication

CDSA checks the integrity of CDSA modules as they are dynamically loaded into the CDSA environment. A bilateral authentication procedure is designed for two entities to establish trust in the identity and integrity of each other. When loading a service provider module CDSA requires that the attaching party participate in this authentication protocol. If authentication fails, the module is denied the ability to be used by CDSA. Both parties in the bilateral authentication procedure must have signed credentials that bind them to the trust hierarchy used by CDSA.

Bilateral authentication can also be performed between applications and the CDSA. The only difference is that the application takes on the role of the initiator and verifies CDSA before loading and using it. Secure Delivery is an application that performs bilateral authentication.

3.3.1.2 Secure Linkage

For a CDSA application or CDSA itself, Secure Linkage checks that the address called is actually in the code module of the shareable image. For the called component, the return address must be verified as being within the calling module.

For the purpose of Secure Delivery, Secure Linkage is not of interest.

3.3.2 The Certificate

CDSA provides tools to generate X509 certificates. These tools are invoked along with additional features but the format of the certificates remains the same. For information about generating CDSA certificates, see Section 5.2.

3.3.3 The Manifest

CDSA also provides a tool to create a digital signature using the X509 certificates. The digital signature takes the form of a separate file called a manifest. The manifest contains the encrypted digest of the target file and the X509 certificates of the signers. This data is sufficient to guarantee the identity of the signer of a file and the authenticity of the file's contents.

The manifest is the key part of the mechanism that is used for bilateral authentication. It is the signed credential that each component must have to carry out the bilateral authentication.

When software kits are built, a manifest should be generated for each kit. This is the signing process. When Secure Delivery is started, the accompanying manifest is used to accomplish the bilateral authentication. This is the validation process.

3.3.4 CDSA Secure Delivery Programs

Table 3-1 lists the CDSA programs that implement Secure Delivery.

Table 3-1 CDSA Secure Delivery Programs

CDSA Program	Function
CDSA$SD_SIGN.COM	Generates manifests. See Section 3.4.2.
CDSA$REVOKE.EXE	Revokes a certificate. See Section 4.6.
CDSA$VALIDATE.EXE (new in V2.2)	Checks manifests. See Section 4.8.
CDSA$VALIDATE_LIBSHR.EXE (A CDSA_FileValidate API is implemented in CDSA$VALIDATE_LIBSHR.EXE)	Validates files programmatically. See Section 3.5.2.

NOTE Validation programs are CDSA signed applications and are mutually authenticated with the rest of CDSA to prevent tampering.

3.4 Creating Manifests

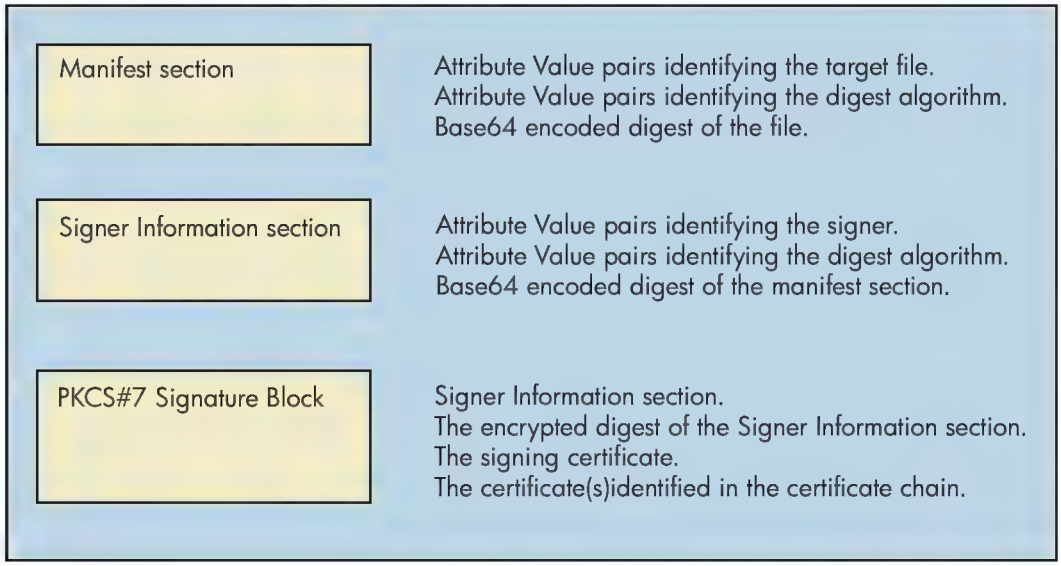
Secure Delivery provides a number of features to help with the creation of manifests. To create a manifest, a vendor must first create a self-signed certificate using CDSA. This certificate provides essential information about the party that can be integrated into all of the manifests that are created. After the self-signed certificates are created, they must be signed by the Certificate Authority (CA). OpenVMS is currently the CA for CDSA/Secure Delivery on OpenVMS. The signed certificates returned from the CA can then be used to create manifests. For information about how to use CDSA$GEN_CERTS.COM to generate self-signed certificates and get them signed by the CA, see Chapter 5.

After the certificates are signed by the CA, they should be placed in the CDSA_SYSDIR:[SIGN] directory on a special system designated as the local signing system. At this point the CA also has a copy of this certificate in case it ever needs to be revoked. This certificate is now ready to be used to create manifests, as described in the following sections.

3.4.1 The Signing Process

The CDSA application CDSA$SIGN.EXE can be used to sign any file, including but not limited to executable files. The signing process combines three types of information into the manifest. Figure 3-1 illustrates these information types.

Figure 3-1 Information Combined Into Manifest



The format used to describe both the manifest and the signer's information is a series of Name:Value pairs (RFC 822).

Manifest Section example:

```
Manifest-Version: 2.0
Name: executable: cdsa$incssm300_shr.exe
SectionName: cdsa$incssm300_shr
Digest-Algorithms: SHA-1
SHA-1-Digest: sqHfj0uHdeNYZ5A062/78fZ6f3Q=
CDSA_MODULE: CSSM
CDSA _GUID: {d6b5e820-f376-11d3-9bea-0008c74fe165}
```

Signer Information Section example:

```
Signature-Version: 2.0
CDSA_PVC_API: OFF
Name: executable: cdsa$incssm300_shr .exe
SectionName: cdsa$incssm300_shr
Digest-Algorithms: SHA-1
SHA-1-Digest: kcwgKvohlRCnRXhghNUAcqT7lvY=
```

The PKCS#7 Signature block for this example is almost 4800 bytes of mostly binary data.

3.4.2 The CDSA$SD_SIGN.COM Procedure

The command file CDSA$SD_SIGN.COM uses CDSA$SIGN.EXE. CDSA$SD_SIGN.COM prompts for the file to sign, the signed certificate from the CA, and a password. This password must match the password that was used in the run files when creating the self-signed certificates.

The following is an example of calling CDSA$SD_SIGN.COM:

```
$ @CDSA_SYSDIR:[SIGN]CDSA$SD_SIGN.COM
Please enter the full name of the module to sign including directory:
    SYS$KITS:[KERBEROS]HP-AXPVMS-KERBEROS-V0200-6-1.PCSI
Please enter the name of your signed certificate, located in cdsa_sysdir:[sign]: intapps.cer
Please enter the password, it must match the password used to create self-signed certificates:
********
Invoking cdsa$sign.exe. Please wait.
Signed Manifest
```

Copy the manifest file to the same location as the file being signed. At this point, both files should be considered an inseparable pair. For example, if the original file is mastered onto a CD for distribution, the manifest should also be placed in the same directory on the CD. This allows the subsequent verification to take place, since the verification process looks in the directory where the original file being verified is located to find the manifest file.

3.4.3 The CDSA$REVOKE.EXE File

In some instances, you might need to revoke one of your certificates. If a certificate is compromised or is rendered invalid for any reason, you can use CDSA$REVOKE.EXE to revoke the certificate.

Input to this application is the name of a file containing a list of certificates (by name) to be revoked. The application simply writes the certificates to the data file along with other bookkeeping information so that the CA can interpret the file.

The resulting revocation file is saved in the CDSA_SYSDIR:[CRL]CDSA$REVOCATION_REQ.CRR file.

The following is an example of using CDSA$REVOKE.EXE:

```
$ REVOKE :== $SYS$SYSTEM:CDSA$REVOKE.EXE ! see SYS$MANAGER:CDSA$SYMBOLS.COM
$ REVOKE CDSA_SYSDIR:[CRL]REVOKE_EX.DAT;1
2 Certificates written to cdsa$revocation_req.crr.
CDSA_SYSDIR:[CRL]CDSA$REVOCATION_REQ.CRR successfully created.
$
```

The CDSA$REVOCATION_REC.CRR file can then be sent to the CA, where the data file is available to the CRL generation program. The CRL generation program updates the directory tree, and the certificate is considered revoked.

3.5 Validating Files and Authenticating Signers

The objective of validation is to determine the authenticity of the signer and the contents of the target file. These tasks are just two of the many pieces of functionality that CDSA uses to perform bilateral authentication of the calling and called programs. Secure Delivery performs both authentication and validation.

CDSA performs file validation in two ways:

- The CDSA$VALIDATE utility. The user invokes this utility specifying a target file argument. This utility is described in Section 4.8.
- The CDSA$VALIDATE_LIBSHR.EXE shareable image, which validates files programmatically. This shareable image is described in Section 3.5.2.

3.5.1 Validation Examples

The following two examples illustrate CDSA file validation. The first example validates a file called HP-AXPVMS-KERBEROS-V0200-6-1.PCSI$COMPRESSED and its associated manifest HP-AXPVMS-KERBEROS-V0200-6-1.PCSI _ESW.

```
$ VALIDATE :== $SYS$SYSTEM:CDSA$VALIDATE.EXE ! see SYS$MANAGER:CDSA$SYMBOLS.COM
$ VALIDATE /SYS$KIT/KERBEROS/HP-AXPVMS-KERBEROS-V0200-6-1.PCSI$COMPRESSED
Validation of /SYS$KIT/KERBEROS/HP-AXPVMS-KERBEROS-V0200-6-1.PCSI$COMPRESSED SUCCEEDED.
```

In the next example the same validation is attempted but the certificate used to create the manifest is revoked.

```
$ VALIDATE SYS$KIT/KERBEROS/HP-AXPVMS-KERBEROS-V0200-6-1.PCSI$COMPRESSED
validation of /SYS$KIT/KERBEROS/HP-AXPVMS-KERBEROS-V0200-6-1.PCSI$COMPRESSED FAILED.
Error: CSSMERR_TP_CERT_REVOKED
Certificate has been revoked
```

3.5.2 The CDSA$VALIDATE_LIBSHR.EXE File

For applications that validate files programmatically, there is no need to call CDSA$VALIDATE.EXE. Applications that link directly with CDSA$VALIDATE_LIBSHR.EXE can call the routine CDSA_FileValidate for their validation needs. Note that CDSA_FileValidate also returns an OpenVMS style return: SS$_NORMAL indicating success and 0 indicating failure. In addition, if CDSA_Ret_Status is not a NULL value passed in, then the address of a CDSA return status is assigned. The calling application must allocate and deallocate memory for CDSA_Ret_Status. Currently, the target file must be passed in as a UNIX style path name as in CDSA$VALIDATE.EXE.

In order for the validation process to succeed, the latest signed CRL published by the CA must be in the CDSA$SYSDIR:[CRL] directory. This file is CDSA$SECURE_DELIVERY.S_CRL and is used to make sure that the manifest file was not signed by a certificate that has already been revoked.

For more information, see the API “CDSA_FileValidate” on page 73.

4 CDSA Utility Programs

This chapter describes a number of administrative and development utilities that are provided with CDSA. Note that some of these programs are typically called only from the CDSA initialization command file unless new add-in modules are being provided.

The CDSA utility programs comprise the following:

- CDSA$CERTGEN.EXE - Generates digital certificates.
- CDSA$ISSUER.EXE - Generates the issuer key functions.
- CDSA$MDS_INSTALL.EXE - Creates the MDS database.
- CDSA$MOD_INSTALL.EXE - Adds entries to the MDS database.
- CDSA$OUTPUT_ERROR.EXE - Translates numeric CDSA error codes into text.
- CDSA$SIGN.EXE - Creates manifests.
- CDSA$VALIDATE.EXE - Validates Secure Delivery manifests.
- CDSA$REVOKE.EXE - Packages certificates to be revoked for shipment back to the Certificate Authority (HP OpenVMS).
- CDSA$X5092XML.EXE - Extracts the subject name from an X509 certificate.

The shortened program names listed in this chapter's Synopsis sections are defined in the file SYS$MANAGER:CDSA$SYMBOLS.COM. The following command should be added to the SYS$MANAGER:SYLOGIN.COM file on the system where CDSA development work is being done:

```
$ @SYS$MANAGER:CDSA$SYMBOLS.COM
```

If this command is not defined at the system level, individual CDSA developers should add it to their personal LOGIN.COM file so that they can use the shortened program names.

4.1 CDSA$CERTGEN.EXE

The certgen utility allows the user to create digital certificates in the form *runfilename.cer*. Private keys will be placed in [.CDSA.PKD]*csp-name.PRI* under the login directory of the current process.

This program generally is called by CDSA_SYSDIR:[SIGN]CDSA$GEN_CERTS.COM.

4.1.1 SYNOPSIS

```
certgen [runfilename]
```

4.1.2 OPTIONS

runfilename
: This optional parameter specifies the name of the run file that contains the parameters that certgen needs to create a certificate. If no run file is specified, the default run file is certgen.run in the current directory.

A certgen run file contains the following items as appropriate, each on a separate line:

certype location

certtype can be one of the following:

-s	Indicates a self-signed certificate.
-i	Indicates a certificate signed by another certificate.
-v	Indicates that the created certificate takes its subject and public key from a certificate issued by another vendor. You cannot use this option to create a self-signed certificate.
location	Indicates where the issuer certificate is read from if -i or -v is specified.

filename

If *certtype* is -s or -i, *filename* indicates the location of the XML template that contains the Subject Name that must go into this certificate. If *certtype* is -v, *filename* indicates the location of the Vendor Certificate.

algorithm

Indicates the algorithm used to generate the key pair associated with the certificate being created. The specified algorithm must be supported by one of the Cryptographic Service Providers available in the local implementation of CDSA. The algorithm can be either DSA or RSA. This parameter is not valid if -v is specified for *certtype*.

keysize

Specifies the logical key size (in bits) of the key pair being generated. Typical examples are 128, 256, 512, 1024, and so on. The specified key size must be supported by one of the Cryptographic Service Providers available in the local implementation of CDSA. This parameter is not valid if -v is specified for *certtype*.

cspguid

The globally unique identifier of the Cryptographic Service Provider that is being used.

certfile

The output file into which the created certificate is to be written.

subject_password

The password used to protect a key pair if one is being generated. This parameter is not valid if -v is specified for *certtype*.

issuer_password

The password used to unlock the private key required to sign the generated certificate. This parameter is not valid if -s is specified for *certtype*.

validity_period

The validity period for the certificate. This parameter contains a start and end date for the validity period in the form YYMMDDHHMMSS YYMMDDHHMMSS. The validity period cannot extend beyond the year 2049. If *validity_period* is not specified, the validity period for the certificate lasts for exactly one year.

4.1.3 EXAMPLE

```
$ certgen intmods.run
```

The following is an example of a run file (`intmods.run`) that creates a certificate named `intmods.cer`, which is signed by `intmanf.cer` and generates a 1024-bit DSA key pair.

```
-i intmanf.cer
intmods.xml
dsa
1024
{67ef50d0-fe74-11d2-a8e6-0090271d266f}
intmods.cer
intmods
intmanf
001013000000 101013000000
```

4.2 CDSA$ISSUER.EXE

The issuer utility is used to create a set of functions that are embedded into CSSM, or are used by EISL. A CDSA application developer needs to create only the `EISL_RetrieveSelfCheckKey()` function. The other functions noted here are applicable only for CDSA vendors (in this case, HP).

This program generally is called by `CDSA_SYSDIR:[SIGN]CDSA$GEN_CERTS.COM()`.

4.2.1 SYNOPSIS

`issuer` *`option certfile codefile functionname`*

4.2.2 OPTIONS

option

A code that defines the function to be created. Specify one of the following values:

`-i`	Creates a function that returns an issuer name from the certificate.
`-s`	Creates a function that returns a signer name from the certificate.
`-k`	Creates a function that returns a trusted public key.

Note: A CDSA application developer who is creating the `EISL_RetrieveSelfCheckKey()` function should specify `-k`. The other codes are used only by CDSA vendors who are building CDSA itself rather than a CDSA application or service provider module.

`certfile`

A text file that contains the name of the certificate to be used.

`codefile`

The file to which the generated function is written.

functionname

Name of the function to be generated.

Note: CDSA application developers need to create only the `EISL_RetrieveSelfCheckKey()` function (the last item in the following list). The full set of functions is listed here to provide a complete overview of the issuer utility. The other functions are applicable only for CDSA vendors. Those who want to learn more about export chains can refer to the *Intel Common Data Security Architecture Manifest Signing Tools User's Guide*.

- `cssm_GetIntegrityRootKeys()` (or `cssm_GetExportRootKeys()` for export)
- `cssm_GetIntegrityRootNames()` (or `cssm_GetExportRootNames()` for export)
- `EISL_RetrieveSelfCheckKey()`

4.2.3 EXAMPLE

The following example extracts the public key from the certificate intmods.cer and creates a function named `EISL_RetrieveSelfCheckKey()` in the file modselfkey.h.

```
$ create intmodscertfile.
intmods.cer
$!
$ issuer -k intmodscertfile. modselfkey.h -
_$ "EISL_RetrieveSelfCheckKey"
```

4.3 CDSA$MDS_INSTALL.EXE

The mds_install utility is used to create (install) or delete (uninstall) the Module Directory Services database used by CDSA.

This program generally is called by SYS$STARTUP:CDSA$INITIALIZE.COM.

4.3.1 SYNOPSIS

`mds_install [[-s source] [-d dbdest] ] [-u]`

4.3.2 OPTIONS

NOTE OpenVMS users can specify only the -u option (or no option). However, the other options are described here for completeness for users who are accustomed to seeing them on another platform.

-s *source* — Specifies the MDS DLL source location (not used by OpenVMS).

-d *dbdest* — Specifies the destination file specification for the MDS database to be created. This parameter is currently hardcoded on OpenVMS, and should not be changed.

-u	Specifies that the operation is an uninstall of MDS, rather than an install. This parameter cannot be used with the -s and -d parameters.

4.3.3 EXAMPLE

The following command creates an empty CDSA MDS database. (If it is run against an already existing database, it does nothing.)

```
mds_install
```

4.4 CDSA$MOD_INSTALL.EXE

The mod_install utility is used to add information about CDSA modules into the Module Directory Services database.

This program generally is called by SYS$STARTUP:CDSA$INITIALIZE.COM.

4.4.1 SYNOPSIS

```
mod_install [-f] option [-s file] [-d path]
```

4.4.2 OPTIONS

-f	Specifies not to warn about unsigned or corrupt modules.	
option	Specifies the action to be taken by the mod_install utility:	
	-i	Install the module.
	-u	Uninstall the module.
	-r	Refresh the installation information.
-s *file*	Specifies the full file specification (in UNIX® directory format) of the source file to be installed.	
-d *path*	Specifies the destination path (in UNIX directory format) of the source file to be installed.	

4.4.3 EXAMPLE

The following example installs the add-in module stubcsp300_shr.exe in the CDSA MDS database. The logical definition in the first command is necessary because the shareable image is not in SYS$LIBRARY and it will be invoked as part of the installation process.

```
$ define stubcsp300_shr "cdsa_tempdir:[addin]stubcsp300_shr.exe"
$ mod_install -i -s /cdsa_tempdir/addin -
_$ /stubcsp300_shr.exe -d /cdsa_tempdir/addin
```

4.5 CDSA$OUTPUT_ERROR.EXE

The CDSA implementation on OpenVMS supplies a special program that can be used to translate numeric CDSA error codes to text messages. This program resides in the SYS$SYSTEM directory and is called CDSA$OUTPUT_ERROR.EXE. It uses the routines described in this section to convert a numeric error code to its associated text label and error string. A foreign command, cdsa_error, has been defined in SYS$MANAGER:CDSA$SYMBOLS.COM to invoke this program. For details about using cdsa_error and its options, see Chapter 4, "CDSA Utility Programs," on page 37.

Note that this utility is defined as cdsa_error by CDSA$SYMBOLS.COM. The cdsa_error utility converts a CDSA numeric error code into its corresponding text strings. The text is output to SYS$OUTPUT.

4.5.1 SYNOPSIS

`cdsa_error` *`base_flag error_code`*

4.5.2 OPTIONS

`base_flag`

The mathematical base in which the error code is represented:

`-d`	Specifies that the numeric value of *`error_code`* is decimal (base 10).
`-o`	Specifies that the numeric value of *`error_code`* is octal (base 8).
`-h`	Specifies that the numeric value of *`error_code`* is hexadecimal (base 16).

If you specify something other than these options, you will get an error message that lists the correct options. (See Example 2.)

`error_code`

The error code stated in the numerical base specified by the *`base-flag`* parameter.

4.5.3 EXAMPLES

1.
```
$ cdsa_error -h 3135
Error: CSSMERR_DL_STALE_UNIQUE_RECORD
The record returned has been changed by someone and is stale
```

2.
```
$ cdsa_error -?
dka300:[sys0.syscommon.][sysexe]cdsa$output_error.exe;1:
illegal option -- ?
cdsa$output_error -d|o|h <Error Code>
options:
-d     : Error code is a decimal number
-o     : Error code is an octal number
-h     : Error code is a hexadecimal number
```

4.6 CDSA$REVOKE.EXE

Packages certificates to be revoked for shipment back to the Certificate Authority (HP OpenVMS).

4.6.1 SYNOPSIS

```
$ CDSA$REVOKE filename
```

4.6.2 OPTIONS

filename	The OpenVMS-style name of the file containing a list of certificates (by name) to be revoked.

4.6.3 RETURN VALUES

CDSA$REVOKE creates the file CDSA_SYSDIR:[CRL]CDSA$REVOCATION_REQ.CRR.

4.7 CDSA$SIGN.EXE

Note that this utility is defined as cdsa_sign by CDSA$SYMBOLS.COM. The cdsa_sign utility takes a service provider product, application, or CSSM binary, plus the manufacturer certificates generated using certgen, and creates a manifest file. Manifest files have a file extension of .ESW.

This utility can be used for Integrity signing and for Export signing. Integrity signing creates a new manifest, while Export signing adds signers to an existing manifest. The options for each function are totally different, so they are described here in separate sections. Integrity signing for a module must always be done before Export signing.

4.7.1 Integrity Signing

Integrity signing is optional for applications and mandatory for add-in modules.

4.7.1.1 SYNOPSIS

```
cdsa_sign module_name subdirectory type signer_cert password cert_chain
          module_guid access_tag pvcapi_tag pvcspi_tag priv_tag
```

4.7.1.2 OPTIONS

module_name	The name of the module being signed.	
subdirectory	The subdirectory (in UNIX directory format) containing the module being signed.	
type	The module type, which can be one of the following:	
	A	Service provider module
	C	CSSM

	D	Application sharable image
	E	Elective Module Manager
	G	Generic file
	X	Application executable

signer_cert — The name of the certificate being used to sign the module.

password — The password for the private key of the certificate being used to sign the module.

cert_chain — A text file identifying the certificates to be embedded. This file has the following form:

```
number
cert1
cert2
.
.
.
```

where *number* is the number of certificates being embedded, and *cert1* and *cert2* are the names of certificates to be embedded; for example:

```
2
introot.cer
intmanf.cer
```

module_guid — The string version of the globally unique identifier of the module being signed (as installed in MDS).

access_tag — For installer modules, this is the base-64 encoded, unsigned, 32-bit value (in big-endian) of the access type defined for CDSA_DB_ACCESS_TYPE. For modules other than installers, specify "XX" for this parameter.

pvcapi_tag — Specifies whether pointer validation checking is to be done on the application program interface boundaries. (Read more about PVC in Section 5.1.3.2.) The values for the CDSA_PVC_API tag are as follows:

"EXEMPT"	Specifies an application manifest, where the program can set the PVC flag in `cssm_Init()`.
"OFF"	Specifies a CSSM manifest, where the PVC flag is not applicable.
"XX"	Specifies that the CDSA_PVC_API tag is not in the manifest.

pvcspi_tag — Specifies whether pointer validation checking is to be done on the service provider interface boundaries. (Read more about PVC in Section 5.1.3.2.) The values for the CDSA_PVC_SPI tag are as follows:

"EXEMPT"	Specifies a service provider manifest, where the program can set the PVC flag in `cssm_Init()`.
"OFF"	Specifies a CSSM manifest, where the PVC flag is not applicable.

	"XX"	Specifies that the CDSA_PVC_SPI tag is not in the manifest.
priv_tag	The CDSA_PRIV tag in the manifest. No CDSA_PRIV tag values are defined, so specify "XX" to indicate that this tag is not in the manifest.	

4.7.1.3 EXAMPLE

The following is an example of the cdsa_sign command for Integrity signing:

```
$ define cdsa_sign "/cdsa_tempdir/addin"
$ set default cdsa_sysdir:[sign]
$ cdsa_sign stubcsp300_shr cdsa_sign A intmods.cer -
_$ intmods intchain. {79BDE0F0-4541-11d3-A8F3-0090271D266F} -
_$ "XX" "EXEMPT" "XX" "XX"
```

The first command defines the logical cdsa_sign (which is used internally by the code) in UNIX directory format as the directory where the executable to be signed can be found.

- stubcsp300_shr is the name of the module being signed.
- cdsa_sign is the logical pointing to the directory containing the module.
- A indicates that stubcsp300_shr is a service provider module.
- intmods.cer is the name of the certificate being used to sign the module.
- intmods is the password for the private key of the certificate (intmods.cer) being used to sign the module.
- intchain. is the name of the text file containing the names of the certificates in the certificate chain.
- {79BDE0F0-4541-11d3-A8F3-0090271D266F} is the GUID of the service provider module.
- "XX" is the access tag, which indicates that this is not an installer module.
- "EXEMPT" is the CDSA_PVC_API tag specifying that this is an application manifest.
- "XX" specifies that the CDSA_PVC_SPI tag is not in the manifest.
- "XX" specifies that the CDSA_PRIV tag is not in the manifest.

4.7.2 Export Signing

Export signing is optional. Before you can do Export signing for a module, you must already have done Integrity signing and a manifest must exist. For more information about Export signing, refer to the *Intel Common Data Security Architecture Manifest Signing Tools User's Guide.*

4.7.2.1 SYNOPSIS

cdsa_sign *manifest_path signer_cert password cert_chain usee_tag priv_tag pvcapi_tag pvcspi_tag*

4.7.2.2 OPTIONS

manifest_path	The path (in UNIX directory format) to the manifest created in the Integrity signing phase.
signer_cert	The name of the certificate being used to sign the module.
password	The password for the private key of the certificate being used to sign the module.

cert_chain

A text file identifying the certificates to be embedded. This file has the following form:

```
number
cert1
cert2
.
.
.
```

where *number* is the number of certificates being embedded, and *cert1* and *cert2* are the names of certificates to be embedded; for example:

```
2
introot.cer
intmanf.cer
```

usee_tag

The base-64 encoded value of the CSSM_USEE_TAG value. This value must be enclosed within double quotation marks.

priv_tag

The CDSA_PRIV tag in the manifest. No CDSA_PRIV tag values are defined, so specify "XX" to indicate that this tag is not in the manifest.

pvcapi_tag

Specifies whether pointer validation checking is to be done on the application program interface boundaries. (Read more about PVC in Section 5.1.3.2.) The values for the CDSA_PVC_API tag are as follows:

"EXEMPT"	Specifies an application manifest, where the program can set the PVC flag in cssm_Init.
"OFF"	Specifies a CSSM manifest, where the PVC flag is not applicable.
"XX"	Specifies that the CDSA_PVC_API tag is not in the manifest.

pvcspi_tag

Specifies whether pointer validation checking is to be done on the service provider interface boundaries. (Read more about PVC in Section 5.1.3.2.) The values for the CDSA_PVC_SPI tag are as follows:

"EXEMPT"	Specifies a service provider manifest, where the program can set the PVC flag in cssm_Init.
"OFF"	Specifies a CSSM manifest, where the PVC flag is not applicable.
"XX"	Specifies that the CDSA_PVC_SPI tag is not in the manifest.

4.7.2.3 EXAMPLE

The following is an example of the cdsa_sign command for Export signing:

```
$ cdsa_sign /cdsa_tempdir/des2/des2.esw exapps.cer secret exchain. -
_$ "AAAAAQ==" "XX" "EXEMPT" "XX"
```

In this example:

- /cdsa_tempdir/des2/des2.esw is the path (in UNIX directory format) to the manifest created during Integrity signing.

- exapps.cer is the name of the certificate being used to sign the module.
- secret is the password for the private key of the certificate being used to sign the module.
- exchain. is the name of the text file identifying the certificates to be embedded in the signature.
- "AAAAAQ==" is the base-64 encoded value of the CDSA_USEE_DOMESTIC tag.
- "XX" specifies that the CDSA_PRIV tag is not in the manifest.
- "EXEMPT" is the CDSA_PVC_API tag specifying that this is an application manifest.
- "XX" specifies that the CDSA_PVC_SPI tag is not in the manifest.

4.8 CDSA$VALIDATE.EXE

4.8.1 SYNOPSIS

```
$ CDSA$VALIDATE filename
```

4.8.2 OPTIONS

filename The name of the file that is the target of validation. Currently, the filename must be a UNIX style path in order to be compatible with CDSA.

4.8.3 DESCRIPTION

CDSA$VALIDATE is a signed CDSA application. It will collect the input parameters, file specs for target file and manifest, and pass them on to `CDSA_FileValidate` routine that is part of CDSA$VALIDATE_LIBSHR.EXE.

4.8.4 EXAMPLE

```
$ validate :== $sys$system:cdsa$validate.exe  ! see SYS$MANAGER:CDSA$SYMBOLS.COM
$ validate user1/mydirectory/myfile.pcsi
```

4.8.5 RETURN VALUES

CDSA$VALIDATE returns SS$_NORMAL for success and 0 if the validation fails, or an error occurs. If a problem is encountered during the validation process, CDSA$VALIDATE prints a CDSA error message.

4.9 CDSA$X5092XML.EXE

The x5092xml utility reads an X509 certificate file, extracts the subject name, and writes the name as XML to an XML file. This tool is useful for producing example template files that can be modified.

4.9.1 SYNOPSIS

`x5092xml` *`infile outfile`*

4.9.2 OPTIONS

`infile`	The name of the X509 certificate file from which the subject name is being extracted.
`outfile`	The name of the XML file to which the name is to be written.

4.9.3 EXAMPLE

```
x5092xml introot.cer introot.xml
```

5 CDSA Programming Concepts

This chapter provides an overview of programming with CDSA on OpenVMS. This chapter should be read in conjunction with the *Intel Common Data Security Architecture Application Developer's Guide*, the *Intel Common Data Security Architecture Service Provider Developer's Guide*, and *the Intel Common Data Security Architecture Manifest Signing Tools User's Guide*.

This chapter covers the following topics:

- An overview of building a CDSA application on OpenVMS (see Section 5.1)
- Details about writing a signed CDSA application or add-in module (see Section 5.2)
- Steps to deploy signed applications and service provider modules (see Section 5.3)
- Descriptions of the CDSA example programs (see Section 5.4)
- Information about CDSA errors and how to get a meaningful error return (see "Decode_CDSA_Error" on page 258 and "Print_CDSA_Error" on page 415.)

5.1 Overview of CDSA Programming on OpenVMS

CDSA programming on OpenVMS works much the same as on any other platform. The following sections indicate differences and important information.

5.1.1 Compiling a CDSA Program

When you compile your program, you need to add the /INCLUDE=CDSA_SYSDIR:[INCLUDES] qualifier to your compiler command line. The following command is taken from the BUILD_DES.COM example in this chapter (see Section 5.4.2):

```
$ CC/LIST/INCLUDE=CDSA_SYSDIR:[INCLUDES]/PREFIX=ALL DO_DES
```

5.1.2 Linking a CDSA Program

Most CDSA applications must link with SYS$SHARE:CDSA$INCSSM300_SHR.EXE. If the application uses MDS, you might need to include SYS$SHARE:CDSA$MDS300_SHR.EXE and SYS$SHARE:CDSA$MDS_UTIL_API.OLB as well.

Because CDSA routines are located in shareable libraries, the use of a link options file is recommended. For details about using link options files, refer to the *OpenVMS Linker Utility Manual*. The CDSA example programs described in Section 5.4 provide examples of using link options files for CDSA applications.

5.1.3 CDSA Integrity Checking

CDSA provides two types of integrity checking: bilateral authentication and pointer validation checking.

5.1.3.1 Bilateral Authentication

Bilateral authentication checks the integrity of modules as they are dynamically loaded into the system. A bilateral authentication procedure is designed for two entities to establish trust in the identity and integrity of each other. When loading a service provider module or an elective module manager, CDSA requires that the attaching module participate in this authentication protocol. Both modules in the bilateral authentication procedure must have signed credentials that bind them to the trust hierarchy used by CDSA. These credentials are stored in the CDSA MDS database during module installation.

Refer to the *Intel Common Data Security Architecture Application Developer's Guide* (Chapter 11, Integrity) and the *Intel Common Data Security Architecture Manifest Signing Tools User's Guide* for more detailed explanations of the bilateral authentication process.

5.1.3.2 Pointer Validation Checking

Pointer validation checking (PVC) entails validating addresses under the following circumstances:

- Before calling across the application interface into CDSA (PVC is optional on OpenVMS in this case.)
- Before calling across the CDSA interface to an add-in module (PVC is required on OpenVMS in this case.)

The Pointer Validation Policy is established using the PvcPolicy parameter in the CSSM_Init call. The parameter values can be derived using the constants in the file CSSMTYPE.H in CDSA_SYSDIR:[INCLUDES]. Starting with OpenVMS Alpha Version 7.3-2, the values for the PvcPolicy parameter that are valid for CDSA are as described in the following table.

Value	Description
2	PVC validation is performed on service provider modules only. CSSM_PVC_SP is used for PVC validation on service provider modules.
3	PVC validation is performed on both service provider and application modules. The bitwise OR of CSSM_PVC_APP and CSSM_PVC_SP is used for PVC validation on both service provider and application modules; for example, (CSSM_PVC_APP | CSSM_PVC_SP).

For more information about pointer validation checking, see the description of the `CSSM_Init()` API.

5.2 Writing Signed Applications

Two types of applications can be developed to use CDSA integrity checking:

- An application that calls into CDSA to use one or more of the services that it provides.

  CDSA applications developed on OpenVMS can optionally participate with CDSA in bilateral authentication.

- A service provider module that "plugs-in" or "adds-in" to CDSA to provide a set of security related functions that an application program can in turn use. On OpenVMS, service provider modules are implemented as shareable images.

  All CDSA add-in modules developed on OpenVMS must participate in bilateral authentication (see Section 5.1.3.1) and pointer validation checking (see Section 5.1.3.2).

The *Intel Common Data Security Architecture Application Developer's Guide* and the *Intel Common Data Security Architecture Service Provider Developer's Guide* have in-depth information about developing applications and add-in modules for CDSA.

The development process includes generating certificates and key pairs to be used in the signing process and later in the integrity checking process. The public keys are extracted from the certificate into a code module that is included in the application. The private keys remain on the signing system. After the code is built, the certificate is used to "sign" the application or service provider module. The product of the signing is a manifest, which is typically kept with the executable.

The following sections summarize the steps for building a signed CDSA application or add-in module on OpenVMS.

5.2.1 The Signing Environment

To create manifests used for authentication of CDSA modules, you must have a working version of CDSA and the signing tools installed on a machine. It is good practice to dedicate a specific machine or set of machines to be the signing center. Certificates for signing should be generated on the signing machine, and the signing of generated modules must be done there. The tools, applications, CDSA stack, and private keys used to generate certificates should not be modified or reinstalled after the certificate generation process has completed. Doing so will invalidate the keys used to make the certificates and will cause any modules signed to fail integrity checking.

Development and testing of modules should be conducted on other machines so as not to disrupt the signing environment.

The signing directory on an OpenVMS system is CDSA_SYSDIR:[SIGN].

On OpenVMS, the account that is used to create certificates must be the same account that is used for signing developed applications and service-provider modules. This is required because the private keys are stored in the namespace of that user account and must be accessible by the code performing those functions. Note that this account requires the SYSPRV privilege to access the signing directory.

5.2.2 The Signing Tools

The following programs are used in developing CDSA applications or add-ins:

Program Name	Description
SYS$SYSTEM:CDSA$CERTGEN.EXE	Certificate creation tool
SYS$SYSTEM:CDSA$ISSUER.EXE	Public key extraction tool
SYS$SYSTEM:CDSA$SIGN.EXE	Signing tool

The following files in CDSA_SYSDIR:[SIGN] are named according to Intel naming conventions. Their names can be changed to suit any other development conventions. If the names introot.cer or intmanf.cer are changed, intchain must be updated to reflect the new names. The new certificate names will also be used as parameters to cdsa_sign.

File Name	Description
introot.cer	The CDSA Integrity Root certificate containing the public key of the root of the integrity chain.

File Name	Description
intmanf.cer	The CDSA Integrity Manufacturing certificate containing the public key of the manufacturer.
ssintapps.run	The run file that is input to the certificate creation tool (CDSA$CERTGEN.EXE) to create a self-signed application certificate.
ssintapps.xml	The X509 formatted identification of the signer of the application certificate.
ssintmods.run	The run file that is input to the certificate creation tool (CDSA$CERTGEN.EXE) to create a self-signed add-in module certificate.
ssintmods.xml	The X509 formatted identification of the signer of the add-in module certificate
intchain.	A list of certificates comprising the integrity certificate chain; that is, introot.cer and intmanf.cer

The file CDSA_SYSDIR:[SIGN]CDSA$GEN_CERTS.COM is used to generate the digital certificates and keypairs that are used by CDSA applications.

5.2.3 The Signing Process

The first five of the following nine steps need to be done only once for each application or add-in module being developed. However, each time the application is changed, a new manifest must be created and the application must be reinstalled in the CDSA MDS database (steps 8 and 9).

If you are building the example programs provided with CDSA Version 2.0 or later, some of the following steps have been done in example code or accompanying command procedures. Read SYS$COMMON:[SYSHLP.EXAMPLES.CDSA]README.TXT for details.

1. Generate a GUID.

   Each signed application and service provider module should have a global unique identifier (GUID). This GUID should be written to a header file in the application development directory — either as an individual header file or included in another header file. (See the model in DESGUID.H in the DES2 or DES3 examples: Section 5.4.4 or Section 5.4.5.)

   If your development environment is OpenVMS Version 7.3-2 or higher, you can simply execute the GUID generating command procedure in CDSA_SYSDIR:[SIGN] and the procedure will output a GUID as shown in the following example:

   ```
   $ @CDSA_SYSDIR:[SIGN]CDSA$UUIDGEN
   xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
   ```

   The string form of the GUID is used as input to the signing tool, CDSA$SIGN.EXE, when the application or add-in module is signed.

   The string form of a GUID is expressed as follows:

   ```
   "{FD52A3EA-D9EC-1159-916B-08002BC48051}"
   ```

   The numeric form of the same GUID (as defined by the data structure CSSM_GUID) would be:

   ```
   {0xfd52a3ea,
   0xd9ec,
   0x1159,
   {0x91, 0x6b, 0x08, 0x0, 0x2b, 0xc4, 0x80, 0x51}}
   ```

Add a GUID variable pointer to the calls to `CSSM_Init()` and, if you are using them, to `CSSM_Introduce()` and `CSSM_Unintroduce()`.

NOTE If you are developing on a system earlier than OpenVMS Version 7.3-2, you must find another method to generate a GUID that conforms to the preceding format.

---

2. Generate a Certificate.

   The first step in the process of creating credentials is to generate a self-signed certificate by running CDSA$CERTGEN.EXE. This is always done on the signing system. The default directory must be set to CDSA_SYSDIR:[SIGN] and the user must have read/write access to this directory. (Steps 3 and 8, generating the key and the manifest, must also be done in this directory.)

   This step produces a private key and a public key for the application. The private key always remains on the signing system. The matching public key is embedded in the generated certificate.

   A .RUN file and an .XML file are input to CDSA$CERTGEN.EXE. The following samples of these files can be found in CDSA_SYSDIR:[SIGN]:

   - ssintapps.run and ssintapps.xml (input to generate an application certificate)
   - ssintmods.run and ssintmods.xml (input to generate a service provider module certificate)

   The .RUN file contains input to the certificate generation process, including the name of the .XML file. The .XML file contains attributes to identify the issuer of a certificate in machine-readable X500 format. The following table shows the attributes that are used. The attribute name is not used in the .XML file but is included in the table for human readability. Note that only one value is specified for each attribute in the .XML file.

Attribute OID	Attribute Name	Example Value	OpenVMS Value
2.5.4.3	Common Name	Senior Technician	Hewlett-Packard
2.5.4.10	Organization Name	XYZ Company	BCS (Business Critical Servers)
2.5.4.11	Organizational Unit Name	ABC Division	OpenVMS
2.5.4.1	Aliased Entry Name	XYZ Security Product	HP OpenVMS Integrity Root
2.5.4.9	Street Address	110 Maple Street	110 Spit Brook Road
2.5.4.7	Locality	Anytown	Nashua
2.5.4.8	State or Province	XX	NH
2.5.4.6	Country	USA	USA
2.5.4.17	Postal Code	54321	03062
2.5.4.23	Telephone Number	777-666-4321	(not used)
1.2.840.113549.1.9.1	Email Address	role@xyz.com	OpenVMSSecurity@hp.com

   Make the desired changes to the attributes in the .XML file to identify the issuer of the certificates. Chapter 3 of the *Intel Common Data Security Architecture Manifest Signing Tools User's Guide* explains the XML syntax used here.

You can run CDSA$CERTGEN.EXE by itself or you can execute the command procedure CDSA_SYSDIR:[SIGN]CDSA$GEN_CERTS.COM to run both CDSA$CERTGEN.EXE to generate a certificate and CDSA$ISSUER.EXE to generate the key code (see Step 3).

3. Generate Key Code.

   CDSA$ISSUER.EXE generates the code that embeds the public key in the application. You can run this program by itself in directory CDSA_SYSDIR:[SIGN] or you can let it execute as part of CDSA_SYSDIR:[SIGN]CDSA$GEN_CERTS.COM. CDSA$ISSUER.EXE extracts the public key into a C structure to be included in the developed program. It generates two certificates, ssintapps.cer and ssintmods.cer.

   Because the generated certificates are self-signed, they also need to be signed with the private key of the root of the integrity certificate chain being used for CDSA. This root is the private key originally generated by OpenVMS. This certificate signing is accomplished by sending email to `OpenVMSSecurity@hp.com`. The response will provide details on how to proceed with having your certificates signed by the OpenVMS integrity root.

   CDSA$ISSUER.EXE also generates the following include files:

   - APPSELFKEY.H (used to develop an application)
   - MODSELFKEY.H (used to develop a service provider module)

   Copy these two files into the application development area.

4. Generate SelfCheck code.

   For an application:

   As part of the self-check process, you must modify the following three procedures in the CALLOUTS.C module found in each CDSA example directory:

   - `EISL_RetrieveSelfCheckSectionName()`
   - `EISL_RetrieveSelfCheckCredentials()`
   - `EISL_RetrieveSelfCheckCredentialsSize()`

   Modify these procedures to use the application GUID in calls to `mdsutil_GetModuleCredentialInfo()`. In the DES2 and DES3 examples in this chapter (see Section 5.4.4 and Section 5.4.5), the application GUID is defined by including a file called DESGUID.H.

   Define the constant SECTION to be the name of the application executable.

   CALLOUTS.H contains function prototypes for all the self-check procedures that will be invoked.

   For an add-in module:

   Change the definition of ADDIN_SELF_CHECK_SECTION in the MAF_CONFIG.H module in the example directory to the name of the shareable image (with no extension).

5. Add CDSA procedures to the Application.

   Before making any calls to CSSM, insert a call to `EISL_SelfCheck()` to validate the integrity of the application itself. After a successful return, call `EISL_RecycleVerifiedModuleCredentials()` to release the structures that were created.

   If you want to ensure the integrity of CDSA, you can load it dynamically and let the code perform integrity checking on it before any CSSM code is executed. One way to do this is by using the Application Adaptation Layer. All code to use this layer is provided in the DES3 example program. Call `AALProxyLoadCssm()` after `EISL_SelfCheck()`, and before making any calls to CSSM.

If you want to perform pointer validation checking across the API boundary, you must call the APIs in the following order so that the necessary data structures are set up:

- `CSSM_Init()`
- `CSSM_Introduce()`
- `CSSM_ModuleLoad()`

When processing ends, the application should call `CSSM_Unintroduce()` (if you used it) before calling `CSSM_Terminate()` and then `AALProxyUnloadCssm()`.

CDSA Add-in Modules

The integrity checking process for add-in modules is provided by the Multi-service Add-in Framework. In fact, the MAF*.* modules provide a framework for developing an add-in module.

Development of a CDSA service provider add-in module is beyond the scope of this document. The OpenVMS CDSA example application ADDIN illustrates the development of a Cryptographic Service Provider add-in module. The *Intel Common Data Security Architecture Service Provider Developer's Guide* provides complete details for developing an add-in module for CDSA.

6. Compile and link the application or add-in module.
7. Build the code to install the application.

   A service provider module can be installed in the CDSA MDS database using SYS$SYSTEM:CDSA$MOD_INSTALL.EXE.

   An application must build a program to perform the installation. The two signed example applications DES2 and DES3 include an installation program that demonstrates the basics of installing an application.
8. Generate the manifest.

   In directory CDSA_SYSDIR:[SIGN] on the signing system, sign the application by generating a set of credentials. The application credentials are contained in a manifest, *application*.ESW, which accompanies the application. Input to the credential generation includes the application executable and the certificate being used to sign the application. For more details, refer to the *Intel Common Data Security Architecture Manifest Signing Tools User's Guide*.

   Each of the example programs described in this chapter includes a procedure called *example*_SIGN.COM that demonstrates how to generate a manifest.

   The manifests are typically kept with the application executable.
9. Install the application in the CDSA MDS database.

   Each of the example programs includes code that produces an application program and a procedure called *example*_INSTALL.COM that demonstrates how to install an application in the CDSA MDS database.

5.3 Deploying Signed Applications and Service Provider Modules

To deploy a CDSA signed application or service provider module, you must deliver the following items to the system where they are to be used:

- The executable

- The manifest (*filename*.ESW) containing the credentials of the executable
- The installation program (for an application, a service provider module can use CDSA$MOD_INSTALL.EXE)

After the files are in place, run the installation program.

5.4 CDSA Example Programs

Eight example programs are provided with CDSA on OpenVMS. Command procedures to build, sign, and install them are provided along with individual README files for each example.

The following table lists the example programs and describes what aspect of CDSA each program is designed to convey.

Example Program	Signed	Description	Section
AES	No	Simple AES encryption/decryption program	Section 5.4.1
DES	No	Simple DES encryption/decryption program	Section 5.4.2
MDS	No	Program to query MDS database for CDSA services	Section 5.4.3
DES2	Yes	DES example with integrity checking, explicitly linked	Section 5.4.4
DES3	Yes	DES example with integrity checking, using AAL (dynamically loaded)	Section 5.4.5
ADDIN	Yes	An add-in module written to the CSP Service Provider Interface, with integrity checking	Section 5.4.6
DUMMYEMM	Yes	An Elective Module Manager to define a new Service Provider Interface, wtih integrity checking	Section 5.4.7.1
DUMMYEMMADDIN	Yes	An add-in module written to the SPI made available by DUMMYEMM, with integrity checking	Section 5.4.7.2

Before you build the example programs, please read the following README files:

- For an overview of all the CDSA examples: SYS$COMMON:[SYSHLP.EXAMPLES.CDSA]README.TXT
- For details about an individual example program, see the README file in each example directory. For example, the README file for DES is in the following location: SYS$COMMON:[SYSHLP.EXAMPLES.CDSA.DES]README.TXT

You must initialize CDSA before running any example program. See Chapter 2 for more information.

Pay special attention to Section 5.2 if you plan to build one of the signed examples or are developing a CDSA add-in module.

The examples are designed to be organized under a local build area or directory such as *disk*:[*directory*.example].

Define the rooted logical CDSA_TEMPDIR as *disk*:[*directory*.] using the following command:

```
$ DEFINE/TRANSLATION=CONCEALED CDSA_TEMPDIR disk:[directory.]
```

Under this directory, the command procedures expect to find individual directories for each example; for example:

```
DISK1:[EXAMPLES.DES]
DISK1:[EXAMPLES.MDS]
DISK1:[EXAMPLES.DES2]
```

5.4.1 AES Encryption/Decryption Example Program

This example is a simple AES encryption/decryption program that uses CDSA, along with the necessary files to build it on OpenVMS. It consists of two source files (AES.C and DO_AES.C), and two build files (AES_BUILD.COM and AES.OPT).

The AES example program can be built by copying the example files into a local build area, and executing the AES_BUILD command file, as follows:

```
$ copy SYS$SYSROOT:[SYSHLP.EXAMPLES.CDSA.AES]*.* local_build_area
$ SET DEF local_build_area
$ @AES_BUILD
```

The resulting AES.EXE file can be run as a foreign command. This can be set up by entering the following command:

```
$ AES :== $ local_build_area AES.EXE
```

You can then execute the program with the following options:

Option	Description
-e	Encrypt with supplied key (requires -k option)
-d	Decrypt with supplied key (requires -k option).
-h	The supplied key is up to a 64 character hexadecimal number.
-k "*key*"	Use key "key" (single quotes are necessary if used with -h).

NOTE Up to 64 characters are used for 256 bit AES (this example is included in CDSA). Up to 48 characters are used for 192 bit AES. Up to 32 characters are used for 128 bit AES.

For example, to encrypt MYFILE.TXT using an ascii key with the AES example program, issue the following command:

```
$ AES -e -k "xyzzy" MYFILE.TXT MYFILE.AES
```

To decrypt the same file, enter the following command:

```
$ AES -d -k "xyzzy" MYFILE.AES MYFILE.TXT
```

To encrypt/decrypt using a hexadecimal key, use a key length of exactly 64 typed characters (32 hex bytes), and the -h switch as follows:

```
$ AES -e -k 012abcde012abcde -h MYFILE.TXT MYFILE.AES
$ AES -d -k 012abcde012abcde -h MYFILE.AES MYFILE.TXT
```

To change this example to a 128 or 192 bit AES example, perform the following steps.

1. Edit AES. For 192 bit AES, change the key size as follows:

   ```
   key[32]
   ```
   to
   ```
   key[24]
   ```

   For 128 bit AES, change the key size as follows:

   ```
   key[32]
   ```
   to
   ```
   key[16]
   ```

2. Edit DO_AES. For 192 bit AES, change the following:

   ```
   key.KeyHeader.AlgorithmId = CSSM_ALGID_EVP_AES_256;
   ```
   to
   ```
   key.KeyHeader.AlgorithmId = CSSM_ALGID_EVP_AES_192;
   ```

   For 128 bit AES, change the following:

   ```
   key.KeyHeader.AlgorithmId = CSSM_ALGID_EVP_AES_256;
   ```
   to
   ```
   key.KeyHeader.AlgorithmId = CSSM_ALGID_EVP_AES_128;
   ```

3. Rebuild the example.

5.4.2 DES Encryption/Decryption Example Program

This example is a simple DES encryption/decryption program that uses CDSA with no integrity checking. It links explicitly against CDSA$INCSSM300_SHR.EXE.

The DES example includes two source files (DES.C and DO_DES.C) and two build files (BUILD_DES.COM and DES.OPT).

Copy the example files into a local build area and then execute the BUILD_DES command file, as follows:

```
$ COPY SYS$SYSROOT:[SYSHLP.EXAMPLES.CDSA.DES]*.* disk:[directory.DES]
$ SET DEFAULT disk:[directory.DES]
$ @BUILD_DES
```

It is easiest to run the resulting DES.EXE file as a foreign command. Define a symbol for this command as follows:

```
$ DES :== $ disk:[directory.DES]DES.EXE
```

You can now execute the program using any of the following applicable options:

Option	Description
-e	Encrypt with supplied key (requires -k option)

Option	Description
-d	Decrypt with supplied key (requires -k option).
-k "*key*"	Supplies a key, which must be enclosed within double quotation marks if it is ASCII and case sensitive; no quotation marks are allowed for hexadecimal numbers.
-h	The supplied key is a 16-character hexadecimal number.

For example, to encrypt MYFILE.TXT using an ASCII key with the DES example program, enter the following command using double quotation marks, as shown, if the key is case sensitive:

```
$ DES -e -k "xyzzy" MYFILE.TXT MYFILE.DES
```

To decrypt the same file, enter the following command:

```
$ DES -d -k "xyzzy" MYFILE.DES MYFILE.TXT
```

To encrypt or decrypt with a hexadecimal key, use the -h option and make sure the key length is exactly 16 typed characters (8 hexadecimal bytes). No quotation marks, either single or double, are allowed. For example:

```
$ DES -e -k 012abcde012abcde -h MYFILE.TXT MYFILE.DES
$ DES -d -k 012abcde012abcde -h MYFILE.DES MYFILE.TXT
```

5.4.3 MDS Example Program

This program uses some of the MDS and CSSM services of CDSA, with no integrity checking. It links explicitly against CDSA$INCSSM300_SHR.EXE.

The MDS example includes one source file (MDS_EXAMPLE.C) and two build files (BUILD_MDS_EXAMPLE.COM and MDS_EXAMPLE.OPT).

The program follows the descriptions and code fragments from the *Intel Common Data Security Architecture Application Developer's Guide*.

Build the MDS example program by copying the example files into a local build area and then executing the BUILD_MDS_EXAMPLE command file, as follows:

```
$ COPY SYS$SYSROOT:[SYSHLP.EXAMPLES.CDSA.MDS]*.* disk:[directory.MDS]
$ SET DEFAULT  disk:[directory.MDS]
$ @BUILD_MDS_EXAMPLE
```

The resulting MDS_EXAMPLE.EXE file takes no parameters and can be executed as follows:

```
$ RUN  disk:[directory.MDS]MDS_EXAMPLE
```

The following is an excerpt of output from the program:

```
$ RUN MDS_EXAMPLE.EXE

Module 0) Name: SSLeay Crypto Based CSP
Module 0) ModuleGuid: {67ef50d0-fe74-11d2-a8e6-0090271d266f}
Module 0) Version: 3.1
Module 0) CompatibleCSSMVersion: 2.1
Module 0) Description: SSLeay Crypto Based CSP
Module 0) Vendor: Hewlett-Packard Company
Module 0) Flags: 0x0
Module 0) ServiceMask: 0x2
  Service 0) Description: SSLeay Crypto Based CSP
```

```
  Service 0) Type: CSSM_SERVICE_CSP
  Service 0) Flags: 0x0
    SubService 0) ModuleType: 0
    SubService 0) SubServiceId: 0
    This is a SOFTWARE subservice with 30 capabilities
        Context Type: CSSM_ALGCLASS_RANDOMGEN
        Algorithm Type: CSSM_ALGID_MD5Random
            Attribute Type: CSSM_ATTRIBUTE_BLOCK_SIZE
            Attribute Type: CSSM_ATTRIBUTE_DESCRIPTION
        Context Type: CSSM_ALGCLASS_DIGEST
        Algorithm Type: CSSM_ALGID_MD5
            Attribute Type: CSSM_ATTRIBUTE_OUTPUT_SIZE
            Attribute Type: CSSM_ATTRIBUTE_DESCRIPTION
.
.
.
Module 1) Name: CDSA Adaptation Layer CSP for the BSafe Toolkit from RSA DSI
Module 1) ModuleGuid: {d6b5e822-f376-11d3-9bea-0008c74fe165}
Module 1) Version: 3.1
Module 1) CompatibleCSSMVersion: 2.1
Module 1) Description: CDSA Adaptation Layer CSP for the BSafe Toolkit from RSA
                  DSI
Module 1) Vendor: Hewlett-Packard Company
Module 1) Flags: 0x0
Module 1) ServiceMask: 0x2
  Service 0) Description: CDSA Adaptation Layer CSP for the BSafe Toolkit from RSA
                  DSI
  Service 0) Type: CSSM_SERVICE_CSP
  Service 0) Flags: 0x0
    SubService 0) ModuleType: 0
    SubService 0) SubServiceId: 0
    This is a SOFTWARE subservice with 33 capabilities
        Context Type: CSSM_ALGCLASS_RANDOMGEN
        Algorithm Type: CSSM_ALGID_MD2Random
            Attribute Type: CSSM_ATTRIBUTE_DESCRIPTION
        Context Type: CSSM_ALGCLASS_RANDOMGEN
        Algorithm Type: CSSM_ALGID_MD5Random
            Attribute Type: CSSM_ATTRIBUTE_DESCRIPTION
.
.
.
```

5.4.4 DES2 Encryption/Decryption Example Program

The DES2 example program is nearly identical to the DES example except that it uses integrity checking in addition to the encryption/decryption CDSA calls. It links explicitly against CDSA$INCSSM300_SHR.EXE. This example is designed to be signed using the CDSA signing tools.

The necessary files to build the example on OpenVMS are included, with the exception of APPSELFKEY.H. This include file must be generated from the certificate created for the application.

See Section 5.2 for complete instructions. A signed CDSA application will not execute until the proper credentials are generated.

After you generate the application credentials and the include file, APPSELFKEY.H, you can build the DES2 example program by copying the example files into a local build area and executing the DES2_BUILD command file, as follows:

```
$ DEFINE/TRANS=CONCEALED CDSA_TEMPDIR disk:[directory.]
$ SET DEFAULT CDSA_TEMPDIR:[DES2]
$ COPY SYS$SYSROOT:[SYSHLP.EXAMPLES.CDSA.DES2]*.* []
$ COPY CDSA_SYSDIR:[SIGN]APPSELFKEY.H []
$ @DES2_BUILD
```

The resulting image, DES2.EXE, must be signed. On the signing system, run the following command procedure to generate the manifest:

```
$ @DES2_SIGN
```

Finally, on the development system, run the command procedure to install the module, as follows:

```
$ @DES2_INSTALL
```

It is easiest to run the application DES2.EXE file as a foreign command. Define a symbol for this command as follows:

```
$ DES2 :== $CDSA_TEMPDIR:[DES2]DES2.EXE
```

The options and program usage are the same as for the DES example.

5.4.5 DES3 Example Program

The DES3 example program is nearly identical to the DES2 example except that it links dynamically at run-time against CDSA$INCSSM300_SHR.EXE using the CDSA Application Adaption Layer.

This example is designed to be signed using the CDSA signing tools.

The files necessary to build the example on OpenVMS are included, with the exception of APPSELFKEY.H. This include file must be generated from the certificate created for the application.

See Section 5.2 for complete instructions on writing a signed application. A signed CDSA application will not execute until the proper credentials are generated.

After you generate the application credentials and the include file APPSELFKEY.H, you can build the DES3 example program by copying the example files into a local build area and executing the DES3_BUILD command file, as follows:

```
$ DEFINE/TRANS=CONCEALED CDSA_TEMPDIR disk:[directory.]
$ SET DEFAULT CDSA_TEMPDIR:[DES3]
$ COPY SYS$SYSROOT:[SYSHLP.EXAMPLES.CDSA.DES3]*.* []
$ COPY CDSA_SYSDIR:[SIGN]APPSELFKEY.H  []
$ @DES3_BUILD
```

The resulting image, DES3.EXE, must be "signed". On the signing system, run the following command procedure to generate the manifest:

```
$ @DES3_SIGN
```

Finally, on the development system, run the command procedure to install the module, as follows:

```
$ @DES3_INSTALL
```

It is easiest to run the resulting DES3.EXE file as a foreign command. Define a symbol for this command as follows:

```
$ DES3 :== $ disk:[directory]DES3.EXE
```

The options and usage of the program are the same as for the DES example.

5.4.6 ADDIN Example Program

The ADDIN example shows how to provide a new add-in for an existing category of service.

This CDSA example is an add-in (plug-in) module written to the CDSA CSP service provider interface with integrity checking. The add-in would be "loaded" and "attached" by an application, as in the DES examples, using `CSSM_ModuleLoad()`, `CSSM_ModuleAttach()`, and so forth. This example demonstrates the mechanics of developing a CDSA add-in module, which is a shareable image on OpenVMS.

This example also provides the CDSA code files that are necessary to build an add-in module. The installation procedure registers the module in the CDSA MDS database, including its credentials, properties, and capability attributes. It attaches the module and executes `RegisterCDSAModule()` (the definition of INSTALL_ENTRY_NAME).

The files necessary to build the example on OpenVMS are included, with the exception of MODSELFKEY.H. This include file must be generated from the certificate created for the add-in module.

See Section 5.2 for complete instructions on writing a signed application. A signed CDSA application will not execute until the proper credentials are generated.

After you generate the application credentials and the include file MODSELFKEY.H, you can build the ADDIN example program by copying the example files to a local build directory and executing the ADDIN_BUILD command file, as follows:

```
$ DEFINE/TRANS=CONCEALED CDSA_TEMPDIR disk:[directory.]
$ SET DEFAULT CDSA_TEMPDIR:[ADDIN]
$ COPY SYS$SYSROOT:[SYSHLP.EXAMPLES.CDSA.ADDIN]*.* []
$ COPY CDSA_SYSDIR:[SIGN]MODSELFKEY.H []
$ @ADDIN_BUILD
```

The resulting shareable image, STUBCSP300_SHR.EXE, must be signed. On the signing system, run the following command procedure to generate the manifest:

```
$ @ADDIN_SIGN
```

Finally, on the development system, run the command procedure to install the module, as follows:

```
$ @ADDIN_INSTALL
```

The add-in module is now ready to be invoked by an application program.

5.4.7 DUMMY Example Programs

The DUMMYEMM and DUMMYEMMADDIN programs together demonstrate how to provide a new category of service for CDSA. DUMMYEMM, an elective module manager (EMM), contains the logic for handling the generic types of operations for the new service, and the add-in (DUMMYEMMADDIN) contains logic that is specific to the particular operation being performed.

The ADDIN example (see the Section 5.4.6) shows how to provide a new add-in for an existing category of service. DUMMYEMM and DUMMYEMMADDIN are designed to provide an entirely new category of service.

5.4.7.1 DUMMYEMM Example Program

This CDSA example is an elective module manager (EMM) that extends the functionality of CDSA by providing an additional category of service. The example defines a new service provider interface (SPI) with integrity checking.

The purpose of this example is to demonstrate the mechanics of developing a CDSA EMM, which is a shareable image on OpenVMS. The example also provides the CDSA code files that are necessary to build an EMM.

The installation procedure registers the module in the CDSA MDS database, including its credentials, properties, and capability attributes. It attaches the module and executes `RegisterCDSAModule()` (the definition of INSTALL_ENTRY_NAME).

The files necessary to build the example on OpenVMS are included, with the exception of MODSELFKEY.H. This include file must be generated from the certificate created for the add-in module.

Refer to Section 5.2 for complete instructions on writing a signed application. A signed CDSA application will not execute until the proper credentials are generated.

After you generate the application credentials and the include file MODSELFKEY.H, you can build the DUMMYEMM example program by copying the example files to a local build directory and executing the DUMMYEMM_BUILD command file, as follows:

```
$ DEFINE/TRANS=CONCEALED CDSA_TEMPDIR disk:[directory.]
$ SET DEFAULT CDSA_TEMPDIR:[DUMMYEMM]
$ COPY SYS$SYSROOT:[SYSHLP.EXAMPLES.CDSA.DUMMYEMM]*.* []
$ COPY CDSA_SYSDIR:[SIGN]MODSELFKEY.H []
$ @DUMMYEMM_BUILD
```

The resulting shareable image, DUMMYEMM_SHR.EXE, must be signed. On the signing system, run the following command procedure to generate the manifest:

```
$ @DUMMYEMM_SIGN
```

Finally, on the development system, run the command procedure to install the module, as follows:

```
$ @DUMMYEMM_INSTALL
```

When an application program loads an add-in module that is written to the SPI of this EMM, the EMM will be automatically loaded.

5.4.7.2 DUMMYEMMADDIN Example Program

This CDSA example is an elective module manager (EMM) that extends the functionality of CDSA by providing an additional category of service. It provides an add-in module with integrity checking, written to the SPI made available by the DUMMYEMM example.

The purpose of this example is to demonstrate the mechanics of developing a CDSA service provider module for a category of service defined by an EMM. It also provides the necessary CDSA code files that are necessary to build the module.

The installation procedure registers the module in the CDSA MDS database, including its credentials, properties, and capability attributes. It attaches the module and executes `RegisterCDSAModule()` (the definition of INSTALL_ENTRY_NAME).

The files necessary to build the example on OpenVMS are included, with the exception of MODSELFKEY.H. This include file must be generated from the certificate created for the add-in module.

See Section 5.2 for complete instructions on writing a signed application. A signed CDSA application will not execute until the proper credentials are generated.

After you generate the application credentials and the include file MODSELFKEY.H, you can build the DUMMYEMMADDIN example program by copying the example files to a local build area and executing the DUMMYEMMADDIN_BUILD command file, as follows:

```
$ DEFINE/TRANS=CONCEALED CDSA_TEMPDIR disk:[directory.]
$ SET DEFAULT CDSA_TEMPDIR:[DUMMYEMMADDIN]
$ COPY SYS$SYSROOT:[SYSHLP.EXAMPLES.CDSA.DUMMYEMMADDIN]*.* []
$ COPY CDSA_SYSDIR:[SIGN]MODSELFKEY.H []
$ @DUMMYEMMADDIN_BUILD
```

The resulting shareable image, DUMMYEMMADDIN_SHR.EXE, must be signed. On the signing system, run the following command procedure to generate the manifest:

```
$ @DUMMYEMMADDIN_SIGN
```

Finally, on the development system, run the command procedure to install the module, as follows:

```
$ @DUMMYEMMADDIN_INSTALL
```

The add-in module is now ready to be invoked by an application program.

CDSA API Functions

This reference section contains descriptions of the CDSA API functions.

These descriptions are also available from online help. To access help, enter the HELP CDSA command at the system prompt.

The MDSUTIL API functions are a special group of functions described in the following paragraphs.

MDS Utility Library API Functions

Although the MDS API is a required part of any CDSA implementation, the MDSUTIL functions are not. This library of functions was provided with the Intel CDSA reference implementation to encapsulate many common queries that applications typically make to MDS. CDSA on OpenVMS implements the Intel CDSA version of the MDS utility library. Other vendors may supply their own utility libraries built on top of MDS.

To use the MDS utility library, you must include two header files, MDS_UTIL_API.H and MDS_UTIL_HELPER.H, which are in the CDSA_SYDIR:[INCLUDES] directory. You must also link with the library files CDSA$MDS300_SHR.EXE and CDSA$MDS_UTIL_API.OLB, which are located in SYS$SHARE.

The MDS example program provides two special routines for deciphering CDSA error codes within a user program. Because the CDSA include file that specifies error codes (CDSA_SYSDIR:[INCLUDES] CSSMERR.H) does not allow for easy translation from the numeric code to the associated error string, these routines can make the job of debugging a CDSA application easier. These routines are Decode_CDSA_Error and Print_CDSA_Error.

For further information, see the *Intel Common Data Security Architecture Application Developer's Guide,* Chapter 2 (Module Directory Services), under the heading MDS Utility Library.

AC_AuthCompute

NAME

AC_AuthCompute – Compute authorization (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMACI CSSM_AC_AuthCompute
(CSSM_AC_HANDLE ACHandle,
const CSSM_TUPLEGROUP *BaseAuthorizations,
const CSSM_TUPLEGROUP *Credentials,
uint32 NumberOfRequestors,
const CSSM_LIST *Requestors,
const CSSM_LIST *RequestedAuthorizationPeriod,
const CSSM_LIST *RequestedAuthorization,
CSSM_TUPLEGROUP_PTR AuthorizationResult)

SPI:
CSSM_RETURN CSSMACI AC_AuthCompute
(CSSM_AC_HANDLE ACHandle,
const CSSM_TUPLEGROUP *BaseAuthorizations,
const CSSM_TUPLEGROUP *Credentials,
uint32 NumberOfRequestors,
const CSSM_LIST *Requestors,
const CSSM_LIST *RequestedAuthorizationPeriod,
const CSSM_LIST *RequestedAuthorization,
CSSM_TUPLEGROUP_PTR AuthorizationResult)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`ACHandle` (*input*)

The handle that describes the authorization computation module used to perform this function.

`BaseAuthorizations` (*input*)

A pointer to a CSSM_TUPLEGROUP containing at least one ACL certificate, specifying the authorization granted to certain root keys, named entities or combinations thereof. A NULL group of `BaseAuthorizations` always results in a NULL `AuthorizationResult`.

`Credentials` (input/optional)

A pointer to a CSSM_TUPLEGROUP containing a group of certificates, in TUPLE form. The tuple-certificates define the delegation of authorizations from the `BaseAuthorizations` to the `Requestors`. If no additional authorization-granting tuples are provided, then this value is NULL and the `BaseAuthorizations` are the only source of trusted authorizations used as input to the authorization computation.

`NumberOfRequestors` (*input*)

The number of entries in the `Requestors` array.

`Requestors` (*input*)

A pointer to a list of requestors that define the "who" portion of the request. The list can be of type CSSM_LIST_TYPE_SEXPR. Typical exhibits include:

- Public keys
- Hashes of keys
- Hashes of other objects offered for proof.

`RequestedAuthorizationPeriod` (input/optional)

A list defining a validity period or NULL (implying "all time"). This is the "when" portion of the request.

If the list is of type CSSM_LIST_TYPE_SEXPR, then the validity interval is specified as a two-element list containing the values ((not-before <date1>)(not-after <date2 >)). Note that each element is a two-element sublist. The <date> is represented by an ASCII byte-string, in the format (for example) "1998-11-24_15:06:16" and is assumed to be GMT. Open-ended time intervals are specified by omitting either of the interval ends. For example, ((not-before 1997-1-1_00:00:0)) specifies all dates and times beginning on January 1, 1997 going forward indefinitely. For programming convenience, when testing for authorization at a single point in time, the date is represented by a one-element list containing (<date>).

`RequestedAuthorization` (*input*)

A list defining the "what" portion of the authorization being requested.

If the list is of type CSSM_LIST_TYPE_SEXPR, then the list presents an authorization request in SPKI format. If a specific authorization is being requested, then this input is a two-element SEXPR list containing (tag <req>). The valid values for <req> are application-specific. If this is a request to derive all possible authorizations based on the `BaseAuthorizations`, `Credentials`, and `Requestors`, then this input value must be the two-element list containing (tag (*)). This list corresponds to "all authorizations". With this input, the function tests the provided ACL and certificates against the `Requestors` (and possibly `RequestedAuthorizationPeriod`) to yield all authorizations for which the provided `Exhibits` qualify.

`AuthorizationResult` (*output*)

A CSSM_TUPLEGROUP structure, giving the result of the authorization computation. Typically there will be one result, but there could be as many as there are entries in the `BaseAuthorizations`. Each of these results says, in effect: "for this machine, under this ACL and the provided certificates, relative to the specified `Requestors`, the following authorizations have been deduced". Those authorizations are available only on the current platform (and possibly only for the application providing the ACL), and are therefore in the form of an ACL. They are not intended to be used by any other machine or application instance, necessarily, and need to be converted into certificates signed by some private key available to the caller if they are to be so used.

DESCRIPTION

This function performs an authorization computation and returns the results as a group of tuple certificates. The computation is based on the following input values:

`Requestors`

One or more items that identify the requestor. These items are matched against subject fields in `BaseAuthorizations` or `Credentials`. These will be of any form that occurs in an ACL or certificate, and the class of entries is extensible. `AuthCompute` uses these fields to compare against `Subject` fields of TUPLES but does not interpret them, so it does not need to be aware of these extensions. `Requestors`, taken together with `RequestedAuthorization` and `RequestedAuthorizationPeriod`, form request tuples of the form "who requests what, when." `Requestors` can be public keys that verify some signed request, hashes of objects submitted for proof of permission, etc. In general, there will be only one `Requestor`, typically the public key of some keyholder signing a request or authenticating a connection.

`RequestedAuthorization`

The authorization against which the `Requestors` are being tested in this computation.

`RequestedAuthorizationPeriod`

The time range of an authorization computation.

`BaseAuthorizations`

The group of ACL entries (unsigned certificates) provided as the basis for this computation.

`Credentials`

A group of tuple-certificates used with the `BaseAuthorizations` to grant authorizations to the `Requestors`.

Kind of Subject	Example Requestor
Public key	(public-key (rsa-pkcs1-sha1 (e #03#) (n ##)))
Hash of object, key, template, etc.	(hash md5 #900150983cd24fb0d6963f7d28e17f72#)

The most likely `Requestor` is a public key that signs a request. In common practice there will be one Requestor per computation, but it is possible for an ACL or certificate to require multiple signatures or other forms of identification before an action is authorized. In that case, there must be multiple `Requestors`. This function can be used in the following modes:

- To verify the authorization of a specific request, backed up by specific Requestors
- To compute the set of authorizations that a particular set of `Requestors` has been granted by the `BaseAuthorizations` and `Credentials`.

When using this function to verify an authorization, the `RequestedAuthorization` is the specific authorization being requested and the `RequestedAuthorizationPeriod` gives the date and time of that request (typically the current date and time) using both NOT_BEFORE and NOT_AFTER dates. The result, if any, should be an ACL entry with the same authorization that was requested. If such an ACL entry is produced by the computation, then the request is authorized.

Requested Authorization Example
(http http://private.cdsa.hp.com/local-data.html) (ftp ftp://private.cdsa.hp.com/users/cme/private/test.txt write)

Requested Authorization Period Example
(valid (not-before "1999-07-28_17:00:44") (not-after "1999-07-28_17:00:44"))

When using this function to compute the full set of possible authorizations from a set of credentials, rather than to verify a specific access request, the inputs should be of the following form:

- `RequestedAuthorizationPeriod` is either an empty list or the list "`valid`", indicating "all time".
- `RequestedAuthorization` is the list "*", indicating all possible authorizations.

The result of this computation, if any, will be one or more ACL entries representing all the granted authorizations for the indicated requestor.

The scope of ACLs output from this function is limited to the local system. Each ACL should be interpreted to mean: "for this machine, under these base authorization ACLs and the provided certificates, relative to the specified requestors, the following authorizations have been deduced". Those authorizations are available only on the current platform (and possibly only for the application providing the ACL) and are therefore in the form of an ACL. They are not intended to be used by any other machine or application instance. However, the resulting ACLs can be transferred and used outside of the local scope by an entity with authority in the target scope/environment. The transfer and use is a three-step process:

1. Convert the ACL into one or more certificates. The certificates must be signed by some private key with appropriate authority in the target scope/environment.
2. Transfer the certificates to the target environment.
3. Use the signed certificates as input `Credentials` to this function in the target scope/environment.

If the function is successful, check `(*AuthorizationResult)->NumCerts` to determine the precise number of authorizations granted by this computation. If 0, then the `requestors` were not authorized.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_AC_INVALID_BASE_ACLS
CSSMERR_AC_INVALID_ENCODING
CSSMERR_AC_INVALID_REQUESTOR
CSSMERR_AC_INVALID_REQUEST_DESCRIPTOR
CSSMERR_AC_INVALID_TUPLE_CREDENTIALS
CSSMERR_AC_INVALID_VALIDITY_PERIOD

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Reference Pages

Functions for the CSSM API:

CSSM_TP_CertGroupToTupleGroup, *CSSM_TP_TupleGroupToCertGroup*

Functions for the AC SPI:

TP_CertGroupToTupleGroup, *TP_TupleGroupToCertGroup*

AC_PassThrough

NAME

AC_PassThrough: CSSM_AC_PassThrough – Call exported module-specific operations (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_AC_PassThrough
(CSSM_AC_HANDLE ACHandle,
CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DL_DB_LIST *DBList,
uint32 PassThroughId,
const void *InputParams,
void **OutputParams)

SPI:
CSSM_RETURN CSSMACI AC_PassThrough
(CSSM_AC_HANDLE ACHandle,
CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DL_DB_LIST *DBList,
uint32 PassThroughId,
const void *InputParams,
void **OutputParams)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`ACHandle` (*input*)

The handle that describes the authorization computation module used to perform this function.

`TPHandle` (input/optional)

The handle that describes the trust policy module that can be used by the authorization computation service to implement this function. If no trust policy module is specified, the AC module uses an assumed TP module, if required.

`CLHandle` (input/optional)

The handle that describes the add-in certificate library module that can be used to manipulate the subject certificate and anchor certificates. If no certificate library module is specified, the AC module uses an assumed CL module, if required.

`CCHandle` (input/optional)

The handle that describes the cryptographic context containing a handle that describes the add-in Cryptographic Service Provider module that can be used to perform cryptographic operations as required to perform the requested operation. If no cryptographic context is specified, the AC module uses an assumed cryptographic context and CSP module, if required.

`DBList` (input/optional)

A list of handle pairs specifying a data storage library module and a data store managed by that module. These data stores can contain certificates, CRLs, and policy objects for use by the AC module. If no DL and DB handle pairs are specified, the AC module uses an assumed DL module and an assumed data store for this operation.

`PassThroughId` (*input*)

An identifier assigned by the AC module to indicate the exported function to perform.

`InputParams` (*input*)

A pointer to a module, implementation-specific structure containing parameters to be interpreted in a function-specific manner by the requested AC module. If the `passthrough` function requires access to a private key located in the CSP referenced by `CSPHandle`, then `InputParams` should contain a passphrase, or a callback or cryptographic context that can be used to obtain the passphrase.

`OutputParams` (output/optional)

A pointer to a module, implementation-specific structure containing the output data. The service provider will allocate the memory for this structure. The application must free the memory for the structure.

DESCRIPTION

This function allows applications to call authorization computation module-specific operations that have been exported. Such operations might include queries or services specific to the domain represented by the AC module.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_AC_INVALID_CL_HANDLE
CSSMERR_AC_INVALID_CONTEXT_HANDLE
CSSMERR_AC_INVALID_DBLIST_POINTER
CSSMERR_AC_INVALID_DB_LIST
CSSMERR_AC_INVALID_DB_HANDLE
CSSMERR_AC_INVALID_DL_HANDLE
CSSMERR_AC_INVALID_PASSTHROUGH_ID
CSSMERR_AC_INVALID_TP_HANDLE
```

SEE ALSO

Intel CDSA Application Developer's Guide

CDSA_FileValidate

NAME

CDSA_FileValidate: FileValidate – Validate a manifest file against its target file

SYNOPSIS

```
#include <cssm.h>

int CDSA_FileValidate( char         *target_file,
                       CSSM_RETURN  *CDSA_Ret_Status );
```

LIBRARY

CDSA$VALIDATE_LIBSHR.EXE

PARAMETERS

target_file (input)	The full UNIX file specication of the file to be validated.
CDSA_Ret_Status (output)	A CDSA status code. If non-zero, the status can be decoded using the routine `Decode_CDSA_Error`.

DECRIPTION

CDSA_FileValidate validates a target file using the associated manifest file. It is the callable equivalent of CDSA$VALIDATE.EXE.

RETURN VALUE

VMS_Success or VMS_Failure.

ERRORS

Errors are described in the CDSA technical standard.

```
CSSM_OK
CSSM_ERRCODE_SELF_CHECK_FAILED
CSSMERR_SD_NO_TARGETFILE
CSSMERR_SD_NO_MANIFESTFILE
CSSM_ERRCODE_MEMORY_ERROR
CSSMERR_SD_MANIFESTFILE_OPEN_FAILED
CSSMERR_SD_MANIFESTFILE_READ_FAILED
CSSMERR_SD_TARGETFILE_STRING_NOT_FOUND
CSSMERR_SD_TARGETFILE_TERMINATOR_NOT_FOUN
```

CL_CertAbortCache

NAME

CL_CertAbortCache: CSSM_CL_CertAbortCache – Terminate a certificate cache handle (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertAbortCache
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE CertHandle)

SPI:
CSSM_RETURN CSSMAPI CSSM_CL_CertAbortCache
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE CertHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the certificate library module used to perform this function.

`CertHandle` (*input*)

The handle that identifies the cached certificate.

DECRIPTION

This function terminates a certificate cache handle created and returned by the function `CSSM_CL_CertCache()` (CSSM API) or `CL_CertCache()` (CL SPI). The Certificate Library module releases all cache space and state information associated with the cached certificate.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_CACHE_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Reference Pages

Functions for the CSSM API:

CSSM_CL_CertCache

Functions for the CLI SPI:

CL_CertCache

CL_CertAbortQuery

NAME

CL_CertAbortQuery: CSSM_CL_CertAbortQuery function – Terminate a results handle (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertAbortQuery
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle)

SPI:
CSSM_RETURN CSSMAPI CSSM_CL_CertAbortQuery
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`ResultsHandle` (*input*)

A pointer to the handle that identifies the results of a `CSSM_CL_GetFieldValue()` (CSSM API), or `CL_GetFieldValue()` (CLI SPI) request.

DESCRIPTION

This function terminates a results handle used to access multiple certificate fields identified by a single OID. The `ResultsHandle` was created and returned by `CSSM_CL_CertGetFirstFieldValue()` (CSSM API), or `CL_CertGetFirstFieldValue()` (CL SPI), or `CSSM_CL_CertGetFirstCachedFieldValue()` (CSSM API), or `CL_CertGetFirstCachedFieldValue()` (CL SPI).

The CL releases all intermediate state information associated with the repeating-value query. Once this function has been invoked, the results handle is invalid.

Applications must invoke this function to terminate the `ResultsHandle`. Using `CSSM_CL_CertGetNextFieldValue()` (CSSM API), or `CL_CertGetNextFieldValue()` (CL SPI), or `CSSM_CL_CertGetNextCachedFieldValue()` (CSSM API), or `CL_CertGetNextCachedFieldValue()` (CL SPI), to access all of the attributes named by a single OID does not terminate the `ResultsHandle`.

This function can be invoked to terminate the results handle without accessing all values identified by the single OID.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_RESULTS_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CertGetFirstFieldValue, *CSSM_CL_CertGetNextFieldValue*, *CSSM_CL_CertGetFirstCachedFieldValue*, *CSSM_CL_CertGetNextCachedFieldValue*

Functions for the CLI SPI:

CL_CertGetFirstFieldValue, *CL_CertGetNextFieldValue*, *CL_CertGetFirstCachedFieldValue*, *CL_CertGetNextCachedFieldValue*

CL_CertCache

NAME

CL_CertCache: CSSM_CL_CertCache – Cache a copy of a certificate (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertCache
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Cert,
CSSM_HANDLE_PTR CertHandle)
SPI:
CSSM_RETURN CSSMAPI CSSM_CL_CertCache
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Cert,
CSSM_HANDLE_PTR CertHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the certificate library module used to perform this function.

`Cert` (*input*)

A pointer to the CSSM_DATA structure containing the encoded certificate.

`CertHandle` (*output*)

A pointer to the CSSM_HANDLE that should be used in all future references to the cached certificate.

DESCRIPTION

This function caches a copy of a certificate for subsequent accesses using the functions `CSSM_CL_CertGetFirstCachedFieldValue()` (CSSM API), or `CL_CertGetFirstCachedFieldValue()` (CL SPI), and `CSSM_CL_CertGetNextCachedFieldValue()` (CSSM API), or `CL_CertGetNextCachedFieldValue()` (CL SPI).

The input certificate must be in an encoded representation. The Certificate Library module can cache the certificate in any appropriate internal representation. Parsed or incrementally parsed representations are common. The selected representation is opaque to the caller.

The application must call `CSSM_CL_CertAbortCache()` (CSSM API), or `CL_CertAbortCache()` (CL SPI), to remove the cached copy when additional `get` operations will not be performed on the cached certificate.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_CERT_POINTER`
`CSSMERR_CL_UNKNOWN_FORMAT`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CertGetFirstCachedFieldValue, *CSSM_CL_CertGetNextCachedFieldValue*, *CSSM_CL_CertAbortQuery*, *CSSM_CL_CertAbortCache*

Functions for the CLI SPI:

CL_CertGetFirstCachedFieldValue, *CL_CertGetNextCachedFieldValue*, *CL_CertAbortQuery*, *CL_CertAbortCache*

CL_CertCreateTemplate

NAME

CL_CertCreateTemplate: CSSM_CL_CertCreateTemplate – Allocate and initialize memory for a certificate template (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertCreateTemplate
(CSSM_CL_HANDLE CLHandle,
uint32 NumberOfFields,
const CSSM_FIELD *CertFields,
CSSM_DATA_PTR CertTemplate)
SPI:
CSSM_RETURN CSSMCLI CL_CertCreateTemplate
(CSSM_CL_HANDLE CLHandle,
uint32 NumberOfFields,
const CSSM_FIELD *CertFields,
CSSM_DATA_PTR CertTemplate)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the certificate library module used to perform this function.

`NumberOfFields` (*input*)

The number of certificate field values specified in the `CertFields`.

`CertFields` (*input*)

A pointer to an array of OID/value pairs that identify the field values to initialize a new certificate.

`CertTemplate` (*output*)

A pointer to a CSSM_DATA structure that will contain the unsigned certificate template as a result of this function.

DESCRIPTION

This function allocates and initializes memory for an encoded certificate template output in `CertTemplate->Data`. The template values are specified by the input OID/value pairs contained in `CertFields`. The initialization process includes encoding all certificate field values according to the certificate type and certificate encoding supported by the certificate library module.

The memory for `CertTemplate->Data` is allocated by the service provider using the calling application's memory management routines. The application must deallocate the memory.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CL_INVALID_FIELD_POINTER
CSSMERR_CL_UNKNOWN_TAG
CSSMERR_CL_INVALID_NUMBER_OF_FIELDS
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CertGetAllTemplateFields, *CSSM_CL_CertSign*

Functions for the CLI SPI:

CL_CertGetAllTemplateFields, *CL_CertSign*

CL_CertDescribeFormat

NAME

CL_CertDescribeFormat: CSSM_CL_CertDescribeFormat – Return a list of the CSSM_OID values (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertDescribeFormat
(CSSM_CL_HANDLE CLHandle,
uint32 *NumberOfOids,
CSSM_OID_PTR *OidList)
SPI:
CSSM_RETURN CSSMAPI CSSM_CL_CertDescribeFormat
(CSSM_CL_HANDLE CLHandle,
uint32 *NumberOfOids,
CSSM_OID_PTR *OidList)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`NumberOfOids` (*output*)

The length of the returned array of OIDs.

`OidList` (*output*)

A pointer to the array of CSSM_OIDs that represent the supported certificate format. The OID list is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function returns a list of the CSSM_OID values this certificate library module uses to name and reference fields of a certificate. Multiple CSSM_OID values can correspond to a single field. These OIDs can be provided as input to `CSSM_CL_CertGetFirstFieldValue()` (CSSM API), or `CL_CertGetFirstFieldValue()` (CL SPI), to retrieve field values from the certificate. The OID also implies the data format of the returned value. When multiple OIDs name the same field of a certificate, those OIDs have different return data formats associated with them.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CertGetAllFields, *CSSM_CL_CertGetFirstFieldValue*, *CSSM_CL_CertGetFirstCachedFieldValue*

Functions for the CLI SPI:

CL_CertGetAllFields, *CL_CertGetFirstFieldValue*, *CL_CertGetFirstCachedFieldValue*

CL_CertGetAllFields

NAME

CL_CertGetAllFields: CSSM_CL_CertGetAllFields – Return a list of input certificate values (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertGetAllFields
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Cert,
uint32 *NumberOfFields,
CSSM_FIELD_PTR *FieldList)
SPI:
CSSM_RETURN CSSMCLI CL_CertGetAllFields
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Cert,
uint32 *NumberOfFields,
CSSM_FIELD_PTR *FieldList)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`Cert` (*input*)

A pointer to the CSSM_DATA structure containing the certificate whose fields will be returned.

`NumberOfFields` (*output*)

The length of the returned array of fields.

`FieldList` (*output*)

A pointer to an array of CSSM_FIELD structures that contain the values of all fields of the input certificate. The field list is allocated by the service provider and must be deallocated by the application by calling `CSSM_CL_FreeFields()` (CSSM API), or `CL_FreeFields()` (CL SPI).

DESCRIPTION

This function returns a list of the values stored in the input certificate.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_CERT_POINTER`
`CSSMERR_CL_UNKNOWN_FORMAT`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

For the CSSM API:

CSSM_CL_CertGetFirstFieldValue, *CSSM_CL_CertDescribeFormat*, *CSSM_CL_FreeFields*

For the CLI SPI:

CL_CertGetFirstFieldValue, *CL_CertDescribeFormat*, *CL_FreeFields*

CL_CertGetAllTemplateFields

NAME

CL_CertGetAllTemplateFields: CSSM_CL_CertGetAllTemplateFields – Extract and return values stored in CertTemplate (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertGetAllTemplateFields
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *CertTemplate,
uint32 *NumberOfFields,
CSSM_FIELD_PTR *CertFields)
SPI:
CSSM_RETURN CSSMCLI CL_CertGetAllTemplateFields
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *CertTemplate,
uint32 *NumberOfFields,
CSSM_FIELD_PTR *CertFields)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the certificate library module used to perform this function.

`CertTemplate` (*input*)

A pointer to the CSSM_DATA structure containing the packed, encoded certificate template.

`NumberOfFields` (*output*)

The length of the output array of fields.

`CertFields` (*output*)

A pointer to an array of CSSM_FIELD structures which contains the OIDs and values of the fields of the input certificate template.

DESCRIPTION

This function extracts and returns a copy of the values stored in the encoded `CertTemplate`. The memory for the `CertFields` output is allocated by the service provider using the calling application's memory management routines. The application must deallocate the memory by calling `CSSM_CL_FreeFields()` (CSSM API), or `CL_FreeFields()` (CL SPI).

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_UNKNOWN_FORMAT`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_FreeFields, *CSSM_CL_CertCreateTemplate*

Functions for the CLI SPI:

CL_FreeFields, *CL_CertCreateTemplate*

CL_CertGetFirstCachedFieldValue

NAME

CL_CertGetFirstCachedFieldValue: CSSM_CL_CertGetFirstCachedFieldValue – Return values from the cached certificate (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertGetFirstCachedFieldValue
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE CertHandle,
const CSSM_OID *CertField,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedFields,
CSSM_DATA_PTR *FieldValue)
SPI:
CSSM_RETURN CSSMAPI CSSM_CL_CertGetFirstCachedFieldValue
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE CertHandle,
const CSSM_OID *CertField,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedFields,
CSSM_DATA_PTR *FieldValue)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`CertHandle` (*input*)

A handle identifying a certificate that the application has temporarily cached with the Certificate Library module. The referenced certificate is searched for the field value named by `CertField`.

`CertField` (*input*)

A pointer to an object identifier that identifies the field value to be extracted from the Cert.

`ResultsHandle` (*output*)

A pointer to the CSSM_HANDLE that should be used to obtain any additional matching fields.

`NumberOfMatchedFields` (*output*)

The total number of fields that match the `CertField` OID. This count includes the first match, which was returned by this function.

`FieldValue(output)`

A pointer to the structure containing the value of the requested field. The structure and the field at I "(*FieldValue)->Data" are allocated by the service provider. The `CSSM_CL_FreeFieldValue()` (CSSM API), or `CL_FreeFieldValue()` (CL SPI), function can be used to deallocate `FieldValue` and `(*FieldValue)->Data`.

DESCRIPTION

This function returns a single structure containing a set of field values from the cached certificate identified by `CertHandle`. The selected fields are designated by the CSSM_OID CertField and returned in the output parameter `FieldValue`. The OID also identifies the data format of the values returned to the caller. If multiple OIDs designate the same certificate field, then each such OID defines a distinct data format for the returned values. The function `CSSM_CL_CertDescribeFormat()` (CSSM API), or `CL_CertDescribeFormat()` (CL SPI), provides a list of all CSSM_OID values supported by a certificate library module for naming fields of a certificate.

The `CertField` OID can identify a single occurrence of a set of certificate fields, or multiple occurrences of a set of certificate fields. If the `CertField` OID matches more than one occurrence, this function outputs the total number of matches and a `ResultsHandle` to be used as input to `CSSM_CertGetNextCachedFieldValue()` (CSM API), or `CertGetNextCachedFieldValue()` (CL SPI), to retrieve the remaining matches. The first match is returned as the return value of this function.

This function determines the complete set of matches. The number of matches and the selected field values do not change between this function and subsequent calls to `CSSM_CL_CertGetNextCachedFieldValue()` (CSSM API), or `CL_CertGetNextCachedFieldValue()` (CL SPI).

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_CACHE_HANDLE`
`CSSMERR_CL_UNKNOWN_TAG`
`CSSMERR_CL_NO_FIELD_VALUES`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CertGetNextCachedFieldValue, *CSSM_CL_CertAbortCache*, *CSSM_CL_CertAbortQuery*, *CSSM_CL_CertGetAllFields*, *CSSM_CL_CertDescribeFormat*, *CSSM_CL_FreeFieldValue*

Functions for the CLI SPI:

CL_CertGetNextCachedFieldValue, *CL_CertAbortCache*, *CL_CertAbortQuery*, *CL_CertGetAllFields*, *CL_CertDescribeFormat*, *CL_FreeFieldValue*

CL_CertGetFirstFieldValue

NAME

CL_CertGetFirstFieldValue: CSSM_CL_CertGetFirstFieldValue – Return the value of the certificate field (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertGetFirstFieldValue
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Cert,
const CSSM_OID *CertField,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedFields,
CSSM_DATA_PTR *Value)
SPI:
CSSM_RETURN CSSMCLI CL_CertGetFirstFieldValue
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Cert,
const CSSM_OID *CertField,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedFields,
CSSM_DATA_PTR *Value)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`Cert` (*input*)

A pointer to the CSSM_DATA structure containing the certificate.

`CertField` (*input*)

A pointer to an object identifier which identifies the field value to be extracted from the `Cert`.

`ResultsHandle` (*output*)

A pointer to the CSSM_HANDLE that should be used to obtain any additional matching fields.

`NumberOfMatchedFields` (*output*)

The total number of fields that match the `CertField` OID. This count includes the first match, which was returned by this function.

`Value` (*output*)

A pointer to the structure containing the value of the requested field. The structure and the field at I "(*Value)->Data" are allocated by the service provider. The `CSSM_CL_FreeFieldValue()` (CSSM API) or `CL_FreeFieldValue()` (CL SPI) function can be used to deallocate `*Value` and `(*Value)->Data`.

DESCRIPTION

This function returns the value of the certificate field designated by the CSSM_OID `CertField`. The OID also identifies the data format for the field value returned to the caller. If multiple OIDs name the same certificate field, then each such OID defines a distinct data format for the returned field value. The function `CSSM_CL_CertDescribeFormat()` (CSSM API), or `CL_CertDescribeFormat()` (CL SPI), provides a list of all CSSM_OID values supported by a certificate library module for naming fields of a certificate.

If more than one field matches the `CertField` OID, the first matching field will be returned. The number of matching fields is an output parameter, as is the `ResultsHandle` to be used to retrieve the remaining matching fields.

The set of matching fields is determined by this function. The number of matching fields and the field values do not change between this function and subsequent calls to `CSSM_CL_CertGetNextFieldValue()` (CSSM API), or `CL_CertGetNextFieldValue()` (CL SPI).

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_CERT_POINTER`
`CSSMERR_CL_UNKNOWN_FORMAT`
`CSSMERR_CL_UNKNOWN_TAG`
`CSSMERR_CL_NO_FIELD_VALUES`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CertGetNextFieldValue, *CSSM_CL_CertAbortQuery*, *CSSM_CL_CertGetAllField*, *CSSM_CL_FreeFieldValue*, *CSSM_CL_CertDescribeFormat*, *CSSM_CL_FreeFieldValue*

Functions for the CLI SPI:

CL_CertGetNextFieldValue, *CL_CertAbortQuery*, *CL_CertGetAllField*, *CL_FreeFieldValue*, *CL_CertDescribeFormat*, *CL_FreeFieldValue*

CL_CertGetKeyInfo

NAME

CL_CertGetKeyInfo: CSSM_CL_CertGetKeyInfo – Return the public key and integral information (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertGetKeyInfo
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Cert,
CSSM_KEY_PTR *Key)
SPI:
CSSM_RETURN CSSMCLI CL_CertGetKeyInfo
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Cert,
CSSM_KEY_PTR *Key)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`Cert` (*input*)

A pointer to the CSSM_DATA structure containing the certificate from which to extract the public key information.

`Key` (*output*)

A pointer to the CSSM_KEY structure containing the public key and possibly other key information. The CSSM_KEY structure and its substructures are allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function returns the public key and integral information about the key from the specified certificate. The key structure returned is a compound object. It can be used in any function requiring a key, such as creating a cryptographic context.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_CERT_POINTER`
`CSSMERR_CL_UNKNOWN_FORMAT`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CertGetFirstFieldValue, *CSSM_CL_FreeFieldValue*

Functions for the CLI SPI:

CL_CertGetFirstFieldValue, *CL_FreeFieldValue*

CL_CertGetNextCachedFieldValue

NAME

CSSM_CL_CertGetNextCachedFieldValue – Return the value of a certificate field (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertGetNextCachedFieldValue
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle,
CSSM_DATA_PTR *FieldValue)
SPI:
CSSM_RETURN CSSMAPI CSSM_CL_CertGetNextCachedFieldValue
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle,
CSSM_DATA_PTR *FieldValue)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the certificate library module used to perform this function.

`ResultsHandle` (*input*)

The handle that identifies the results of a certificate query.

`FieldValue` (*output*)

A pointer to the structure containing the value of the requested field. The structure and the field at I "(*FieldValue)->Data" are allocated by the service provider. The `CSSM_CL_FreeFieldValue()` (CSSM API), or `CL_FreeFieldValue()` (CL SPI) function can be used to deallocate `*FieldValue` and `(*FieldValue)->Data`.

DESCRIPTION

This function returns the value of a certificate field, when that field occurs multiple times in a certificate. Certificates with repeated fields (such as multiple signatures) have multiple field values corresponding to a single OID. A call to the function `CSSM_CL_CertGetFirstCachedFieldValue()` (CSSM API), or `CL_CertGetFirstCachedFieldValue()` (CL SPI), returns a `ResultsHandle` identifying the size and values contained in the result set. The `CSSM_CL_CertGetNextCachedFieldValue()` (CSSMAPI), or `CL_CertGetNextCachedFieldValue()` (CL SPI), function can be called repeatedly to obtain these values, one at a time. The result set does not change in size or value between calls to this function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_RESULTS_HANDLE`
`CSSMERR_CL_NO_FIELD_VALUES`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CertGetFirstCachedFieldValue, *CSSM_CL_CertAbortCache*, *CSSM_CL_CertAbortQuery*, *CSSM_CL_CertGetAllFields*, *CSSM_CL_CertDescribeFormat*, *CSSM_CL_FreeFieldValue*

Functions for the CLI SPI:

CL_CertGetFirstCachedFieldValue, *CL_CertAbortCache*, *CL_CertAbortQuery*, *CL_CertGetAllFields*, *CL_CertDescribeFormat*, *CL_FreeFieldValue*

CL_CertGetNextFieldValue

NAME

CL_CertGetNextFieldValue: CSSM_CL_CertGetNextFieldValue – Return the value of a certificate field (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertGetNextFieldValue
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle,
CSSM_DATA_PTR *Value)
SPI:
CSSM_RETURN CSSMCLI CL_CertGetNextFieldValue
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle,
CSSM_DATA_PTR *Value)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`ResultsHandle` (*input*)

The handle that identifies the results of a certificate query.

`Value` (*output*)

A pointer to the structure containing the value of the requested field. The structure and the field at I "(*Value)->Data" are allocated by the service provider. The `CSSM_CL_FreeFieldValue()` (CSSM API) or `CL_FreeFieldValue()` (CL SPI), function can be used to deallocate `*Value` and `(*Value)->Data`.

DESCRIPTION

This function returns the value of a certificate field, when that field occurs multiple times in a certificate. Certificates with repeated fields (such as multiple signatures) have multiple field values corresponding to a single OID. A call to the function `CSSM_CL_CertGetFirstFieldValue()` (CSSM API), or `CL_CertGetFirstFieldValue()` (CL SPI), returns a results handle identifying the size and values contained in the result set. The `CSSM_CL_CertGetNextFieldValue()` (CSSM API), or `CL_CertGetNextFieldValue()` (CL SPI), function can be called repeatedly to obtain these values, one at a time. The result set does not change in size or value between calls to this function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_RESULTS_HANDLE`
`CSSMERR_CL_NO_FIELD_VALUES`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CertGetFirstFieldValue, *CSSM_CL_CertAbortQuery*, *CSSM_CL_FreeFieldValue*

Functions for the CLI SPI:

CL_CertGetFirstFieldValue, *CL_CertAbortQuery*, *CL_FreeFieldValue*

CL_CertGroupFromVerifiedBundle

NAME

CL_CertGroupFromVerifiedBundle: CSSM_CL_CertGroupFromVerifiedBundle – Verify the signature of a bundle (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertGroupFromVerifiedBundle
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CERT_BUNDLE *CertBundle,
const CSSM_DATA *SignerCert,
CSSM_CERTGROUP_PTR *CertGroup)
SPI:
CSSM_RETURN CSSMCLI CL_CertGroupFromVerifiedBundle
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CERT_BUNDLE *CertBundle,
const CSSM_DATA *SignerCert,
CSSM_CERTGROUP_PTR *CertGroup)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`CCHandle` (input/optional)

The handle of the cryptographic context to control the verification operation.

`CertBundle` (*input*)

A structure containing a reference to a signed, encoded bundle of certificates and to descriptors of the type and encoding of the bundle. The bundled certificates are to be separated into a certificate group (list of individual encoded certificates). If the bundle type and bundle encoding are not specified, the add-in module might either attempt to decode the bundle assuming a default type and encoding or might immediately fail.

`SignerCert` (input/optional)

The certificate to be used to verify the signature on the certificate bundle. If the bundle is signed but this field is not specified, then the module will assume a default certificate for verification.

`CertGroup` (*output*)

A pointer to the certificate group, represented as an array of individual, encoded certificates. The certificate group and CSSM_CERTGROUP substructures are allocated by the serivce provider and must be deallocated by the application. The group contains all certificates contained in the certificate bundle.

DESCRIPTION

This function accepts as input a certificate bundle (a codified and signed aggregation of the certificates in the group), verifies the signature of the bundle (if a signature is present), and returns a certificate group (as an array of individual certificates) including every certificate contained in the bundle. The signature on the certificate aggregate is verified using the cryptographic context and possibly using the input signer certificate. The CL module embeds the knowledge of the verification scope for the bundle types that it supports. A CL module's supported bundle types and encodings are available to applications by querying the CSSM registry. The type and encoding of the certificate bundle must be specified with the input bundle. If signature verification is successful, the certificate aggregate will be parsed into a certificate group whose order corresponds to the certificate aggregate ordering. This certificate group will then be returned to the calling application.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CL_INVALID_CONTEXT_HANDLE
CSSMERR_CL_INVALID_BUNDLE_POINTER
CSSMERR_CL_INVALID_BUNDLE_INFO
CSSMERR_CL_INVALID_CERT_POINTER
CSSMERR_CL_INVALID_CERTGROUP_POINTER
CSSMERR_CL_UNKNOWN_FORMAT
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CertGroupToSignedBundle

Functions for the CLI SPI:

CL_CertGroupToSignedBundle

CL_CertGroupToSignedBundle

NAME

CL_CertGroupToSignedBundle: CSSM_CL_CertGroupToSignedBundle – Convert a certificate group to a certificate bundle (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertGroupToSignedBundle
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CERTGROUP *CertGroupToBundle,
const CSSM_CERT_BUNDLE_HEADER *BundleInfo,
CSSM_DATA_PTR SignedBundle)
SPI:
CSSM_RETURN CSSMCLI CL_CertGroupToSignedBundle
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CERTGROUP *CertGroupToBundle,
const CSSM_CERT_BUNDLE_HEADER *BundleInfo,
CSSM_DATA_PTR SignedBundle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`CCHandle` (input/optional)

The handle of the cryptographic context to control the signing operation. The operation will fail if a signature is required for this type of bundle and the cryptographic context is not valid.

`CertGroupToBundle` (*input*)

An array of individual, encoded certificates. All certificates in this list will be included in the resulting certificate bundle.

`BundleInfo` (input/optional)

A structure containing the type and encoding of the bundle to be created. If the type and the encoding are not specified, then the module will use a default bundle type and bundle encoding.

`SignedBundle` (*output*)

The function returns a pointer to a signed certificate bundle containing all certificates in the certificate group. The bundle is of the type and encoding requested by the caller or is the default type defined by the library module if the `BundleInfo` was not specified by the caller. The `SignedBundle->Data` is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function accepts as input a certificate group (as an array of individual certificates) and returns a certificate bundle (a codified and signed aggregation of the certificates in the group). The certificate group will first be encoded according to the `BundleInfo` input by the user. If `BundleInfo` is NULL, the library will perform a default encoding for its default bundle type. If possible, the certificate group ordering will be maintained in this certificate aggregate encoding. After encoding, the certificate aggregate will be signed using the input context. The CL module embeds knowledge of the signing scope for the bundle types it supports. The signature is then associated with the certificate aggregate according to the bundle type and encoding rules and is returned as a bundle to the calling application.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CL_INVALID_CONTEXT_HANDLE
CSSMERR_CL_INVALID_CERTGROUP_POINTER
CSSMERR_CL_INVALID_CERT_POINTER
CSSMERR_CL_UNKNOWN_FORMAT
CSSMERR_CL_INVALID_BUNDLE_POINTER
CSSMERR_CL_INVALID_BUNDLE_INFO
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CertGroupFromVerifiedBundle

Functions for the CLI SPI:

CL_CertGroupFromVerifiedBundle

CL_CertSign

NAME

CSSM_CL_CertSign – Sign a certificate (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertSign
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CertTemplate,
const CSSM_FIELD *SignScope,
uint32 ScopeSize,
CSSM_DATA_PTR SignedCert)
SPI:
CSSM_RETURN CSSMCLI CL_CertSign
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CertTemplate,
const CSSM_FIELD *SignScope,
uint32 ScopeSize,
CSSM_DATA_PTR SignedCert)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`CCHandle` (*input*)

A signature context defining the CSP, signing algorithm, and private key that must be used to perform the operation. The passphrase for the private key is also provided.

`CertTemplate` (*input*)

A pointer to a CSSM_DATA structure containing a certificate template in the default format supported by this CL. The template contains values that are currently contained in or will be contained in a signed certificate.

`SignScope` (input/optional)

A pointer to the CSSM_FIELD array containing the OID/value pairs of the fields to be signed. A null input signs all the fields provided by `CertTemplate`.

`ScopeSize` (*input*)

The number of entries in the `SignScope` list. If the sign scope is not specified, the input value for scope size must be zero.

`SignedCert` (*output*)

A pointer to the CSSM_DATA structure containing the signed certificate.

DESCRIPTION

This function signs a certificate using the private key and signing algorithm specified in the `CCHandle`. The result is a signed, encoded certificate in `SignedCert`. The certificate field values are specified in the input certificate template. The template is constructed using `CSSM_CL_CertCreateTemplate()` (CSSM API), or `CL_CertCreateTemplate()` (CL SPI). The template is in the default format for this CL.

The `CCHandle` must be a signature context created using the function `CSSM_CSP_CreateSignatureContext()` (CSSM API), or `CSP_CreateSignatureContext()` (SPI). The context must specify the Cryptographic Services Provider (CSP) module, the signing algorithm, and the signing key that must be used to perform this operation. The context must also provide the passphrase or a callback function to obtain the passphrase required to access and use the private key.

The fields included in the signing operation are identified by the OIDs in the optional `SignScope` array.

The memory for the `SignedCert->Data` output is allocated by the service provider using the calling application's memory management routines. The application must deallocate the memory.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CL_INVALID_CONTEXT_HANDLE
CSSMERR_CL_UNKNOWN_FORMAT
CSSMERR_CL_INVALID_FIELD_POINTER
CSSMERR_CL_UNKNOWN_TAG
CSSMERR_CL_INVALID_SCOPE
CSSMERR_CL_INVALID_NUMBER_OF_FIELDS
CSSMERR_CL_SCOPE_NOT_SUPPORTED
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CertVerify, *CSSM_CL_CertCreateTemplate*

Functions for the CLI SPI:

CL_CertVerify, *CL_CertCreateTemplate*

CL_CertVerify

NAME

CL_CertVerify: CSSM_CL_CertVerify – Verify a signed certificate (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CertVerify
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CertToBeVerified,
const CSSM_DATA *SignerCert,
const CSSM_FIELD *VerifyScope,
uint32 ScopeSize)
SPI:
CSSM_RETURN CSSMAPI CSSM_CL_CertVerify
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CertToBeVerified,
const CSSM_DATA *SignerCert,
const CSSM_FIELD *VerifyScope,
uint32 ScopeSize)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`CCHandle` (input/optional)

The handle that describes the context of this cryptographic operation.

`CertToBeVerified` (*input*)

A pointer to the CSSM_DATA structure with a certificate containing at least one signature for verification. An unsigned certificate template cannot be verified.

`SignerCert` (input/optional)

A pointer to the CSSM_DATA structure containing the certificate used to sign the subject certificate. This certificate provides the public key to use in the verification process and if the certificate being verified contains multiple signatures, the signer's certificate indicates which signature is to be verified.

`VerifyScope` (input/optional)

A pointer to the CSSM_FIELD array containing the tag/value pairs of the fields to be used in verifying the signature. (This should include all fields that were used to calculate the signature.) If the verify scope is null, the certificate library module assumes that its default set of certificate fields were used to calculate the signature, and those same fields are used in the verification process.

`ScopeSize` (*input*)

The number of entries in the verify scope list. If the verification scope is not specified, the input value for scope size must be zero.

DESCRIPTION

This function verifies that the signed certificate has not been altered since it was signed by the designated signer. Only one signature is verified by this function. If the certificate to be verified includes multiple signatures, this function must be applied once for each signature to be verified. This function verifies a digital signature over the certificate fields specified by `VerifyScope`. If the verification scope fields are not specified, the function performs verification using a preselected set of fields in the certificate.

The caller can specify a Cryptographic Service Provider (CSP) and verification algorithm that the CL can use to perform the verification. The handle for the CSP is contained in the cryptographic context identified by `CCHandle`.

The verification process requires that the caller must specify the necessary verification algorithm parameters. These parameter values are specified in one of two locations:

- As a field value in the `SignerCert` parameter
- As a set of algorithm parameters contained in the cryptographic context identified by `CCHandle`

If both of the preceding arguments are supplied, a consistency check is performed to ensure that they result in the same verification algorithm parameters. If they are not consistent, an error is returned. If only one of the above arguments is supplied, that argument is used to generate the verification algorithm parameters. If no algorithm parameters are found, the certificate cannot be verified and the operation fails.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CL_INVALID_CONTEXT_HANDLE
CSSMERR_CL_INVALID_CERT_POINTER
CSSMERR_CL_UNKNOWN_FORMAT
CSSMERR_CL_INVALID_FIELD_POINTER
CSSMERR_CL_UNKNOWN_TAG
CSSMERR_CL_INVALID_SCOPE
CSSMERR_CL_INVALID_NUMBER_OF_FIELDS
CSSMERR_CL_SCOPE_NOT_SUPPORTED
CSSMERR_CL_VERIFICATION_FAILURE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CertSign

Functions for the CLI SPI:

CL_CertSign

CL_CertVerifyWithKey

NAME

CL_CertVerifyWithKey: CSSM_CL_CertVerifyWithKey – Verify with a key (CDSA)

SYNOPSIS

```
# include <cssm.h>
```

API:

```
CSSM_RETURN CSSMAPI CSSM_CL_CertVerifyWithKey
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CertToBeVerified)
```

SPI:

```
CSSM_RETURN CSSMCLI CL_CertVerifyWithKey
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CertToBeVerified)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the certificate library service module used to perform this function.

`CCHandle` (*input*)

A signature verification context defining the CSP, verification algorithm, and public key that must be used to perform the operation.

`CertToBeVerified` (*input*)

A signed certificate whose signature is to be verified.

DESCRIPTION

This function verifies that the `CertToBeVerified` parameter was signed using a specific private key and that the certificate has not been altered since it was signed using that private key. The public key corresponding to the private signing key is used in the verification process.

The `CCHandle`, must be a signature verification context created using the function `CSSM_CSP_CreateSignatureContext()` (CSSM API), or `CSP_CreateSignatureContext()` (SPI). The context must specify the Cryptographic Services Provider (CSP) module, the verification algorithm, and the public verification key that must be used to perform this operation.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CL_INVALID_CONTEXT_HANDLE
CSSMERR_CL_INVALID_CERT_POINTER
CSSMERR_CL_UNKNOWN_FORMAT
CSSMERR_CL_VERIFICATION_FAILURE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CertVerify, *CSSM_CL_CrlVerify*

Functions for the CLI SPI:

CL_CertVerify, *CL_CrlVerify*

CL_CrlAbortCache

NAME

CL_CrlAbortCache: CSSM_CL_CrlAbortCache – Terminate a CRL cache handle (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlAbortCache
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE CrlHandle)
SPI:
CSSM_RETURN CSSMCLI CL_CrlAbortCache
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE CrlHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the certificate library module used to perform this function.

`CrlHandle` (*input*)

The handle that identifies the cached CRL.

DESCRIPTION

This function terminates a CRL cache handle created and returned by the function `CSSM_CL_CrlCache()` (CSSM API), or `CL_CrlCache()` (CL SPI). The Certificate Library module releases all cache space and state information associated with the cached CRL.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_CACHE_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlCache

Functions for the CLI SPI:

CL_CrlCache

CL_CrlAbortQuery

NAME

CL_CrlAbortQuery: CSSM_CL_CrlAbortQuery – Terminate a query (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlAbortQuery
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle)
SPI:
CSSM_RETURN CSSMCLI CL_CrlAbortQuery
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`ResultsHandle` (*input*)

The handle that identifies the results of a CRL query.

DESCRIPTION

This function terminates the query initiated by `CSSM_CL_CrlGetFirstFieldValue()` or `CSSM_CL_CrlGetFirstCachedFieldValue()` function (or their CL SPI equivalents), and allows the CL to release all intermediate state information associated with the repeating-value `get` operation. Once this function has been invoked, the results handle is invalid.

Applications must invoke this function to terminate the `ResultsHandle`. Using `CSSM_CL_CrlGetNextFieldValue()` or `CSSM_CL_CrlGetNextCachedFieldValue()` (or their CL SPI equivalents), to access all of the attributes named by a single OID does not terminate the `ResultsHandle`. This function can be invoked to terminate the results handle without accessing all of the values identified by the single OID.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CL_INVALID_RESULTS_HANDLE

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlGetFirstFieldValue, *CSSM_CL_CrlGetNextFieldValue*, *CSSM_CL_CrlGetFirstCachedFieldValue*, *CSSM_CL_CrlGetNextCachedFieldValue*

Functions for the CL SPI:

CL_CrlGetFirstFieldValue, *CL_CrlGetNextFieldValue*, *CL_CrlGetFirstCachedFieldValue*, *CL_CrlGetNextCachedFieldValue*

CL_CrlAddCert

NAME

CL_CrlAddCert: CSSM_CL_CrlAddCert – Revoke an input certificate (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlAddCert
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *Cert,
uint32 NumberOfFields,
const CSSM_FIELD *CrlEntryFields,
const CSSM_DATA *OldCrl,
CSSM_DATA_PTR NewCrl)
SPI:
CSSM_RETURN CSSMCLI CL_CrlAddCert
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *Cert,
uint32 NumberOfFields,
const CSSM_FIELD *CrlEntryFields,
const CSSM_DATA *OldCrl,
CSSM_DATA_PTR NewCrl)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation.

`Cert` (*input*)

A pointer to the CSSM_DATA structure containing the certificate to be revoked.

`NumberOfFields` (*input*)

The number of OID/value pairs specified in the CrlEntryFields input parameter.

`CrlEntryFields` (*input*)

An array of OID/value pairs specifying the initial values for descriptive data fields of the new CRL entry.

`OldCrl` (*input*)

A pointer to the CSSM_DATA structure containing the CRL to which the newly revoked certificate will be added.

`NewCrl` (*output*)

A pointer to the CSSM_DATA structure containing the updated CRL. The `NewCrl->Data` is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function revokes the input certificate by adding a record representing the certificate to the CRL. The values for the new entry in the CRL are specified by the list of OID/value input pairs. The reason for revocation is a typical value specified in the list. The new CRL entry is signed using the private key and signing algorithm specified in the `CCHandle`.

The `CCHandle` must be a context created using the function `CSSM_CSP_CreateSignatureContext()` (CSSM API), or `CSP_CreateSignatureContext()` (CL SPI). The context must specify the Cryptographic Services Provider (CSP) module, the signing algorithm, and the signing key that must be used to perform this operation. The context must also provide the passphrase or a callback function to obtain the passphrase required to access and use the private key.

The operation is valid only if the CRL has not been closed by the process of signing the CRL, by calling `CSSM_CL_CrlSign()` (CSSM API), or `CL_CrlSign()` (CL SPI). Once the CRL has been signed, entries cannot be added or removed.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_CONTEXT_HANDLE`
`CSSMERR_CL_INVALID_CERT_POINTER`
`CSSMERR_CL_UNKNOWN_FORMAT`
`CSSMERR_CL_INVALID_FIELD_POINTER`
`CSSMERR_CL_UNKNOWN_TAG`
`CSSMERR_CL_INVALID_NUMBER_OF_FIELDS`
`CSSMERR_CL_INVALID_CRL_POINTER`
`CSSMERR_CL_CRL_ALREADY_SIGNED`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlRemoveCert

Functions for the CLI SPI:

CL_CrlRemoveCert

CL_CrlCache

NAME

CL_CrlCache: CSSM_CL_CrlCache – Cache a copy of a certificate revocation list (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlCache
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Crl,
CSSM_HANDLE_PTR CrlHandle)
SPI:
CSSM_RETURN CSSMCLI CL_CrlCache
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Crl,
CSSM_HANDLE_PTR CrlHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the certificate library module used to perform this function.

`Crl` (*input*)

A pointer to the CSSM_DATA structure containing the encoded CRL.

`CrlHandle` (*output*)

A pointer to the CSSM_HANDLE that should be used in all future references to the cached CRL.

DESCRIPTION

This function caches a copy of a `CertificateRevocationList` (CRL) for subsequent accesses using the functions `CSSM_CL_CrlGetFirstFieldValue()` and `CSSM_CL_CrlGetNextFieldValue()` (or their CL SPI equivalents).

The input CRL must be in an encoded representation. The Certificate Library module can cache the CRL in any appropriate internal representation. Parsed or incrementally parsed representations are common. The selected representation is opaque to the caller.

The application must call `CSSM_CL_CrlCacheAbort()` (CSSM API), or `CL_CrlCacheAbort()` (CL SPI), to remove the cached copy when additional `get` operations will not be performed on the cached CRL.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_CRL_POINTER`
`CSSMERR_CL_UNKNOWN_FORMAT`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlGetFirstCachedFieldValue, *CSSM_CL_CrlGetNextCachedFieldValue*, *CSSM_CL_IsCertInCachedCrl*, *CSSM_CL_CrlAbortQuery*, *CSSM_CL_CrlAbortCache*

Functions for the CLI SPI:

CL_CrlGetFirstCachedFieldValue, *CL_CrlGetNextCachedFieldValue*, *CL_IsCertInCachedCrl*, *CL_CrlAbortQuery*, *CL_CrlAbortCache*

CL_CrlCreateTemplate

NAME

CL_CrlCreateTemplate: CSSM_CL_CrlCreateTemplate – Create an unsigned, memory-resident CRL (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlCreateTemplate
(CSSM_CL_HANDLE CLHandle,
uint32 NumberOfFields,
const CSSM_FIELD *CrlTemplate,
CSSM_DATA_PTR NewCrl)
SPI:
CSSM_RETURN CSSMCLI CL_CrlCreateTemplate
(CSSM_CL_HANDLE CLHandle,
uint32 NumberOfFields,
const CSSM_FIELD *CrlTemplate,
CSSM_DATA_PTR NewCrl)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`NumberOfFields` (*input*)

The number of OID/value pairs specified in the CrlTemplate input parameter.

`CrlTemplate` (*input*)

An array of OID/value pairs specifying the initial values for descriptive data fields of the new CRL.

`NewCrl` (*output*)

A pointer to the CSSM_DATA structure containing the new CRL. The `NewCrl-> Data` is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function creates an unsigned, memory-resident CRL. Fields in the CRL are initialized with the descriptive data specified by the OID/value input pairs. The specified OID/value pairs can initialize all or a subset of the general attribute fields in the new CRL. Subsequent values can be set using the `CSSM_CL_CrlSetFields()` (CSSM API) or the `CL_CrlSetFields()` (CL SPI) function. The new CRL contains no revocation records.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CL_INVALID_FIELD_POINTER
CSSMERR_CL_UNKNOWN_TAG
CSSMERR_CL_INVALID_NUMBER_OF_FIELDS
CSSMERR_CL_INVALID_CRL_POINTER

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlSetFields, *CSSM_CL_CrlAddCert*, *CSSM_CL_CrlSign*, *CSSM_CL_CertGetFirstFieldValue*

Functions for the CLI SPI:

CL_CrlSetFields, *CL_CrlAddCert*, *CL_CrlSign*, *CL_CertGetFirstFieldValue*

CL_CrlDescribeFormat

NAME

CL_CrlDescribeFormat: CSSM_CL_CrlDescribeFormat – Return a list of the CSSM_OID values (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlDescribeFormat
(CSSM_CL_HANDLE CLHandle,
uint32 *NumberOfOids,
CSSM_OID_PTR *OidList)
SPI:
CSSM_RETURN CSSMCLI CL_CrlDescribeFormat
(CSSM_CL_HANDLE CLHandle,
uint32 *NumberOfOid,
CSSM_OID_PTR *OidList)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`NumberOfOids` (*output*)

The length of the returned array of OIDs.

`OidList` (*output*)

A pointer to the array of CSSM_OIDs that represent the supported CRL format. The OID list is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function returns a list of the CSSM_OID values the Certificate Library module uses to name and reference fields of a CRL. Multiple CSSM_OID values can correspond to a single field. These OIDs can be provided as input to `CSSM_CL_CrlGetFirstFieldValue()` (CSSM API), or `CL_CrlGetFirstFieldValue()` (CL SPI), calls to retrieve field values from the CRL. The OID also implies the data format of the returned value. When multiple OIDs name the same field of a CRL, those OIDs have different return data formats associated with them.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Intel CDSA Application Developer's Guide

CL_CrlGetAllCachedRecordFields

NAME

CL_CrlGetAllCachedRecordFields: CSSM_CL_CrlGetAllCachedRecordFields – Return field values from a CRL record (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlGetAllCachedRecordFields
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE CrlHandle,
const CSSM_DATA *CrlRecordIndex,
uint32 *NumberOfFields,
CSSM_FIELD_PTR *Fields)
SPI:
CSSM_RETURN CSSMCLI CL_CrlGetAllCachedRecordFields
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE CrlHandle,
const CSSM_DATA *CrlRecordIndex,
uint32 *NumberOfFields,
CSSM_FIELD_PTR *Fields)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in certificate library module used to perform this function.

`CrlHandle` (*input*)

A handle identifying a CRL that the application has temporarily cached with the Certificate Library module. The referenced CRL must contain the CRL record identified by `CrlRecordIndex`.

`CrlRecordIndex` (*input*)

An index value identifying a particular revocation record in a cached CRL.

`NumberOfFields` (*output*)

The number of OID-value pairs returned by this function.

`Fields` (*output*)

A pointer to an array of CSSM_FIELD structures, describing the contents of the preselected CRL record using OID-value pairs. The field list is allocated by the service provider and must be deallocated by the application by calling `CSSM_CL_FreeFields()` (CSSM API), or `CL_FreeFields()` (CL SPI).

DESCRIPTION

This function returns all field values from a prelocated, cached CRL record. The scanned CRL record is identified by `CrlRecordIndex`, which is returned by the function `CSSM_CL_IsCertInCachedCrl()` (CSSM API), or `CL_IsCertInCachedCrl()` (CL SPI).

Fields are returned as a set of OID-value pairs. The OID identifies the CRL record field and the data format of the value extracted from that field. The Certificate Library module defines and uses a preferred data format for returning field values in this function.

Each CRL record may be digitally signed when it is added to the CRL using the function `CSSM_CL_CrlAddCert()` (CSSM API), or `CL_CrlAddCert()` (CL SPI). This function does not perform any signature verification on the CRL record.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_CACHE_HANDLE`
`CSSMERR_CL_INVALID_CRL_INDEX`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_IsCertInCachedCrl, *CSSM_CL_CrlCache*, *CSSM_CL_CrlAbortCache*, *CSSM_CL_FreeFields*

Functions for the CLI SPI:

CL_IsCertInCachedCrl, *CL_CrlCache*, *CL_CrlAbortCache*, *CL_FreeFields*

CL_CrlGetAllFields

NAME

CL_CrlGetAllFields: CSSM_CL_CrlGetAllFields – Get the field values from the CRL (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlGetAllFields
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Crl,
uint32 *NumberOfCrlFields,
CSSM_FIELD_PTR *CrlFields)
SPI:
CSSM_RETURN CSSMCLI CL_CrlGetAllFields
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Crl,
uint32 *NumberOfCrlFields,
CSSM_FIELD_PTR *CrlFields)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`Crl` (*input*)

A pointer to the CSSM_DATA structure that contains the encoded, packed CRL from which field values are to be extracted.

`NumberOfCrlFields` (*output*)

The number of entries in the array `CrlFields`.

`CrlFields` (*output*)

A pointer to an array of OID-value pairs that describe the contents of the CRL. The extracted CRL fields are returned as the value portion of each OID-value pair. The field list is allocated by the service provider and must be deallocated by the application by calling `CSSM_CL_FreeFields()` (CSSM API), or `CL_FreeFields()` (CL SPI).

DESCRIPTION

This function returns one or more structures. Each structure contains a set of field values from the encoded, packed CRL contained in `Crl`. Each structure is returned in the `FieldValue` entry of the CSSM_FIELD structure `CrlFields`. The parameter `NumberOfCrlFields` indicates the number of returned structures.

The CRL can be signed or unsigned. This function does not perform any signature verification on the CRL fields or the CRL records. Each CRL record can be digitally signed when it is added to the CRL using the function `CSSM_CL_CrlAddCert()` (CSSM API), or `CL_CrlAddCert()` (CL SPI).

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_CRL_POINTER`
`CSSMERR_CL_UNKNOWN_FORMAT`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_FreeFields

Functions for the CLI SPI:

CL_FreeFields

CL_CrlGetFirstCachedFieldValue

NAME

CL_CrlGetFirstCachedFieldValue: CSSM_CL_CrlGetFirstCachedFieldValue – Get field values from the cached CRL (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlGetFirstCachedFieldValue
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE CrlHandle,
const CSSM_DATA *CrlRecordIndex,
const CSSM_OID *CrlField,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedFields,
CSSM_DATA_PTR *FieldValue)
SPI:
CSSM_RETURN CSSMCLI CL_CrlGetFirstCachedFieldValue
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE CrlHandle,
const CSSM_DATA *CrlRecordIndex,
const CSSM_OID *CrlField,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedFields,
CSSM_DATA_PTR *FieldValue)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`CrlHandle` (*input*)

A handle identifying a CRL that the application has temporarily cached with the Certificate Library module. The referenced CRL is searched for the field values identified by `CrlField`.

`CrlRecordIndex` (input/optional)

An index value identifying a particular revocation record in a cached CRL. If an index value is supplied, the scan for the field values identified by `CrlField` is limited to the preselected revocation record.

`CrlField` (*input*)

A pointer to an object identifier that identifies the field value to be extracted from the CRL.

`ResultsHandle` (*output*)

A pointer to the CSSM_HANDLE that should be used to obtain any additional matching fields.

`NumberOfMatchedFields` (*output*)

The total number of fields that match the `CrlField` OID. This count includes the first match, which was returned by this function.

`FieldValue` (*output*)

A pointer to the structure containing the value of the requested field. The structure and the field at `I "(*FieldValue)->Data"` are allocated by the service provider. The `CSSM_CL_FreeFieldValue()` (CSSM API), or `CL_FreeFieldValue()` (CL SPI), function can be used to deallocate `*FieldValue` and `(*FieldValue)->Data`.

DESCRIPTION

This function returns a single structure containing a set of field values from the cached CRL identified by `CrlHandle` parameter. The selected fields are designated by the CSSM_OID `CrlField` parameter and returned in the output parameter `FieldValue`. The OID also identifies the data format of the values returned to the caller. If multiple OIDs designate the same CRL field, then each such OID defines a distinct data format for the returned values. The function `CSSM_CL_CrlDescribeFormat()` (CSSM API), or `CL_CrlDescribeFormat()` (CL SPI), provides a list of all CSSM_OID values supported by a CL module for naming fields of a CRL.

The search can be limited to a particular revocation record within the CRL. A single record is identified by the `CrlRecordIndex` parameter, which is returned by the function `CSSM_CL_IsCertInCachedCrl()` (CSSM API), or `CL_IsCertInCachedCrl()` (CL SPI). If no record index is supplied, the search is initiated from the beginning of the CRL.

The CRL can be signed or unsigned. This function does not perform any signature verification on the CRL fields or the CRL records. Each CRL record can be digitally signed when it is added to the CRL using the function `CSSM_CL_CrlAddCert()` (CSSM API), or `CL_CrlAddCert()` (CL SPI). The caller can examine fields in the CRL and CRL records at any time using this function.

The `CrlField` OID can identify a single occurrence of a set of CRL fields or multiple occurrences of a set of CRL fields. If the `CrlField` OID matches more than one occurrence, this function outputs the total number of matches and a `ResultsHandle` to be used as input to `CSSM_CrlGetNextFieldValue()` (CSSM API), or `CrlGetNextFieldValue()` (CL SPI), to retrieve the remaining matches. The first match is returned as the return value of this function.

This function determines the complete set of matches. The number of matches and the selected field values do not change between this function and subsequent calls to `CSSM_CL_CrlGetNextFieldValue()` (CSSM API), or `CL_CrlGetNextFieldValue()` (CL SPI).

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CL_INVALID_CACHE_HANDLE
CSSMERR_CL_INVALID_CRL_INDEX
CSSMERR_CL_UNKNOWN_TAG
CSSMERR_CL_NO_FIELD_VALUES
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlGetNextCachedFieldValue, *CSSM_CL_IsCertInCachedCrl*, *CSSM_CL_CrlAbortQuery*, *CSSM_CL_CrlCache*, *CSSM_CL_CrlAbortCache*, *CSSM_CL_CrlDescribeFormat*, *CSSM_CL_FreeFieldValue*

Functions for the CLI SPI:

CL_CrlGetNextCachedFieldValue, *CL_IsCertInCachedCrl*, *CL_CrlAbortQuery*, *CL_CrlCache*, *CL_CrlAbortCache*, *CL_CrlDescribeFormat*, *CL_FreeFieldValue*

CL_CrlGetFirstFieldValue

NAME

CL_CrlGetFirstFieldValue: CSSM_CL_CrlGetFirstFieldValue – Get the value of the first CRL field (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlGetFirstFieldValue
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Crl,
const CSSM_OID *CrlField,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedFields,
CSSM_DATA_PTR *Value)
SPI:
CSSM_RETURN CSSMCLI CL_CrlGetFirstFieldValue
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Crl,
const CSSM_OID *CrlField,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedFields,
CSSM_DATA_PTR *Value)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`Crl` (*input*)

A pointer to the CSSM_DATA structure that contains the CRL from which the field is to be retrieved.

`CrlField` (*input*)

An object identifier that identifies the field value to be extracted from the CRL.

`ResultsHandle` (*output*)

A pointer to the CSSM_HANDLE that should be used to obtain any additional matching fields.

`NumberOfMatchedFields` (*output*)

The total number of fields that match the `CrlField` OID. This count includes the first match, which was returned by this function.

`Value` (*output*)

A pointer to the structure containing the value of the requested field. The structure and the field at `I "(*Value)->Data"` are allocated by the service provider. The `CSSM_CL_FreeFieldValue()` (CSSM API), or `CL_FreeFieldValue()` (CL SPI), function can be used to deallocate `*Value` and `(*Value)->Data`.

DESCRIPTION

This function returns the value of the CRL field designated by the CSSM_OID `CrlField`. The OID also identifies the data format for the field value returned to the caller. If multiple OIDs name the same CRL field, then each OID defines a distinct data format for the returned field value. The function `CSSM_CL_CrlDescribeFormat()` (CSSM API), or `CL_CrlDescribeFormat()` (CL SPI), provides a list of all CSSM_OID values supported by a Certificate Library module for naming fields of a CRL.

If more than one field matches the `CrlField` OID, the first matching field will be returned. The number of matching fields is an output parameter, as is the `ResultsHandle` to be used to retrieve the remaining matching fields.

The set of matching fields is determined by this function. The number of matching fields and the field values do not change between this function and subsequent calls to `CSSM_CL_CrlGetNextFieldValue()` (CSSM API), or `CL_CrlGetNextFieldValue()` (CL SPI).

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_CRL_POINTER`
`CSSMERR_CL_UNKNOWN_FORMAT`
`CSSMERR_CL_UNKNOWN_TAG`
`CSSMERR_CL_NO_FIELD_VALUES`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlGetNextFieldValue, *CSSM_CL_CrlAbortQuery*, *CSSM_CL_CrlGetAllFields*

Functions for the CLI SPI:

CL_CrlGetNextFieldValue, *CL_CrlAbortQuery*, *CL_CrlGetAllFields*

CL_CrlGetNextCachedFieldValue

NAME

CL_CrlGetNextCachedFieldValue: CSSM_CL_CrlGetNextCachedFieldValue – Get the value of the next cached CRL field (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlGetNextCachedFieldValue
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle,
CSSM_DATA_PTR *FieldValue)
SPI:
CSSM_RETURN CSSMCLI CL_CrlGetNextCachedFieldValue
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle,
CSSM_DATA_PTR *FieldValue)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`ResultsHandle` (*input*)

The handle that identifies the results of a CRL query.

`FieldValue` (*output*)

A pointer to the structure containing the value of the requested field. The structure and the field at `I "(*FiledValue)->Data"` are allocated by the service provider. The `CSSM_CL_FreeFieldValue()` (CSSM API), or `CL_FreeFieldValue()` (CL SPI), function can be used to deallocate `*FieldValue` and `(*FieldValue)->Data`.

DESCRIPTION

This function returns the value of a CRL field, when that field occurs multiple times in a CRL. CRLs with repeated fields (such as revocation records) have multiple field values corresponding to a single OID. A call to the function `CSSM_CL_CrlGetFirstCachedFieldValue()` (CSSM API), or `CL_CrlGetFirstCachedFieldValue()` (CL SPI), initiates the process and returns a `ResultsHandle` identifying the size and values contained in the result set. The `CSSM_CL_CrlGetNextCachedFieldValue()` (CSSM API), or `CL_CrlGetNextCachedFieldValue()` (CL SPI), function can be called repeatedly to obtain these values, one at a time. The result set does not change in size or value between calls to this function.

The result set selected by `CSSM_CL_CrlGetFirstCachedFieldValue()` (CSSM API), or `CL_CrlGetFirstCachedFieldValue()` (CL SPI), and identified by `ResultsHandle` can reference CRL fields repeated across multiple revocation records or within one revocation record. The scope of the scan was set by an optional `CrlRecordIndex` input to the function `CSSM_CL_CrlGetFirstCachedFieldValue()` (CSSM API), or `CL_CrlGetFirstCachedFieldValue()` (CL SPI). If the record index was specified, then the results set is the revocation record identified by the index. If no record index was specified, then the results set can include repeated fields from multiple revocation records in a CRL.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_RESULTS_HANDLE`
`CSSMERR_CL_NO_FIELD_VALUES`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlGetFirstCachedFieldValue, *CSSM_CL_CrlAbortQuery*, *CSSM_CL_IsCertInCachedCrl*, *CSSM_CL_CrlCache*, *CSSM_CL_CrlAbortCache*, *CSSM_CL_FreeFieldValue*

Functions for the CLI SPI:

CL_CrlGetFirstCachedFieldValue, *CL_CrlAbortQuery*, *CL_IsCertInCachedCrl*, *CL_CrlCache*, *CL_CrlAbortCache*, *CL_FreeFieldValue*

CL_CrlGetNextFieldValue

NAME

CL_CrlGetNextFieldValue: CSSM_CL_CrlGetNextFieldValue – Get the value of the next CRL field (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlGetNextFieldValue
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle,
CSSM_DATA_PTR *Value)
SPI:
CSSM_RETURN CSSMCLI CL_CrlGetNextFieldValue
(CSSM_CL_HANDLE CLHandle,
CSSM_HANDLE ResultsHandle,
CSSM_DATA_PTR *Value)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`ResultsHandle` (*input*)

The handle that identifies the results of a CRL query.

`Value` (*output*)

A pointer to the structure containing the value of the requested field. The structure and the field at `I "(*Value)->Data"` are allocated by the service provider. The `CSSM_CL_FreeFieldValue()` (CSSM API), or `CL_FreeFieldValue()` (CL SPI), function can be used to deallocate `*Value` and `(*Value)->Data`.

DESCRIPTION

This function returns the value of a CRL field, when that field occurs multiple times in a CRL. CRLs with repeated fields (such as revocation records) have multiple field values corresponding to a single OID. A call to the function `CSSM_CL_CrlGetFirstFieldValue()` (CSSM API), or `CL_CrlGetFirstFieldValue()` (CL SPI), initiates the process and returns a results handle identifying the size and values contained in the result set. The `CSSM_CL_CrlGetNextFieldValue()` (CSSM API), or `CL_CrlGetNextFieldValue()` (CL SPI), function can be called repeatedly to obtain these values, one at a time. The result set does not change in size or value between calls to this function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_RESULTS_HANDLE`
`CSSMERR_CL_NO_FIELD_VALUES`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlGetFirstFieldValue, *CSSM_CL_CrlAbortQuery*

Functions for the CLI SPI:

CL_CrlGetFirstFieldValue, *CL_CrlAbortQuery*

CL_CrlRemoveCert

NAME

CL_CrlRemoveCert: CSSM_CL_CrlRemoveCert – Reinstate a certificate (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlRemoveCert
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Cert,
const CSSM_DATA *OldCrl,
CSSM_DATA_PTR NewCrl)
SPI:
CSSM_RETURN CSSMCLI CL_CrlRemoveCert
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Cert,
const CSSM_DATA *OldCrl,
CSSM_DATA_PTR NewCrl)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`Cert` (*input*)

A pointer to the CSSM_DATA structure containing the certificate to be reinstated.

`OldCrl` (*input*)

A pointer to the CSSM_DATA structure containing the CRL from which the certificate is to be removed.

`NewCrl` (*output*)

A pointer to the CSSM_DATA structure containing the updated CRL. The `NewCrl->Data` is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function reinstates a certificate by removing it from the specified CRL. The operation is valid only if the CRL has not been closed by the process of signing the CRL by executing `CSSM_CL_CrlSign()`(CSSM API), or `CL_CrlSign()` (CL SPI). Once the CRL has been signed, entries cannot be added or removed.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CL_INVALID_CERT_POINTER
CSSMERR_CL_INVALID_CRL_POINTER
CSSMERR_CL_UNKNOWN_FORMAT
CSSMERR_CL_CRL_ALREADY_SIGNED

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlAddCert

Functions for the CLI SPI:

CL_CrlAddCert

CL_CrlSetFields

NAME

CL_CrlSetFields: CSSM_CL_CrlSetFields – Set new field values (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlSetFields
(CSSM_CL_HANDLE CLHandle,
uint32 NumberOfFields,
const CSSM_FIELD *CrlTemplate,
const CSSM_DATA *OldCrl,
CSSM_DATA_PTR ModifiedCrl)
SPI:
CSSM_RETURN CSSMCLI CL_CrlSetFields
(CSSM_CL_HANDLE CLHandle,
uint32 NumberOfFields,
const CSSM_FIELD *CrlTemplate,
const CSSM_DATA *OldCrl,
CSSM_DATA_PTR ModifiedCrl)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`NumberOfFields` (*input*)

The number of OID value pairs specified in the CrlTemplate input parameter.

`CrlTemplate` (*input*)

Any array of field OID value pairs containing the values to initialize the CRL attribute fields.

`OldCrl` (*input*)

The CRL to be updated with the new attribute values. The CRL must be unsigned and available for update.

`ModifiedCrl` (*output*)

A pointer to the modified, unsigned CRL. The `ModifiedCrl->Data` is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function will set the fields of the input CRL to the new values, specified by the input OID/value pairs. If there is more than one possible instance of an OID (for example, as in an extension or CRL record), then a new field with the specified value is added to the CRL.

This function should be used to update any of the CRL field values. If a specified field was initialized by `CSSM_CL_CrlCreateTemplate()` (CSSM API), or `CL_CrlCreateTemplate()` (CL SPI), the field value is set to the new specified value. If a specified field was not initialized by the `CSSM_CL_CrlCreateTemplate()` (CSSM API), or `CL_CrlCreateTemplate()` (CL SPI), the field is set to the new specified value. The `OldCrl` must be unsigned. Once a CRL has been signed using `CSSM_CL_CrlSign()` (CSSM API), or `CL_CrlSign()` (CL SPI), the signed CRL's field values cannot be modified. Modification would invalidate the cryptographic signature of the CRL.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_FIELD_POINTER`
`CSSMERR_CL_UNKNOWN_TAG`
`CSSMERR_CL_INVALID_NUMBER_OF_FIELDS`
`CSSMERR_CL_UNKNOWN_FORMAT`
`CSSMERR_CL_INVALID_CRL_POINTER`
`CSSMERR_CL_CRL_ALREADY_SIGNED`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlCreateTemplate, *CSSM_CL_CrlAddCert*, *CSSM_CL_CrlSign*, *CSSM_CL_CertGetFirstFieldValue*

Functions for the CLI SPI:

CL_CrlCreateTemplate, *CL_CrlAddCert*, *CL_CrlSign*, *CL_CertGetFirstFieldValue*

CL_CrlSign

NAME

CL_CrlSign: CSSM_CL_CrlSign, CL_CrlSign – Sign a CRL (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlSign
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *UnsignedCrl,
const CSSM_FIELD *SignScope,
uint32 ScopeSize,
CSSM_DATA_PTR SignedCrl)
SPI:
CSSM_RETURN CSSMCLI CL_CrlSign
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *UnsignedCrl,
const CSSM_FIELD *SignScope,
uint32 ScopeSize,
CSSM_DATA_PTR SignedCrl)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation.

`UnsignedCrl` (*input*)

A pointer to the CSSM_DATA structure containing the CRL to be signed.

`SignScope` (input/optional)

A pointer to the CSSM_FIELD array containing the tag/value pairs of the fields to be signed. If the signing scope is null, the Certificate Library module includes a default set of CRL fields in the signing process.

`ScopeSize` (*input*)

The number of entries in the sign scope list. If the signing scope is not specified, the input scope size must be zero.

`SignedCrl` (*output*)

A pointer to the CSSM_DATA structure containing the signed CRL. The `SignedCrl->Data` is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function signs a CRL using the private key and signing algorithm specified in the `CCHandle` parameter. The result is a signed, encoded certificate revocation list in `SignedCrl`. The unsigned CRL is specified in the input `UnsignedCrl`. The `UnsignedCrl` is constructed using the `CSSM_CL_CrlCreateTemplate()`, `CSSM_CL_CrlSetFields()`, `CSSM_CL_CrlAddCert()`, and `CSSM_CL_CrlRemoveCert()` functions (for the CSSM API), or their CL SPI equivalents.

The `CCHandle` must be context created using the function `CSSM_CSP_CreateSignatureContext()` (CSSM API), or `CSP_CreateSignatureContext()` (SPI). The context must specify the Cryptographic Services Provider module, the signing algorithm, and the signing key that must be used to perform this operation. The context must also provide the passphrase or a callback function to obtain the passphrase required to access and use the private key.

The fields included in the signing operation are identified by the OIDs in the optional `SignScope` array.

Once the CRL has been signed it cannot be modified. This means that entries cannot be added or removed from the CRL through application of the `CSSM_CL_CrlAddCert()` or `CSSM_CL_CrlRemoveCertCSSM_CL_CrlRemoveCert()` (or their CL SPI equivalent operations. A signed CRL can be verified, applied to a data store, and searched for values.

The memory for the `SignedCrl->Data` output is allocated by the service provider using the calling application's memory management routines. The application must deallocate the memory.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CL_INVALID_CONTEXT_HANDLE
CSSMERR_CL_INVALID_CRL_POINTER
CSSMERR_CL_UNKNOWN_FORMAT
CSSMERR_CL_INVALID_FIELD_POINTER
CSSMERR_CL_UNKNOWN_TAG
CSSMERR_CL_INVALID_SCOPE
CSSMERR_CL_SCOPE_NOT_SUPPORTED
CSSMERR_CL_INVALID_NUMBER_OF_FIELDS
CSSMERR_CL_CRL_ALREADY_SIGNED
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

Functions:

CSSM_CL_CrlVerify, *CSSM_CL_CrlVerifyWithKey*

Functions for the CLI SPI:

CL_CrlVerify, *CL_CrlVerifyWithKey*

CL_CrlVerify

NAME

CL_CrlVerify: CSSM_CL_CrlVerify – Verify a signed CRL has not been altered (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlVerify
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CrlToBeVerified,
const CSSM_DATA *SignerCert,
const CSSM_FIELD *VerifyScope,
uint32 ScopeSize)
SPI:
CSSM_RETURN CSSMCLI CL_CrlVerify
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CrlToBeVerified,
const CSSM_DATA *SignerCert,
const CSSM_FIELD *VerifyScope,
uint32 ScopeSize)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`CCHandle` (input/optional)

The handle that describes the context of this cryptographic operation.

`CrlToBeVerified` (*input*)

A pointer to the CSSM_DATA structure containing the CRL to be verified.

`SignerCert` (input/optional)

A pointer to the CSSM_DATA structure containing the certificate used to sign the CRL.

`VerifyScope` (input/optional)

A pointer to the CSSM_FIELD array containing the tag/value pairs of the fields to be verified. If the verification scope is null, the Certificate Library module assumes that a default set of fields were used in the signing process and those same fields are used in the verification process.

`ScopeSize` (*input*)

The number of entries in the verify scope list. If the verification scope is not specified, the input value for scope size must be zero.

DESCRIPTION

This function verifies that the signed CRL has not been altered since it was signed by the designated signer. It does this by verifying the digital signature over the fields specified by the `VerifyScope` parameter.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CL_INVALID_CONTEXT_HANDLE
CSSMERR_CL_INVALID_CERT_POINTER
CSSMERR_CL_INVALID_CRL_POINTER
CSSMERR_CL_UNKNOWN_FORMAT
CSSMERR_CL_INVALID_FIELD_POINTER
CSSMERR_CL_UNKNOWN_TAG
CSSMERR_CL_INVALID_SCOPE
CSSMERR_CL_INVALID_NUMBER_OF_FIELDS
CSSMERR_CL_SCOPE_NOT_SUPPORTED
CSSMERR_CL_VERIFICATION_FAILURE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlSign

Functions for the CLI SPI:

CL_CrlSign

CL_CrlVerifyWithKey

NAME

CL_CrlVerifyWithKey: CSSM_CL_CrlVerifyWithKey – Verify a CRL with a specific key (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_CrlVerifyWithKey
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CrlToBeVerified)
SPI:
CSSM_RETURN CSSMCLI CL_CrlVerifyWithKey
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CrlToBeVerified)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the Certificate Library service module used to perform this function.

`CCHandle` (*input*)

A signature verification context defining the Cryptographic Services Provider (CSP), verification algorithm, and public key that must be used to perform the operation.

`CrlToBeVerified` (*input*)

A signed certificate revocation list whose signature is to be verified.

DESCRIPTION

This function verifies that the `CrlToBeVerified` parameter was signed using a specific private key and that the certificate revocation list has not been altered since it was signed using that private key. The public key corresponding to the private signing key is used in the verification process.

The cryptographic context indicated by the `CCHandle` parameter must be a signature verification context created using the function `CSSM_CSP_CreateSignatureContext()` (CSSM API) or `CSP_CreateSignatureContext()` (CL SPI). The context must specify the Cryptographic Services Provider (CSP) module, the verification algorithm, and the public verification key that must be used to perform this operation.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CL_INVALID_CONTEXT_HANDLE
CSSMERR_CL_INVALID_CRL_POINTER
CSSMERR_CL_UNKNOWN_FORMAT
CSSMERR_CL_VERIFICATION_FAILURE

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlVerify

Functions for the CLI SPI:

CL_CrlVerify

CL_FreeFields

NAME

CL_FreeFields: CSSM_CL_FreeFields – Free fields (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_FreeFields
(CSSM_CL_HANDLE CLHandle,
uint32 NumberOfFields,
CSSM_FIELD_PTR *FieldArray)
SPI:
CSSM_RETURN CSSMCLI CL_FreeFields
(CSSM_CL_HANDLE CLHandle,
uint32 NumberOfFields,
CSSM_FIELD_PTR *FieldArray)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`NumberOfFields` (*input*)

The length of the array of fields in `FieldArray`.

`FieldArray` (*input*)

A pointer to an array of CSSM_FIELD structures that need to be deallocated.

DEFINITIONS

This function frees the fields in the `FieldArray` by freeing the data pointers for both the `FieldOid` and `FieldValue` fields. It also frees the top level `FieldArray` pointer.

This function should be used only to free CSSM_FIELD_PTR values returned from calls `CSSM_TP_CertGetAllTemplateFields()`, `CSSM_CL_CertGetAllTemplateFields()`, `CSSM_CL_CertGetAllFields()`, `CSSM_CL_CrlGetAllFields()`, `CSSM_CL_CrlGetAllCachedRecordFields()`, or their SPI equivalent calls.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Intel CDSA Application Developer's Guide

CL_FreeFieldValue

NAME

CL_FreeFieldValue: CSSM_CL_FreeFieldValue – Free field data (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_FreeFieldValue
(CSSM_CL_HANDLE CLHandle,
const CSSM_OID *CertOrCrlOid,
CSSM_DATA_PTR Value)
SPI:
CSSM_RETURN CSSMCLI CL_FreeFieldValue
(CSSM_CL_HANDLE CLHandle,
const CSSM_OID *CertOrCrlOid,
CSSM_DATA_PTR Value)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`CertOrCrlOid` (*input*)

A pointer to the CSSM_OID structure describing the type of the `Value` to be freed.

`Value` (*input*)

A pointer to the CSSM_DATA structure containing the `Data` to be freed.

DESCRIPTION

This function frees the data specified by `Value` and `Value->Data`. `CertOrCrlOid` indicates the type of the data in `Value`.

This function should be used only to free CSSM_DATA values returned from calls `CSSM_CL_CertGetFirstFieldValue()`, `CSSM_CL_CertGetNextFieldValue()`, `CSSM_CL_CertGetFirstCachedFieldValue()`, `CSSM_CL_CertGetNextCachedFieldValue()`, `CSSM_CL_CrlGetFirstFieldValue()`, `CSSM_CL_CrlGetNextFieldValue()`, `CSSM_CL_CrlGetFirstCachedFieldValue()`, `CSSM_CL_CrlGetNextCachedFieldValue()`, or their CLI SPI equivalents.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_UNKNOWN_TAG`

SEE ALSO

Intel CDSA Application Developer's Guide

CL_IsCertInCachedCrl

NAME

CL_IsCertInCachedCrl: CSSM_CL_IsCertInCachedCrl – Search cached CRL for a record (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_IsCertInCachedCrl
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Cert,
CSSM_HANDLE CrlHandle,
CSSM_BOOL *CertFound,
CSSM_DATA_PTR CrlRecordIndex)
SPI:
CSSM_RETURN CSSMCLI CL_IsCertInCachedCrl
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Cert,
CSSM_HANDLE CrlHandle,
CSSM_BOOL *CertFound,
CSSM_DATA_PTR CrlRecordIndex)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`Cert` (*input*)

A pointer to the CSSM_DATA structure containing an encoded, packed certificate.

`CrlHandle` (*input*)

A handle identifying a CRL that the application has temporarily cached with the Certificate Library module. The referenced CRL is searched for a revocation record matching the specified `Cert`.

`CertFound` (*output*)

A pointer to a CSSM_BOOL indicating success or failure in finding the specified certificate in the CRL. CSSM_TRUE signifies that the certificate was found in the CRL. CSSM_FALSE indicates that the certificate was not found in the CRL.

`CrlRecordIndex` (*output*)

A pointer to a CSSM_DATA structure containing an index descriptor for direct access to the located CRL record. `CrlRecordIndex->Data` is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function searches the cached CRL for a record corresponding to the certificate. The result of the search is returned in `CertFound`. The CRL and the records within the CRL must be digitally signed. This function does not verify either signature. The caller should use `CSSM_TP_CrlVerify()` or `CSSM_CL_CrlVerify()` (or their SPI equivalents) before invoking this function. Once the CRL has been verified, the caller can invoke this function repeatedly without repeating the verification process.

If the certificate is found in the CRL, the CL module returns an index descriptor `CrlRecordIndex` for use with other Certificate Library CRL functions. The index provides more direct access to the selected CRL record.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CL_INVALID_CERT_POINTER`
`CSSMERR_CL_UNKNOWN_FORMAT`
`CSSMERR_CL_INVALID_CACHE_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlGetFirstCachedFieldValue, *CSSM_CL_CrlGetNextCachedFieldValue*, *CSSM_CL_CrlGetAllCachedRecordField*, *CSSM_CL_CrlCache*, *CSSM_CL_CrlAbortCache*

Functions for the CLI SPI:

CL_CrlGetFirstCachedFieldValue, *CL_CrlGetNextCachedFieldValue*, *CL_CrlGetAllCachedRecordField*, *CL_CrlCache*, *CL_CrlAbortCache*

CL_IsCertInCrl

NAME

CL_IsCertInCrl: CSSM_CL_IsCertInCrl – Search CRL for a certificate record (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_IsCertInCrl
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Cert,
const CSSM_DATA *Crl,
CSSM_BOOL *CertFound)
SPI:
CSSM_RETURN CSSMCLI CL_IsCertInCrl
(CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *Cert,
const CSSM_DATA *Crl,
CSSM_BOOL *CertFound)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`Cert` (*input*)

A pointer to the CSSM_DATA structure containing the certificate to be located.

`Crl` (*input*)

A pointer to the CSSM_DATA structure containing the CRL to be searched.

`CertFound` (*output*)

A pointer to a CSSM_BOOL indicating success or failure in finding the specified certificate in the CRL. CSSM_TRUE signifies that the certificate was found in the CRL. CSSM_FALSE indicates that the certificate was not found in the CRL.

DESCRIPTION

This function searches the CRL for a record corresponding to the certificate. The result of the search is returned in `CertFound`. The CRL and the records within the CRL must be digitally signed. This function does not verify either signature. The caller should use `CSSM_TP_CrlVerify()` or `CSSM_CL_CrlVerify()` (or their SPI equivalents) before invoking this function. Once the CRL has been verified, the caller can invoke this function repeatedly without repeating the verification process.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CL_INVALID_CERT_POINTER
CSSMERR_CL_INVALID_CRL_POINTER
CSSMERR_CL_UNKNOWN_FORMAT
```

SEE ALSO

Intel CDSA Application Developer's Guide

CL_PassThrough

NAME

CL_PassThrough: CSSM_CL_PassThrough – Extend certificate library functionality (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CL_PassThrough
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
uint32 PassThroughId,
const void *InputParams,
void **OutputParams)
SPI:
CSSM_RETURN CSSMCLI CL_PassThrough
(CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
uint32 PassThroughId,
const void *InputParams,
void **OutputParams)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CLHandle` (*input*)

The handle that describes the add-in Certificate Library module used to perform this function.

`CCHandle` (input/optional)

The handle that describes the context of the cryptographic operation. If the module-specific operation does not perform any cryptographic operations, a cryptographic context is not required.

`PassThroughId` (*input*)

An identifier assigned by the CL module to indicate the exported function to perform.

`InputParams` (input/optional)

A pointer to a module, implementation-specific structure containing parameters to be interpreted in a function-specific manner by the requested CL module.

`OutputParams` (output/optional)

A pointer to a module, implementation-specific structure containing the output data. The service provider allocates the memory for substructures. The application must free the memory for the substructures.

DESCRIPTION

This function allows applications to call certificate library module-specific operations. Such operations might include queries or services that are specific to the domain represented by the CL module.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CL_INVALID_CONTEXT_HANDLE
CSSMERR_CL_INVALID_PASSTHROUGH_ID
CSSMERR_CL_INVALID_DATA
```

SEE ALSO

Intel CDSA Application Developer's Guide

CSP_EventNotify

NAME

CSP_EventNotify – Notify service module of a context event

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMSPI CSP_EventNotify
(CSSM_MODULE_HANDLE CSPHandle,
CSSM_CONTEXT_EVENT Event,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context)
```

The CSP_EventNotify( ) function is used by the CSSM Core to interact with the CSP module. Because this function is exposed to CSSM only as a function pointer, the function name internal to the CSP can be assigned at the discretion of the CSP module developer. However, the parameter list and return value types must match those defined for this function.

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

`Event` (*input*)

One of the following event types listed:

Event	Description
`CSSM_CONTEXT_EVENT_CREATE`	A caller using this module attach handle has created a new cryptographic context using `CSSM_Create***Context`.
`CSSM_CONTEXT_EVENT_DELETE`	A caller using this module attach handle has deleted a cryptographic context using `CSSM_DeleteContext()`.
`CSSM_CONTEXT_EVENT_UPDATE`	A caller using this module attach handle has updated an existing cryptographic context.

`CCHandle` (*input*)

The cryptographic context handle for the context affected by the event.

`Context`

A pointer to the cryptographic context affected by the event. The results of the event are visible in the context.

DESCRIPTION

This function is used to notify the service module of a context event related to a particular attach handle. Valid events include creation, deletion, or modification of a cryptographic context. The service module can examine the new or modified context referenced by `pContext` to determine whether the context is acceptable to the service module.

If the cryptographic context is acceptable (if the service module examines the contents of the context only upon use of the context), then the service module should return `CSSM_OK`. If the cryptographic context is not acceptable, then the service module should return `CSSM_FAIL`.

Upon receiving a return value of `CSSM_OK`, CSSM completes the operation signaled by this event and returns to the calling application. If the return value is `CSSM_FAIL`, CSSM deletes a newly created context or modifications to an existing context, and returns the failed result to the calling application. When deleting a cryptographic context, CSSM always returns success to the calling application.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions:

CSSM_CSP_CreateSignatureContext, *CSSM_CSP_CreateDigestContext*, *CSSM_CSP_CreateSymmetricContext*, *CSSM_CSP_CreateMacContext*, *CSSM_CSP_CreateRandomGenContext*, *CSSM_CSP_CreateAsymmetricContext*, *CSSM_CSP_CreateDeriveKeyContext*, *CSSM_CSP_CreateKeyGenContext*, *CSSM_CSP_CreatePassThroughContext*, *CSSM_DeleteContext*, *CSSM_UpdateContextAttributes*

cssm_CcToHandle

NAME

cssm_CcToHandle – Get the module attach handle (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI cssm_CcToHandle
(CSSM_CC_HANDLE Cc,
CSSM_MODULE_HANDLE_PTR ModuleHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`Cc` (*input*)

A handle identifying a cryptographic context.

`ModuleHandle` (*output*)

A service provider's module attach handle. This value will be set to CSSM_INVALID_HANDLE if the function fails.

DESCRIPTION

This function returns the module attach handle identifying the service module that is managing the specified cryptographic context.

The entry point to this function is provided to a service module in a table of `upcall` functions passed to the service provider during module attach processing.

If the PVC checking for service providers is on, the service provider has to introduce itself before calling this function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

CSSM_ChangeKeyAcl

NAME

CSSM_ChangeKeyAcl – Edit a stored ACL associated with the target key (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_ChangeKeyAcl
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_ACL_EDIT *AclEdit,
const CSSM_KEY *Key)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The module handle that identifies the Cryptographic Service Provider to perform this operation

`AccessCred` (*input*)

A pointer to the set of one or more credentials used to authenticate and validate the caller's authorization to modify the ACL associated with the key. Required credentials can include zero or more certificates, zero or more caller names, and one or more samples. If certificates and/or caller names are provided as input, these must be provided as immediate values in this structure. The samples can be provided as immediate values or can be obtained through a callback function included in the `AccessCred` structure.

`AclEdit` (*input*)

A structure containing information that defines the edit operation. Valid operations include: adding, replacing, and deleting entries in an ACL managed by the service provider. The `AclEdit` can contain information for a new ACL entry and a handle uniquely identifying an existing ACL entry. The information controls the edit operation as follows:

Value of AclEdit.EditMode	Use of AclEdit.NewEntry and AclEdit.OldEntryHandle
CSSM_ACL_EDIT_MODE_ADD	Adds a new ACL entry to the set of ACL entries associated with the specified `Key`. The new ACL entry is created from the ACL entry prototype contained in `NewEntry`. `OldEntryHandle` is ignored for this edit mode.
CSSM_ACL_EDIT_MODE_DELETE	Deletes the ACL entry identified by `OldEntryHandle` and associated with the specified `Key`. `NewEntry` is ignored for this edit mode.

Value of AclEdit.EditMode	Use of AclEdit.NewEntry and AclEdit.OldEntryHandle
CSSM_ACL_EDIT_MODE_REPLACE	Replaces the ACL entry identified by `OldEntryHandle` and associated with the specified `Key`. The existing ACL is replaced based on the ACL entry prototype contained in the `NewEntry`.

When replacing an existing ACL entry, the caller must replace all of the items in an ACL entry. The replacement prototype includes:

Subject type and value

A CSSM_LIST structure containing a typed Subject. The Subject identifies the entity authorized by this ACL entry.

Delegation flag

A CSSM_BOOL value indicating whether the subject can delegate the permissions recorded in the authorization array.

Authorization array

A CSSM_AUTHORIZATIONGROUP structure defining the set of operations for which permission is granted to the Subject.

Validity period

A CSSM_ACL_VALIDITY_PERIOD structure containing two elements, the start time and the stop time for which the ACL entry is valid.

ACL entry tag

A CSSM_STRING containing a user-defined value associated with the ACL entry.

Key (input)

A pointer to the target key whose associated ACL is being modified.

DESCRIPTION

This function edits the stored ACL associated with the target key. The ACL is modified according to the edit mode and information provided in `AclEdit`.

The caller must be authorized to modify the target ACL. Caller authentication and authorization to edit the ACL is determined based on the caller-provided `AccessCred`.

The caller must be authorized to add, delete, or replace the ACL entries associated with the target key. When adding or replacing an ACL entry, the service provider must reject the creation of duplicate ACL entries.

When adding a new ACL entry to an ACL, the caller must provide a complete ACL entry prototype. All ACL entry items, except the ACL entry `Subject` must be provided as an immediate value in `AclEdit->NewEntry`. The ACL entry `Subject` can be provided as an immediate value, from a verifier with a protected data path, from an external authentication or authorization service, or through a callback function specified in `AclEdit->NewEntry->Callback`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_GetKeyAcl*

CSSM_ChangeKeyOwner

NAME

CSSM_ChangeKeyOwner – Change the owner of a key (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_ChangeKeyOwner
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_KEY *Key,
const CSSM_ACL_OWNER_PROTOTYPE *NewOwner)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The module handle that identifies the Cryptographic Service Provider to perform this operation.

`AccessCred` (*input*)

A pointer to the set of one or more credentials used to prove the caller is the current Oowner of the key. Required credentials can include zero or more certificates, zero or more caller names, and one or more samples. If certificates and/or caller names are provided as input, these must be provided as immediate values in this structure. The samples can be provided as immediate values or can be obtained through a callback function included in the `AccessCred` structure.

`Key` (*input*)

A pointer to the target key whose associated Owner is changed.

`NewOwner` (*Input*)

A CSSM_ACL_OWNER_PROTOTYPE defining the new owner of the key.

DESCRIPTION

This function takes a CSSM_ACL_OWNER_PROTOTYPE defining the new owner of the key.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_GetKeyOwner*

CSSM_CSP_ChangeLoginAcl

NAME

CSSM_CSP_ChangeLoginAcl – Edit a stored CSP ACL login session (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_ChangeLoginAcl
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_ACL_EDIT *AclEdit)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The module handle that identifies the Cryptographic Service Provider to perform this operation

`AccessCred` (*input*)

A pointer to the set of one or more credentials used to authenticate and validate the caller's authorization to modify the ACL controlling login sessions with the CSP. Required credentials can include zero or more certificates, zero or more caller names, and one or more samples. Traditionally a caller name has been used to establish the context of a login session. Certificates can be used for the same purpose. If certificates and/or caller names are provided as input, these must be provided as immediate values in this structure. The samples can be provided as immediate values or can be obtained through a callback function included in the `AccessCred` structure.

`AclEdit` (*input*)

A structure containing information that defines the edit operation. Valid operations include adding, replacing, and deleting entries in an ACL managed by the service provider. The `AclEdit` parameter can contain information for a new ACL entry and a handle uniquely identifying an existing ACL entry. The information controls the edit operation as follows:

Value of AclEdit.EditMode	Use of AclEdit.NewEntry and AclEdit.OldEntryHandle
CSSM_ACL_EDIT_MODE_ADD	Adds a new ACL entry to the set of ACL entries controlling login sessions with the CSP. The new ACL entry is created from the ACL entry prototype contained in `NewEntry`. `OldEntryHandle` is ignored for this `EditMode`.

Value of AclEdit.EditMode	Use of AclEdit.NewEntry and AclEdit.OldEntryHandle
CSSM_ACL_EDIT_MODE_DELETE	Deletes the ACL entry identified by `OldEntryHandle` and associated with login sessions with the CSP. `NewEntry` is ignored for this `EditMode`.
CSSM_ACL_EDIT_MODE_REPLACE	Replaces the ACL entry identified by `OldEntryHandle` and controlling login sessions with the CSP. The existing ACL is replaced based on the ACL entry prototype contained in the `NewEntry`.

When replacing an existing ACL entry, the caller must replace all items in an ACL entry. The replacement prototype includes:

- Subject type and value – A CSSM_LIST structure containing a typed subject. The subject identifies the entity authorized by this ACL entry.
- Delegation flag – A CSSM_BOOL value indicating whether the subject can delegate the permissions recorded in the authorization array.
- Authorization array – A CSSM_AUTHORIZATIONGROUP structure defining the set of operations for which permission is granted to the subject.
- Validity period – A CSSM_ACL_VALIDITY_PERIOD structure containing two elements, the start time and the stop time for which the ACL entry is valid.
- ACL entry tag – A CSSM_STRING containing a user-defined value associated with the ACL entry.

DESCRIPTION

This function edits the stored ACL controlling login sessions for a Cryptographic Service Provider (CSP). The ACL is modified according to the edit mode and information provided in `AclEdit`.

The caller must have a login session in process and must be authorized to modify the target ACL. Caller authentication and authorization to edit the ACL is determined based on the caller-provided `AccessCred`.

The caller must be authorized to add, delete, or replace the ACL entries controlling login to the CSP. When adding or replacing an ACL entry, the service provider must reject the creation of duplicate ACL entries.

When adding a new ACL entry to an ACL, the caller must provide a complete ACL entry prototype. All ACL entry items, except the ACL entry Subject, must be provided as an immediate value in `AclEdit.NewEntry`. The ACL entry Subject can be provided as an immediate value, from a verifier with a protected data path, from an external authentication or authorization service, or through a callback function specified in `AclEdit.NewEntry.Callback`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_CSP_GetLoginACLCSSM_CSP_Login*, *CSSM_CSP_Logout*

CSSM_CSP_ChangeLoginOwner

NAME

CSSM_CSP_ChangeLoginOwner – Define a new login owner (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_ChangeLoginOwner
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_ACL_OWNER_PROTOTYPE *NewOwner)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The module handle that identifies the Cryptographic Service Provider to perform this operation.

`AccessCred` (*input*)

A pointer to the set of one or more credentials used to prove the caller is the current login owner. Required credentials can include zero or more certificates, zero or more caller names, and one or more samples. If certificates and/or caller names are provided as input, these must be provided as immediate values in this structure. The samples can be provided as immediate values or can be obtained through a callback function included in the `AccessCred` structure.

`NewOwner` (*Input*)

A CSSM_ACL_OWNER_PROTOTYPE defining the new login owner.

DESCRIPTION

This function takes a CSSM_ACL_OWNER_PROTOTYPE describing the new login owner.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_CSP_GetLoginOwner*

CSSM_CSP_CreateAsymmetricContext

NAME

CSSM_CSP_CreateAsymmetricContext – Create an asymmetric encryption cryptographic context (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_CreateAsymmetricContext
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ALGORITHMS AlgorithmID,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_KEY *Key,
CSSM_PADDING Padding,
CSSM_CC_HANDLE *NewContextHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns an error.

`AlgorithmID` (*input*)

The algorithm identification number for the algorithm used for asymmetric encryption.

`AccessCred` (*input*)

A pointer to the set of one or more credentials required to unlock the private key. The credentials structure can contain an immediate value for the credential, such as a passphrase, or the caller can specify a `callback` function the CSP can use to obtain one or more credentials. Credentials can be required for encryption and decryption operations.

`Key` (*input*)

The key used for asymmetric encryption. The caller passes a pointer to a CSSM_KEY structure containing the key. When the context is used for a sign operation, `AccessCredentials` is required to access the private key used for signing. When the context is used for a verify operation, the public key is used to verify the signature. When the context is used for a wrapkey operation, the public key can be used as the wrapping key. When the context is used for an unwrap operation, `AccessCredentials` is required to access the private key used to perform the unwrapping.

`Padding` (input/optional)

The method for padding. Typically specified for ciphers that pad.

`NewContextHandle` (*output*)

Cryptographic context handle.

DESCRIPTION

This function creates an asymmetric encryption cryptographic context, given a handle of a CSP, an algorithm identification number, a key, and padding. The cryptographic context handle is returned. The cryptographic context handle can be used to call asymmetric encryption functions and cryptographic wrap or unwrap functions.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_DecryptData*, *CSSM_DecryptDataInit*, *CSSM_DecryptDataUpdate*, *CSSM_DecryptDataFinal*, *CSSM_DeleteContext*, *CSSM_EncryptData*, *CSSM_EncryptDataInit*, *CSSM_EncryptDataUpdate*, *CSSM_EncryptDataFinal*, *CSSM_GetContext*, *CSSM_GetContextAttribute*, *CSSM_QuerySize*, *CSSM_SetContext*, *CSSM_UpdateContextAttributes*

CSSM_CSP_CreateDeriveKeyContext

NAME

CSSM_CSP_CreateDeriveKeyContext – Create a cryptographic context to derive a symmetric key (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_CreateDeriveKeyContext
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ALGORITHMS AlgorithmID,
CSSM_KEY_TYPE DeriveKeyType,
uint32 DeriveKeyLengthInBits,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_KEY *BaseKey,
uint32 IterationCount,
const CSSM_DATA *Salt,
const CSSM_CRYPTO_DATA *Seed,
CSSM_CC_HANDLE *NewContextHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns an error.

`AlgorithmID` (*input*)

The algorithm identification number for a derived key algorithm.

`DeriveKeyType` (*input*)

The type of symmetric key to derive.

`DeriveKeyLengthInBits` (*input*)

The logical length of the key in bits to be derived (`LogicalKeySizeInBits`)

`AccessCred` (input/optional)

A pointer to the set of one or more credentials required to access the base key. The credentials structure can contain an immediate value for the credential, such as a passphrase, or the caller can specify a callback function the CSP can use to obtain one or more credentials. If the `BaseKey` is NULL, then this parameter is optional.

`BaseKey` (input/optional)

The base key used to derive the new key. The base key can be a public key, a private key, or a symmetric key

`IterationCount` (input/optional)

The number of iterations to be performed during the derivation process. Used heavily by password-based derivation methods.

`Salt` (input/optional)

A Salt used in deriving the key.

`Seed` (input/optional)

A seed used to generate a random number. The caller can either pass a seed and seed length in bytes or pass a callback function. If `Seed` is NULL, the Cryptographic Service Provider will use its default seed-handling mechanism.

`NewContextHandle` (*output*)

Cryptographic context handle.

DESCRIPTION

This function creates a cryptographic context to derive a symmetric key, given a handle of a CSP, an algorithm, the type of symmetric key to derive, the length of the derived key, and an optional seed or an optional `AccessCredentials` structure from which to derive a new key. The cryptographic context handle is returned. The cryptographic context handle can be used for calling the cryptographic derive key function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_DeriveKey*

CSSM_CSP_CreateDigestContext

NAME

CSSM_CSP_CreateDigestContext – Create a digest cryptographic context (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_CreateDigestContext
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ALGORITHMS AlgorithmID,
CSSM_CC_HANDLE *NewContextHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

`AlgorithmID` (*input*)

The algorithm identification number for message digests.

`NewContextHandle` (*output*)

Cryptographic context handle.

DESCRIPTION

This function creates a digest cryptographic context, given a handle of a CSP and an algorithm identification number. The cryptographic context handle is returned. The cryptographic context handle can be used to call digest cryptographic functions.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_DigestData*, *CSSM_DigestDataInit*, *CSSM_DigestDataUpdate*, *CSSM_DigestDataFinal*, *CSSM_GetContext*, *CSSM_SetContext*, *CSSM_DeleteContext*, *CSSM_GetContextAttribute*, *CSSM_UpdateContextAttributes*

CSSM_CSP_CreateKeyGenContext

NAME

CSSM_CSP_CreateKeyGenContext – Create a key generation cryptographic context (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_CreateKeyGenContext
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ALGORITHMS AlgorithmID,
uint32 KeySizeInBits,
const CSSM_CRYPTO_DATA *Seed,
const CSSM_DATA *Salt,
const CSSM_DATE *StartDate,
const CSSM_DATE *EndDate,
const CSSM_DATA *Params,
CSSM_CC_HANDLE *NewContextHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns an error.

`AlgorithmID` (*input*)

The algorithm identification number of the algorithm used for key generation.

`KeySizeInBits` (*input*)

The logical size of the key (specified in bits). This refers to either the actual key size (for symmetric key generation) or the modulus size (for asymmetric key pair generation).

`Seed` (input/optional)

A seed used to generate the key. The caller can either pass a seed and seed length in bytes or pass a callback function. If NULL is passed, the Cryptographic Service Provider will use its default seed-handling mechanism.

`Salt` (input/optional)

A salt used to generate the key.

`StartDate` (input/optional)

A start date for the validity period of the key or key pair being generated.

`EndDate` (input/optional)

An end date for the validity period of the key or key pair being generated.

`Params` (input/optional)

A data buffer containing parameters required to generate a key pair for a specific algorithm.

`NewContextHandle(`*output*`)`

Cryptographic context handle.

DESCRIPTION

This function creates a key generation cryptographic context, given a handle of a CSP, an algorithm identification number, a passphrase, a modulus size (for public or private keypair generation), a key size (for symmetric key generation), a seed, and a salt. The cryptographic context handle is returned. The cryptographic context handle can be used to call key/ or keypair generation functions.

Additional attributes can be added to the newly created context using the `CSSM_UpdateContextAttributes()` function. Incremental attributes of interest for key generation include a handle-pair identifying a Data Storage Library service module and an open data store for CSPs that manage multiple persistent key stores. If a CSP does not support multiple key stores, the CSP ignores the presence or absence of this attribute.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_GenerateKey*, *CSSM_GenerateKeyPair*, *CSSM_GetContext*, *CSSM_SetContext*, *CSSM_DeleteContext*, *CSSM_GetContextAttribute*, *CSSM_UpdateContextAttributes*

CSSM_CSP_CreateMacContext

NAME

CSSM_CSP_CreateMacContext – Create a message authentication code cryptographic context (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_CreateMacContext
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ALGORITHMS AlgorithmID,
const CSSM_KEY *Key,
CSSM_CC_HANDLE *NewContextHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

`AlgorithmID` (*input*)

The algorithm identification number for the MAC algorithm.

`Key` (*input*)

The key used to generate a message authentication code. Caller passes a pointer to a CSSM_KEY structure containing the key.

`NewContextHandle` (*output*)

Cryptographic context handle.

DESCRIPTION

This function creates a message authentication code cryptographic context, given a handle of a CSP, algorithm identification number, and a key. The cryptographic context handle is returned. The cryptographic context handle can be used to call message authentication code functions.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_GenerateMac*, *CSSM_GenerateMacInit*, *CSSM_GenerateMacUpdate*, *CSSM_GenerateMacFinal*, *CSSM_VerifyMac*, *CSSM_VerifyMacInit*, *CSSM_VerifyMacUpdate*, *CSSM_VerifyMacFinal*, *CSSM_GetContext*, *CSSM_SetContext*, *CSSM_DeleteContext*, *CSSM_GetContextAttribute*, *CSSM_UpdateContextAttributes*

CSSM_CSP_CreatePassThroughContext

NAME

CSSM_CSP_CreatePassThroughContext – Create a custom cryptographic context (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_CreatePassThroughContext
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_KEY *Key,
CSSM_CC_HANDLE *NewContextHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns an error.

`Key` (*input*)

The key to be used for the context. The caller passes a pointer to a CSSM_KEY structure containing the key.

`NewContextHandle` (*output*)

Cryptographic context handle.

DESCRIPTION

This function creates a custom cryptographic context, given a handle of a CSP and a pointer to a custom input data structure. The cryptographic context handle is returned. The cryptographic context handle can be used to call the CSSM pass-through function for the CSP.

NOTES

A CSP can create its own set of custom functions. The context information can be passed through its own data structure. The `CSSM_CSP_PassThrough()` function should be used with the function ID to call the desired custom function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_CSP_PassThroughCSSM_GetContext*, *CSSM_SetContext*, *CSSM_DeleteContext*, *CSSM_GetContextAttribute*, *CSSM_UpdateContextAttributes*

CSSM_CSP_CreateDeriveKeyContext

NAME

CSSM_CSP_CreateDeriveKeyContext – Create a cryptographic context to derive a symmetric key (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_CreateDeriveKeyContext
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ALGORITHMS AlgorithmID,
CSSM_KEY_TYPE DeriveKeyType,
uint32 DeriveKeyLengthInBits,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_KEY *BaseKey,
uint32 IterationCount,
const CSSM_DATA *Salt,
const CSSM_CRYPTO_DATA *Seed,
CSSM_CC_HANDLE *NewContextHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns an error.

`AlgorithmID` (*input*)

The algorithm identification number for a derived key algorithm.

`DeriveKeyType` (*input*)

The type of symmetric key to derive.

`DeriveKeyLengthInBits` (*input*)

The logical length of the key in bits to be derived (`LogicalKeySizeInBits`)

`AccessCred` (input/optional)

A pointer to the set of one or more credentials required to access the base key. The credentials structure can contain an immediate value for the credential, such as a passphrase, or the caller can specify a callback function the CSP can use to obtain one or more credentials. If the `BaseKey` is NULL, then this parameter is optional.

`BaseKey` (input/optional)

The base key used to derive the new key. The base key can be a public key, a private key, or a symmetric key

`IterationCount` (input/optional)

The number of iterations to be performed during the derivation process. Used heavily by password-based derivation methods.

`Salt` (input/optional)

A Salt used in deriving the key.

`Seed` (input/optional)

A seed used to generate a random number. The caller can either pass a seed and seed length in bytes or pass a callback function. If `Seed` is NULL, the Cryptographic Service Provider will use its default seed-handling mechanism.

`NewContextHandle` (*output*)

Cryptographic context handle.

DESCRIPTION

This function creates a cryptographic context to derive a symmetric key, given a handle of a CSP, an algorithm, the type of symmetric key to derive, the length of the derived key, and an optional seed or an optional `AccessCredentials` structure from which to derive a new key. The cryptographic context handle is returned. The cryptographic context handle can be used for calling the cryptographic derive key function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_DeriveKey*

CSSM_CSP_CreateDigestContext

NAME

CSSM_CSP_CreateDigestContext – Create a digest cryptographic context (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_CreateDigestContext
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ALGORITHMS AlgorithmID,
CSSM_CC_HANDLE *NewContextHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

`AlgorithmID` (*input*)

The algorithm identification number for message digests.

`NewContextHandle` (*output*)

Cryptographic context handle.

DESCRIPTION

This function creates a digest cryptographic context, given a handle of a CSP and an algorithm identification number. The cryptographic context handle is returned. The cryptographic context handle can be used to call digest cryptographic functions.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_DigestData*, *CSSM_DigestDataInit*, *CSSM_DigestDataUpdate*, *CSSM_DigestDataFinal*, *CSSM_GetContext*, *CSSM_SetContext*, *CSSM_DeleteContext*, *CSSM_GetContextAttribute*, *CSSM_UpdateContextAttributes*

CSSM_CSP_CreateKeyGenContext

NAME

CSSM_CSP_CreateKeyGenContext – Create a key generation cryptographic context (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_CreateKeyGenContext
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ALGORITHMS AlgorithmID,
uint32 KeySizeInBits,
const CSSM_CRYPTO_DATA *Seed,
const CSSM_DATA *Salt,
const CSSM_DATE *StartDate,
const CSSM_DATE *EndDate,
const CSSM_DATA *Params,
CSSM_CC_HANDLE *NewContextHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns an error.

`AlgorithmID` (*input*)

The algorithm identification number of the algorithm used for key generation.

`KeySizeInBits` (*input*)

The logical size of the key (specified in bits). This refers to either the actual key size (for symmetric key generation) or the modulus size (for asymmetric key pair generation).

`Seed` (input/optional)

A seed used to generate the key. The caller can either pass a seed and seed length in bytes or pass a callback function. If NULL is passed, the Cryptographic Service Provider will use its default seed-handling mechanism.

`Salt` (input/optional)

A salt used to generate the key.

`StartDate` (input/optional)

A start date for the validity period of the key or key pair being generated.

`EndDate` (input/optional)

An end date for the validity period of the key or key pair being generated.

`Params` (input/optional)

A data buffer containing parameters required to generate a key pair for a specific algorithm.

`NewContextHandle(`*output*`)`

Cryptographic context handle.

DESCRIPTION

This function creates a key generation cryptographic context, given a handle of a CSP, an algorithm identification number, a passphrase, a modulus size (for public or private keypair generation), a key size (for symmetric key generation), a seed, and a salt. The cryptographic context handle is returned. The cryptographic context handle can be used to call key/ or keypair generation functions.

Additional attributes can be added to the newly created context using the `CSSM_UpdateContextAttributes()` function. Incremental attributes of interest for key generation include a handle-pair identifying a Data Storage Library service module and an open data store for CSPs that manage multiple persistent key stores. If a CSP does not support multiple key stores, the CSP ignores the presence or absence of this attribute.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_GenerateKey*, *CSSM_GenerateKeyPair*, *CSSM_GetContext*, *CSSM_SetContext*, *CSSM_DeleteContext*, *CSSM_GetContextAttribute*, *CSSM_UpdateContextAttributes*

CSSM_CSP_CreateMacContext

NAME

CSSM_CSP_CreateMacContext – Create a message authentication code cryptographic context (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_CreateMacContext
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ALGORITHMS AlgorithmID,
const CSSM_KEY *Key,
CSSM_CC_HANDLE *NewContextHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

`AlgorithmID` (*input*)

The algorithm identification number for the MAC algorithm.

`Key` (*input*)

The key used to generate a message authentication code. Caller passes a pointer to a CSSM_KEY structure containing the key.

`NewContextHandle` (*output*)

Cryptographic context handle.

DESCRIPTION

This function creates a message authentication code cryptographic context, given a handle of a CSP, algorithm identification number, and a key. The cryptographic context handle is returned. The cryptographic context handle can be used to call message authentication code functions.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_GenerateMac*, *CSSM_GenerateMacInit*, *CSSM_GenerateMacUpdate*, *CSSM_GenerateMacFinal*, *CSSM_VerifyMac*, *CSSM_VerifyMacInit*, *CSSM_VerifyMacUpdate*, *CSSM_VerifyMacFinal*, *CSSM_GetContext*, *CSSM_SetContext*, *CSSM_DeleteContext*, *CSSM_GetContextAttribute*, *CSSM_UpdateContextAttributes*

CSSM_CSP_CreatePassThroughContext

NAME

CSSM_CSP_CreatePassThroughContext – Create a custom cryptographic context (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_CreatePassThroughContext
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_KEY *Key,
CSSM_CC_HANDLE *NewContextHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns an error.

`Key` (*input*)

The key to be used for the context. The caller passes a pointer to a CSSM_KEY structure containing the key.

`NewContextHandle` (*output*)

Cryptographic context handle.

DESCRIPTION

This function creates a custom cryptographic context, given a handle of a CSP and a pointer to a custom input data structure. The cryptographic context handle is returned. The cryptographic context handle can be used to call the CSSM pass-through function for the CSP.

NOTES

A CSP can create its own set of custom functions. The context information can be passed through its own data structure. The `CSSM_CSP_PassThrough()` function should be used with the function ID to call the desired custom function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_CSP_PassThroughCSSM_GetContext*, *CSSM_SetContext*, *CSSM_DeleteContext*, *CSSM_GetContextAttribute*, *CSSM_UpdateContextAttributes*

CSSM_CSP_CreateRandomGenContext

NAME

CSSM_CSP_CreateRandomGenContext – Create a random number generation cryptographic context (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_CreateRandomGenContext
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ALGORITHMS AlgorithmID,
const CSSM_CRYPTO_DATA *Seed,
uint32 Length,
CSSM_CC_HANDLE *NewContextHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns AN error.

`AlgorithmID` (*input*)

The algorithm identification number for random number generation.

`Seed` (input/optional)

A seed used to generate THE random number. The caller can either pass a seed and seed length in bytes or pass a callback function. If NULL is passed, the Cryptographic Service Provider will use its default seed-handling mechanism.

`Length` (*input*)

The length of the random number to be generated.

`NewContextHandle` (*output*)

Cryptographic context handle.

DESCRIPTION

This function creates a random number generation cryptographic context, given a handle of a CSP, an algorithm identification number, a seed, and the length of the random number in bytes. The cryptographic context handle is returned and can be used for the random number generation function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_GenerateRandom*, *CSSM_GetContext*, *CSSM_SetContext*, *CSSM_DeleteContext*, *CSSM_GetContextAttribute*, *CSSM_UpdateContextAttributes*

CSSM_CSP_CreateSignatureContext

NAME

CSSM_CSP_CreateSignatureContext – Create a signature cryptographic context (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_CreateSignatureContext
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ALGORITHMS AlgorithmID,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_KEY *Key,
CSSM_CC_HANDLE *NewContextHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

`AlgorithmID` (*input*)

The algorithm identification number for a signature/verification algorithm.

`AccessCred` (input/optional)

A pointer to the set of one or more credentials required to unlock the private key. The credentials structure can contain an immediate value for the credential, such as a passphrase, or the caller can specify a callback function the CSP can use to obtain one or more credentials. Credentials are required for signature operations, not for verify operations.

`Key` (*input*)

The key used to sign and verify. The caller passes a pointer to a CSSM_KEY structure containing the key and the key length.

`NewContextHandle` (*output*)

Cryptographic context handle.

DESCRIPTION

This function creates a signature cryptographic context for sign and verify, given a handle of a CSP, an algorithm identification number, a key, and an `AccessCredentials` structure. The `AccessCredentials` structure will be used to unlock the private key when this context is used to perform a signing operation. The cryptographic context handle is returned. The cryptographic context handle can be used to call sign and verify cryptographic functions.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_SignData*, *CSSM_SignDataInit*, *CSSM_SignDataUpdate*, *CSSM_SignDataFinal*, *CSSM_VerifyData*, *CSSM_VerifyDataInit*, *CSSM_VerifyDataUpdate*, *CSSM_VerifyDataFinal*, *CSSM_GetContext*, *CSSM_SetContext*, *CSSM_DeleteContext*, *CSSM_GetContextAttribute*, *CSSM_UpdateContextAttributes*

CSSM_CSP_CreateSymmetricContext

NAME

CSSM_CSP_CreateSymmetricContext – Create a symmetric encryption cryptographic context (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_CreateSymmetricContext
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ALGORITHMS AlgorithmID,
CSSM_ENCRYPT_MODE Mode,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_KEY *Key,
const CSSM_DATA *InitVector,
CSSM_PADDING Padding,
void *Reserved,
CSSM_CC_HANDLE *NewContextHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

`AlgorithmID` (*input*)

The algorithm identification number for symmetric encryption.

`Mode` (*input*)

The mode of the specified algorithm ID.

`AccessCred` (input/optional)

A pointer to the set of one or more credentials required to unlock the private key. The credentials structure can contain an immediate value for the credential, such as a passphrase, or the caller can specify a `callback` function the CSP can use to obtain one or more credentials. Credentials may be required for encryption, decryption, and wrapping operations.

`Key` (*input*)

The key used for symmetric encryption. The caller passes a pointer to a CSSM_KEY structure containing the key.

`InitVector` (input/optional)

The initial vector for symmetric encryption. This is typically specified for block ciphers.

`Padding` (input/optional)

The method for padding. This is typically specified for ciphers that pad.

`Reserved (input)`

Reserved for future use.

`NewContextHandle (output)`

Cryptographic context handle.

DESCRIPTION

This function creates a symmetric encryption cryptographic context, given a handle of a CSP, an algorithm identification number, a key, an initial vector, padding, and the number of encryption rounds. Algorithm-specific attributes must be added to the context after the initial creation using the `CSSM_UpdateContextAttributes()` function. The cryptographic context handle is returned. The cryptographic context handle can be used to call symmetric encryption functions and the cryptographic wrap or unwrap functions.

Additional attributes can be added to the newly created context using the `CSSM_UpdateContextAttributes()` function . Incremental attributes of interest when using this context to unwrap a key include a handle-pair identifying a Data Storage Library service module and an open data store for CSPs that manage multiple, persistent key stores. If a CSP does not support multiple key stores, the CSP ignores the presence or absence of this attribute.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_DecryptData*, *CSSM_DecryptDataInit*, *CSSM_DecryptDataUpdate*, *CSSM_DecryptDataFinal*, *CSSM_DeleteContext*, *CSSM_EncryptData*, *CSSM_EncryptDataInit*, *CSSM_EncryptDataUpdate*, *CSSM_EncryptDataFinal*, *CSSM_GetContext*, *CSSM_GetContextAttribute*, *CSSM_QuerySize*, *CSSM_SetContext*, *CSSM_UpdateContextAttributes*

CSSM_CSP_GetLoginAcl

NAME

CSSM_CSP_GetLoginAcl – Get description of CSP ACL entries (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_GetLoginAcl
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_STRING *SelectionTag,
uint32 *NumberOfAclInfos,
CSSM_ACL_ENTRY_INFO_PTR *AclInfos)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The module handle that identifies the Cryptographic Service Provider to perform this operation.

`SelectionTag` (input/optional)

A CSSM_STRING value matching the user-defined tag value associated with one or more ACL entries controlling login sessions. To retrieve a description of all ACL entries controlling login sessions, this parameter must be NULL.

`NumberOfAclInfos` (*output*)

The number of entries in the `AclInfos` array. If no ACL entry descriptions are returned, this value is zero.

`AclInfos` (*output*)

An array of CSSM_ACL_ENTRY_INFO structures. The unique handle contained in this structure can be used during the current attach session and the current login session to reference specific ACL entries for editing. The structure is allocated by the service provider and must be released by the caller when the structure is no longer needed. If no ACL entry descriptions are returned, this value is NULL.

DESCRIPTION

This function returns a description of zero or more ACL entries managed by the CSP and used to control login sessions with the CSP. The optional input `SelectionTag` parameter restricts the returned descriptions to those ACL entries with a matching `EntryTag` value. If a `SelectionTag` value is specified and no matches are found, zero descriptions are returned. If no `SelectionTag` is specified, a description of all ACL entries used to control login sessions are returned by this function.

Each `AclInfo` structure contains:

- Public contents of an ACL entry

- ACL `EntryHandle`, which is a unique value defined and managed by the service provider

The public ACL entry information returned by this function includes:

- Subject type — A CSSM_LIST structure containing one element identifying the type of subject stored in the ACL entry.
- Delegation flag — A CSSM_BOOL value indicating whether the subject can delegate the permissions recorded in the authorization array.
- Authorization array — A CSSM_AUTHORIZATIONGROUP structure defining the set of operations for which permission is granted to the subject.
- Validity period — A CSSM_ACL_VALIDITY_PERIOD structure containing two elements, the start time and the stop time for which the ACL entry is valid.
- ACL entry tag — A CSSM_STRING containing a user-defined value associated with the ACL entry.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_CSP_Login*, *CSSM_CSP_LoginAclCSSM_CSP_Logout*

CSSM_CSP_GetLoginOwner

NAME

CSSM_CSP_GetLoginOwner – Get login owner data (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_GetLoginOwner
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ACL_OWNER_PROTOTYPE_PTR Owner)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The module handle that identifies the Cryptographic Service Provider to perform this operation.

`Owner` (*output*)

A CSSM_ACL_OWNER_PROTOTYPE describing the login owner.

DESCRIPTION

This function returns a CSSM_ACL_OWNER_PROTOTYPE describing the current login owner of the CSP.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_CSP_ChangeLoginOwner*

CSSM_CSP_Login

NAME

CSSM_CSP_Login – Log user in to the CSP (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_Login
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_DATA *LoginName,
const void *Reserved)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

Handle of the CSP to log in to.

`AccessCred` (*input*)

A pointer to the set of one or more credentials required to log in to the token or Cryptographic Service Provider. The credentials structure can contain an immediate value for the credential, such as a passphrase or PIN, or the caller can specify a callback function the CSP can use to obtain one or more credentials.

`LoginName` (input/optional)

A name or ID of the caller. The value is used with the provided `AccessCred` to authenticate and authorize the caller for login with the CSP. The CSP can require that a name value be provided. If a name value is not provided, the CSP can assume a default name under which to perform the authentication and authorization check, or the login request can fail.

`Reserved` (*input*)

This field is reserved for future use. The value NULL should always be given. (May be used for multiple user support in the future.)

DESCRIPTION

Logs the user in to the CSP, allowing for multiple login types.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSP_INVALID_LOGIN_NAME
CSSMERR_CSP_ALREADY_LOGGED_IN
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_CSP_GetLoginAcl*, *CSSM_CSP_ChangeLoginAcl*, *CSSM_CSP_Logout*

CSSM_CSP_Logout

NAME

CSSM_CSP_Logout – Terminate the login session (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_CSP_Logout
(CSSM_CSP_HANDLE CSPHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

Handle for the target CSP.

DESCRIPTION

Terminates the login session associated with the specified CSP handle.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_CSP_Login*, *CSSM_CSP_GetLoginAcl*, *CSSM_CSP_ChangeLoginAcl*

CSSM_DeleteContext

NAME

CSSM_DeleteContext – Free the context structure (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_DeleteContext
(CSSM_CC_HANDLE CCHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CCHandle` (*input*)

The handle that describes a context to be deleted.

DESCRIPTION

This function frees the context structure allocated by any of the `CSSM_Create`*xxxxx* context functions.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSSM_INVALID_CONTEXT_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_CSP_CreateAsymmetricContext*, *CSSM_CSP_CreateKeyGenContext*, *CSSM_CSP_CreateDigestContext*, *CSSM_CSP_CreateSignatureContext*, *CSSM_CSP_CreateSymmetricContext*, and others.

CSSM_DeleteContextAttributes

NAME

CSSM_DeleteContextAttributes – Delete internal data (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_DeleteContextAttributes
(CSSM_CC_HANDLE CCHandle,
uint32 NumberOfAttributes,
const CSSM_CONTEXT_ATTRIBUTE *ContextAttributes)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CCHandle` (*input*)

The handle that describes a context that is to be deleted.

`NumberOfAttributes` (*input*)

The number of attributes to be deleted as specified in the array of context attributes.

`ContextAttributes` (*input*)

The attributes to be deleted from the context. Only the attribute type is required. Any attribute values in the CSSM_CONTEXT_ATTRIBUTE structures are ignored.

DESCRIPTION

This function deletes internal data associated with the given attribute type of the context handle.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSSM_INVALID_CONTEXT_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_GetContextAttributes*, *CSSM_UpdateContextAttributes*

cssm_DeregisterManagerServices

NAME

cssm_DeregisterManagerServices – Deregister manager services

SYNOPSIS

include <cssm.h>

```
void CSSMAPI cssm_DeregisterManagerServices
(const CSSM_GUID *Guid);
```

PARAMETERS

GUID (*input*)

A pointer to the CSSM_GUID structure containing the global unique identifier for this module.

DESCRIPTION

This function is used by an elective module manager to deregister its function table with CSSM core services prior to termination. This function is invoked by an elective module manager only when exiting due to an error condition detected by the EMM. This allows CSSM to clean up any state information associated with the exiting EMM.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

CSSM_FreeContext

NAME

CSSM_FreeContext – Free memory associated with the context structure (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_FreeContext
(CSSM_CONTEXT_PTR Context)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`Context` (*input*)

The pointer to the memory that describes the context structure.

DESCRIPTION

This function frees the memory associated with the context structure.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_GetContext*

CSSM_GetAPIMemoryFunctions

NAME

CSSM_GetAPIMemoryFunctions – Retrieve the memory function table associated with the security service module

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_GetAPIMemoryFunctions
(CSSM_MODULE_HANDLE AddInHandle,
CSSM_API_MEMORY_FUNCS_PTR AppMemoryFuncs)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`AddInHandle` (*input*)

The handle to the security service module that is associated with the requested memory function table.

`AppMemoryFuncs` (*output*)

The pointer to an empty memory functions table. Upon function return, the table is filled with the memory function pointers associated with the specified attach handle. Caller has to allocate the buffer.

DESCRIPTION

This function retrieves the memory function table associated with the security service module identified by the input handle.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Intel CDSA Application Developer's Guide

cssm_GetAppMemoryFunctions

NAME

cssm_GetAppMemoryFunctions – Get service functions (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI cssm_GetAppMemoryFunctions
(CSSM_MODULE_HANDLE hAddIn,
CSSM_UPCALLS_PTR UpcallTable)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`hAddIn` (*input*)

The handle identifying the attach-session whose memory management function table is returned by this function.

`UpcallTable` (*output*)

The table containing sets of service functions among them a set of four memory management functions provided by the application that initiated the attach-session identified by `hAddIn`.

DESCRIPTION

This function gets a function table containing sets of service functions. Among these service functions are four application-provided memory management functions. The elective module manager can use these functions to manage memory on behalf of the application. The returned function table is specific to the attach-session identified by the module handle.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

cssm_GetAttachFunctions

NAME

cssm_GetAttachFunctions – Get SPI function table (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI cssm_GetAttachFunctions
(CSSM_MODULE_HANDLE hAddIn,
CSSM_SERVICE_MASK AddinType,
void **SPFunctions,
CSSM_GUID_PTR Guid)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`hAddIn` (*input*)

The handle identifying the attach-session whose function table is to be returned by this function.

`AddinType` (*input*)

A CSSM_SERVICE_MASK value identifying the type of service module whose function table is to be returned by this function.

`SPFunctions` (*output*)

A pointer to the service module function table, which CSSM acquired from the service module during module-attach processing. The module manager should use this table to forward application invocation of the elective APIs to their corresponding SPIs. The memory pointed to by the function pointers should not be freed by the EMM.

`Guid` (*output*)

A CSSM_GUID value identifying the service module whose function table is to be returned by this function.

DESCRIPTION

This function returns an SPI function table for the service module identified by the module handle. The module must be of the type specified by the service mask. The `SPFunctions` parameter contains the returned function table. The elective module manager must use this function table to forward an application's call to the elective APIs to their corresponding SPIs represented in the function table. The returned `Guid` identifies the service module. It can be used to locate credentials and other information about the service module.

This function sets a lock on the SP functions table. The CSSM service function `cssm_ReleaseAttachFunctions()` must be used to release the lock.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

CSSM_GetContext

NAME

CSSM_GetContext – Get context information (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_GetContext
(CSSM_CC_HANDLE CCHandle,
CSSM_CONTEXT_PTR *Context)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CCHandle` (*input*)

The handle to the context information.

`Context` (*output*)

The pointer to the CSSM_CONTEXT_PTR structure that describes the context associated with the `CCHandle` handle. The pointer will be set to NULL if the function fails. Use `CSSM_FreeContext()` to free the memory allocated by the CSSM.

DESCRIPTION

This function retrieves the context information when provided with a context handle.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSSM_INVALID_CONTEXT_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_FreeContext*, *CSSM_SetContext*

CSSM_GetContextAttribute

NAME

CSSM_GetContextAttribute – Get context attribute (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_GetContextAttribute
(const CSSM_CONTEXT *Context,
uint32 AttributeType,
CSSM_CONTEXT_ATTRIBUTE_PTR *ContextAttribute)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`Context` (*input*)

A pointer to the context.

`AttributeType` (*input*)

The attribute type of the desired attribute value.

`ContextAttribute` (*output*)

The pointer to the CSSM_CONTEXT_ATTRIBUTE that describes the context attributes associated with the `CCHandle` handle and the attribute type. The pointer will be set to NULL if the function fails. Call `CSSM_DeleteContextAttributes()` to free memory allocated by the CSSM.

DESCRIPTION

This function returns the value of a context attribute. Context references the cryptographic context to be searched for the attribute specified by `AttributeType`. If the specified attribute is not present, then a NULL pointer is returned.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSSM_ATTRIBUTE_NOT_IN_CONTEXT`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_DeleteContextAttributes*, *CSSM_GetContext*

CSSM_GetKeyAcl

NAME

CSSM_GetKeyAcl – Get ACL entries by key (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_GetKeyAcl
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_KEY *Key,
const CSSM_STRING *SelectionTag,
uint32 *NumberOfAclInfos,
CSSM_ACL_ENTRY_INFO_PTR *AclInfos)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The module handle that identifies the Cryptographic Service Provider to perform this operation.

`Key` (*input*)

A pointer to the target key whose associated ACL entries are scanned and returned.

`SelectionTag` (input/optional)

A CSSM_STRING value matching the user-defined tag value associated with one or more ACL entries for the target Key. To retrieve a description of all ACL entries for the target Key, this parameter must be NULL.

`NumberOfAclInfos` (*output*)

The number of entries in the `AclInfos` array. If no ACL entry descriptions are returned, this value is zero.

`AclInfos` (*output*)

An array of CSSM_ACL_ENTRY_INFO structures. The unique handle contained in this structure can be used during the current attach session to reference specific ACL entries for editing. The structure is allocated by the service provider and must be released by the caller when the structure is no longer needed. If no ACL entry descriptions are returned, this value is NULL.

DESCRIPTION

This function returns a description of zero or more ACL entries managed by the CSP and associated with the target key. The optional input `SelectionTag` restricts the returned descriptions to those ACL entries with a matching `EntryTag` value. If a `SelectionTag` value is specified and no matches are found, zero descriptions are returned. If no `SelectionTag` is specified, a description of all ACL entries associated with the key is returned by this function.

Each `AclInfo` structure contains:

- Public contents of an ACL entry
- ACL `EntryHandle`, which is a unique value defined and managed by the service provider

The public ACL entry information returned by this function includes:

Subject type and value

A CSSM_LIST structure containing one element identifying the type of subject stored in the ACL entry.

Delegation flag

A CSSM_BOOL value indicating whether the subject can delegate the permissions recorded in the authorization array.

Authorization array

A CSSM_AUTHORIZATIONGROUP structure defining the set of operations for which permission is granted to the subject.

Validity period

A CSSM_ACL_VALIDITY_PERIOD structure containing two elements, the start time and the stop time for which the ACL entry is valid.

ACL entry tag

A CSSM_STRING containing a user-defined value associated with the ACL entry.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_ChangeKeyAcl*

CSSM_GetKeyOwner

NAME

CSSM_GetKeyOwner – Get data describing key owner (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_GetKeyOwner
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_KEY *Key,
CSSM_ACL_OWNER_PROTOTYPE_PTR Owner)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The module handle that identifies the Cryptographic service provider to perform this operation.

`Key` (*input*)

A pointer to the target key whose associated Owner is returned.

`Owner` (*output*)

A CSSM_ACL_OWNER_PROTOTYPE describing the current Owner of the Key.

DESCRIPTION

This function returns a CSSM_ACL_OWNER_PROTOTYPE describing the current Owner of the Key.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_ChangeKeyOwner*

CSSM_GetModuleGUIDFromHandle

NAME

CSSM_GetModuleGUIDFromHandle – Get GUID of the attached module (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_GetModuleGUIDFromHandle
(CSSM_MODULE_HANDLE ModuleHandle,
CSSM_GUID_PTR ModuleGUID)
```

PARAMETERS

`ModuleHandle` (*input*)

The handle of the module for which the GUID should be returned.

`ModuleGUID` (*output*)

The GUID of the module associated with `ModuleHandle.n`.

DESCRIPTION

This function returns the GUID of the attached module identified by the specified handle.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_GetSubserviceUIDFromHandle*

cssm_GetModuleInfo

NAME

cssm_GetModuleInfo – Get the module handle state information

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI cssm_GetModuleInfo
(CSSM_MODULE_HANDLE Module,
CSSM_GUID_PTR Guid,
CSSM_VERSION_PTR Version,
uint32 *SubServiceId,
CSSM_SERVICE_TYPE *SubServiceType,
CSSM_ATTACH_FLAGS *AttachFlags,
CSSM_KEY_HIERARCHY *KeyHierarchy,
CSSM_API_MEMORY_FUNCS_PTR AttachedMemFuncs,
CSSM_FUNC_NAME_ADDR_PTR FunctionTable,
uint32 NumFunctionTable);
```

PARAMETERS

`Module` (*input*)

The handle to a service provider module.

`GUID` (*input*)

A pointer to the CSSM_GUID structure containing the global unique identifier for this module.

`Version` (*output*)

The version number set on ModuleAttach.

`SubServiceId` (*output*)

The slot number of the reader to which the module is attached.

`SubServiceType` (*output*)

A CSSM_SERVICE_TYPE value identifying the class of security service.

`AttachFlags` (*output*)

This parameter provides the caller with session specific information associated with the module handle.

`KeyHierarchy` (*output*)

The key hierarchy supplied when the module was attached.

`AttachedMemFuncs` (*output*)

The memory functions supplied when the module was attached.

`FunctionTable` (input/output optional)

A table of function-name and API function-pointer pairs. The caller provides the name of the functions as input. The corresponding API function pointers are returned on output.

The function table allows dynamic linking of CDSA interfaces, including interfaces to Elective Module Managers, which are transparently loaded by CSSM during the `CSSM_ModuleAttach()` function. The caller of this function should allocate the memory for the number of slots required.

`NumFunctionTable(`*input*`)`

The number of entries in the FunctionTable parameter. If no FunctionTable is provided, this value must be zero.

DESCRIPTION

This function returns the state information associated with the module handle. The information returned by this function is that set by the call to the `CSSM_ModuleAttach()` function. The entry point to this function is provided to a service module in a table of upcall functions passed to the service provider during module attach processing.

If the PVC checking for service providers is on, the service provider has to introduce itself before calling this function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

CSSM_GetPrivilege

NAME

CSSM_GetPrivilege – Get CSSM privilege value (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_GetPrivilege
(CSSM_PRIVILEGE *Privilege;
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`Privilege` (*output*)

The CSSM_PRIVILEGE value currently set.

DESCRIPTION

The `CSSM_GetPrivilege()` function returns the CSSM_PRIVILEGE value currently established in the framework.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

CSSM_GetSubserviceUIDFromHandle

NAME

CSSM_GetSubserviceUIDFromHandle – Complete a subservice unique identifier structure (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_GetSubserviceUIDFromHandle
(CSSM_MODULE_HANDLE ModuleHandle,
CSSM_SUBSERVICE_UID_PTR SubserviceUID)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`ModuleHandle` (*input*)

Handle of the module subservice for which the subservice unique identifier should be returned.

`SubserviceUID` (*output*)

Subservice UID value associated with `ModuleHandle`. The caller has to allocate the buffer.

DESCRIPTION

This function completes a structure containing the persistent unique identifier of the attached module subservice, as identified by the input handle.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_GetModuleGUIDFromHandle*

CSSM_Init

NAME

CSSM_Init – Initialize CSSM (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_Init(
const CSSM_VERSION *Version,
CSSM_PRIVILEGE_SCOPE Scope,
const CSSM_GUID * CallerGuid,
CSSM_KEY_HIERARCHY KeyHierarchy,
CSSM_PVC_MODE *PvcPolicy,
const void *Reserved)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`Version` (*input*)

The major and minor version number of the CSSM release the application is compatible with.

`Scope` (*input*)

The scope of the global privilege value. The scope may either process scope wide (CSSM_PRIVILEGE_SCOPE_PROCESS) or thread wide (CSSM_PRIVILEGE_SCOPE_THREAD). This parameter is ignored after the first call to `CSSM_Init()`.

`CallerGuid` (*input*)

The GUID associated with the caller. This GUID is used to locate the caller's credentials when evaluating the request for privileges.

`KeyHierarchy` (*input*)

The CSSM_KEY_HIERARCHY option directing CSSM what embedded key to use when verifying integrity of the named module.

`PvcPolicy` (input/output)

Configures the way in which pointer validation checks will be performed. If not the first call to `CSSM_Init()`, the previously configured policy is returned in the `PvcPolicy` bitmask and the `CSSM_Init()` call continues processing. If successfully completed, the error code CSSMERR_CSSM_PVC_ALREADY_CONFIGURED is returned.

Value	Description
0	PVC validation is not performed
1	PVC validation is performed on application modules

Value	Description
2	PVC validation is performed on service provider modules
3	Both types of PVC validations are performed

`Reserved` (*input*)

A reserved input.

DESCRIPTION

This function initializes CSSM and verifies that the version of CSSM expected by the application is compatible with the version of CSSM on the system. This function should be called at least once by the application. It is an error to call any function of the CSSM API other than `CSSM_Init()` before a call to `CSSM_Init()` has returned successfully (that is, with CSSM_OK).

Implementations of CSSM might have platform specific characteristics associated with the implementation of `CSSM_SetPrivilege()` API. The privilege value might have thread specific scope or process specific scope. The application can specify the anticipated scope at `CSSM_Init()`. If the anticipated scope is not appropriate for the implementation, an error is returned. The scope can be configured only once. Subsequent attempts to configure scope are ignored.

CSSM integrity model includes the ability to make and check assertions about trusted dynamically loaded libraries. Checking assertions happens while the program executes. It is known as Pointer Validation Checking (PVC). Pointer validation checking can be applied every time execution flow crosses the CSSM API or SPI interfaces.

Performing pointer validation checks has two purposes:

- It allows exportation of CSSM.
- It aids in detering unanticipated run-time modification of the program.

The CSSM can be configured to bypass pointer validation under some circumstances. Pointer validation cannot be bypassed when privileged operations are being performed.

The prerequisites for performing PVC on another module, be it service provider, CSSM, or other library, are:

- The module must have been signed and have an accompanying signed manifest.
- The module must be loaded into process address space.
- An entry-point into the module must be available.

Typically, the entry points are discovered when a module's functions are called by another module. The CSSM performs pointer validation checks based on the configured checking policy. Checking policies are established by the manufacturers of CSSM and other libraries. The checking policy to be applied during execution is configured using the `CSSM_Init()` call. The policy can be configured once during the life of the process and occurs the first time `CSSM_Init()` is called.

PVC POLICY CONFIGURATION OPTIONS

Pointer validation checking can be applied at the CSSM API interface, the CSSM SPI interface, or both. The CSSM vendor can configure a default policy through instructions contained in the CSSM signed manifest. Manifest attributes pertaining to pointer validation checking are defined as follows:

Module	Tag	Value	Description
CSSM	CDSA_PVC_API	unspecified	CSSM will perform PVC checks at the API boundary.
CSSM	CDSA_PVC_API	OFF	CSSM will not perform PVC checks at the API boundary.
CSSM	CDSA_PVC_SPI	unspecified	CSSM will perform PVC checks at the SPI boundary.
CSSM	CDSA_PVC_SPI	OFF	CSSM will not perform PVC checks at the SPI boundary.
App	CDSA_PVC_API	EXEMPT	The calling module is allowed to override the CSSM policy for the API boundary.
App	CDSA_PVC_API	unspecified	The calling module cannot weaken the CSSM API policy.
App	CDSA_PVC_SPI	EXEMPT	The calling module is allowed to override the CSSM policy for the SPI boundary.
App	CDSA_PVC_SPI	unspecified	The calling module cannot weaken the CSSM SPI policy.

The `PvcPolicy` parameter to `CSSM_Init()` configures the run-time policy for the process. The `PvcPolicy` parameter is a bitmask allowing both API and SPI policies to be specified simultaneously. Unspecified policies default to the most conservative operational mode. CSSM performs pointer validation checks unless explicitly disabled. Application modules cannot override CSSM policy unless exemptions are explicitly granted. The following table shows the what policies can be configured for various manifest attribute values:

CSSM Manifest	Calling Module Manifest	Acceptable PvcPolicy Values
CDSA_PVC_API=<n/a>	CDSA_PVC_API=EXEMPT	API checks: off (0) or on (1)
CDSA_PVC_API=OFF	CDSA_PVC_API=EXEMPT	API checks: off (0) or on (1)
CDSA_PVC_API=<n/a>	CDSA_PVC_API=<n/a>	API checks: on (1)
CDSA_PVC_API=OFF	CDSA_PVC_API=<n/a>	API checks: off (0) or on (1)

The following table shows the `PvcPolicy` configuations available for the SPI:

SSM Manifest	Calling Module Manifest	Acceptable PvcPolicy Values
CDSA_PVC_SPI=<n/a>	CDSA_PVC_SPI=EXEMPT	SPI checks: off (0) or on (2)
CDSA_PVC_SPI=OFF	CDSA_PVC_SPI=EXEMPT	SPI checks: off (0) or on (2)
CDSA_PVC_SPI=<n/a>	CDSA_PVC_SPI=<n/a>	SPI checks: on (2)
CDSA_PVC_SPI=OFF	CDSA_PVC_SPI=<n/a>	SPI checks: off (0) or on (2)

If an application module does not have a manifest and CSSM requires the application module be subject to pointer validation checks, then pointer validation checks fail and CSSM will not operate with the anonymous module. All service provider modules are expected to have signed manifests.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSSM_SCOPE_NOT_SUPPORTED`
`CSSMERR_CSSM_PVC_ALREADY_CONFIGURED`
`CSSMERR_CSSM_INVALID_PVC`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

CSSM_Introduce

NAME

CSSM_Introduce – Identify an executable module (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_Introduce
(const CSSM_GUID *ModuleID,
CSSM_KEY_HIERARCHY KeyHierarchy)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`ModuleID` (*input*)

The CSSM_GUID of the calling library or other library that might call CDSA interfaces. The GUID is used to locate the signed manifest credentials of the named module to calculate module integrity information.

`KeyHierarchy` (*input*)

The CSSM_KEY_HIERARCHY option directing CSSM what embedded key to use when verifying integrity of the named module.

DESCRIPTION

The `CSSM_Introduce()` function identifies a dynamically loadable executable module (for example, DLL) to the CSSM framework. CSSM uses the `ModuleID` information to locate the signed manifest and library on the host platform. The Module Directory Service (MDS) should be used to obtain the information. CSSM performs an integrity cross-check on the module identified by `ModuleID` and caches the result in an internal structure. The integrity cross-check uses the `KeyHierarchy` information to determine which classes of embedded public keys must serve as anchors when doing certificate path validation. If the export key hierarchy is specified, the set of export privileges contained in the manifest are retrieved from the manifest and saved with the integrity state information in the cache. Privileges granted to a module are accepted only if the manifest sections containing the privilege set have been signed by a principal in the export key hierarchy class and that hash of the module binary is part of the hash of the privilege attributes.

The `CSSM_Introduce()` can be called at any time after `CSSM_Init()`, by any module, on behalf of any module.

Once a module is introduced into CSSM the load location of the module must not change. If the load location changes then the module must be reintroduced. Once introduced, the module load location, integrity, and privilege information is held until `CSSM_Terminate()` is called or the process terminates. Initialization of internal data structures maintaining the table of introductions is performed when `CSSM_Init()` is called.

If `CSSM_Introduce()` is called on behalf of another module, then the caller needs to make sure that the other module is loaded into the process address space. If the library is already loaded into process address space, but a reference to the library cannot be obtained, a different error is returned (CSSMERR_CSSM_LIB_REF_NOT_FOUND).

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSSM_INVALID_KEY_HIERARCHY`
`CSSMERR_CSSM_LIB_REF_NOT_FOUND`

SEE ALSO

Intel CDSA Application Developer's Guide

cssm_IsFuncCallValid

NAME

cssm_IsFuncCallValid – Check secure linkage (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI cssm_IsFuncCallValid
(CSSM_MODULE_HANDLE hAddin,
CSSM_PROC_ADDR SrcAddress, /* application */,
CSSM_PROC_ADDR DestAddress,
CSSM_PRIVILEGE InPriv,
CSSM_PRIVILEGE *OutPriv,
CSSM_BITMASK Hints,
CSSM_BOOL * IsOK)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`hAddIn` (*input*)

The handle identifying the attach-session whose caller and callee scope is being tested by this function.

`SrcAddress` (input/optional)

An address to be tested for containment within the application that requested and created the attach-session identified by the module handle.

`DestAddress` (input/optional)

An address within a service module. The destination address must be valid for the service provider associated with the attach-session identified by the module handle.

`InPriv` (*input*)

The privilege value to be checked. Privilege checks apply to both `SrcAddress` and `DestAddress`.

`OutPriv` (*output*)

If non-NULL, the global privilege will be checked and returned in `OutPriv`.

`Hints` (*input*)

A flag providing search hints.

`IsOK` (*output*)

CSSM_TRUE if success, CSSM_FALSE if fail.

DESCRIPTION

This function checks secure linkage between an application and a service module. Based on address scope of the application and the service module associated with the attach handle, CSSM determines whether the `SrcAddress` is within an associated application and `DestAddress` is within the associated service module. The scope of the application and the service module is determined by their respective signed manifest credentials, which attest to the integrity of each entity.

This function uses the input privilege value `InPriv` to compare against the privilege range associated with the ranges for `SrcAddress` and `DestAddres`. The privilege check is performed when the `InPriv` privilege value is non-NULL. If the EMM wants the global privilege value to be checked, `InPriv` is zero and `OutPriv` is non-NULL. CSSM will return the privilege value in `OutPriv`. If integrity only checks are to be performed, `InPriv` is zero and `OutPriv` is NULL.

Another parameter called `Hints` is used to help CSSM efficiently perform the integrity and privilege verification operations. `Hints` helps CSSM know where to look to find the desired state information. In the regular case, CSSM will look for `SrcAddress` in the `CallerList` and `DestAddress` in the `AttachList`. For callback functions, the `SrcAddress` and `DestAddress` are likely to be in `AttachList`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

CSSM_ListAttachedModuleManagers

NAME

CSSM_ListAttachedModuleManagers – Get a list of GUIDs for the attached module manager(CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_ListAttachedModuleManagers
(uint32 *NumberOfModuleManagers,
CSSM_GUID_PTR ModuleManagerGuids)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`NumberOfModuleManagers` (input/output)

The number of GUIDs in the array. If the array is not large enough, then the actual number needed is returned and the error CSSMERR_CSSM_BUFFER_TOO_SMALL is returned. The caller should then allocate an appropriately sized list and call the function again. If the supplied list is larger than needed, the number of module managers found is returned and no error is set.

`ModuleManagerGuids` (input/output)

A pointer to an array of CSSM_GUID structures, one per active module manager. The caller allocates this array.

DESCRIPTION

This function returns a list of GUIDs for the currently attached and active module managers in the CSSM environment.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_BUFFER_TOO_SMALL
CSSMERR_CSSM_INVALID_GUID
```

SEE ALSO

Intel CDSA Application Developer's Guide

CSSM_ModuleAttach

NAME

CSSM_ModuleAttach – Attach and verify a service provider module (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_ModuleAttach
(const CSSM_GUID *ModuleGuid,
const CSSM_VERSION *Version,
const CSSM_API_MEMORY_FUNCS *MemoryFuncs,
uint32 SubserviceID,
CSSM_SERVICE_TYPE SubServiceType,
CSSM_ATTACH_FLAGS AttachFlags,
CSSM_KEY_HIERARCHY KeyHierarchy,
CSSM_FUNC_NAME_ADDR *FunctionTable,
uint32 NumFunctionTable,
const void *Reserved,
CSSM_MODULE_HANDLE_PTR NewModuleHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`ModuleGuid` (*input*)

A pointer to the CSSM_GUID structure containing the global unique identifier for the CSP module.

`Version` (*input*)

The major and minor version number of CDSA that the application is compatible with.

`MemoryFuncs` (*input*)

A structure containing pointers to the memory routines.

`SubserviceID` (*input*)

A `SubServiceID` identifying a particular subservice within the module. Subservice IDs can be obtained from MDS or gleaned from insertion events reported through the `callback` function installed through `CSSM_ModuleLoad()`. Modules that provide only one service can use zero as their subservice ID.

`SubServiceType` (*input*)

A service mask describing the type of service the caller is requesting of the service provider module.

`AttachFlags` (*input*)

A mask representing the caller's request for session-specific services.

`KeyHierarchy` (*input*)

The CSSM_KEY_HIERARCHY option directing CSSM what embedded key to use when verifying integrity of the named module.

`FunctionTable` (input/output/optional)

A table of function-name and API function-pointer pairs. The caller provides the name of the functions as input. The corresponding API function pointers are returned on output. The function table allows dynamic linking of CDSA interfaces, including interfaces to Elective Module Managers (EMMs), which are transparently loaded by CSSM during `CSSM_ModuleAttach()`.

`NumFunctionTable` (*input*)

The number of entries in the `FunctionTable` parameter. If no `FunctionTable` is provided, this value must be zero.

`Reserved` (*input*)

This field is reserved for future use. It should always be set to zero

`NewModuleHandle` (*output*)

A new module handle that can be used to interact with the requested service provider. The value will be set to CSSM_INVALID_HANDLE if the function fails.

DESCRIPTION

This function attaches the service provider module and verifies that the version of the module expected by the application is compatible with the version on the system. The module can implement subservices (described in your service provider's documentation). The caller can specify a specific subservice provided by the module.

If the subservice is supported as part of the CSSM framework as well as by an EMM, `ModuleAttach` attaches the Service Provider to the CSSM framework. If the subservice is supported only by an EMM, `ModuleAttach` loads the appropriate EMM. The service provider is given an indication of whether it is being attached to the CSSM framework or an EMM.

The caller can provide a function table containing function names for the desired services. On output each function name is matched with an API function pointer. The caller can use the pointers to invoke service module operations through CSSM.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_INVALID_ADDIN_FUNCTION_TABLE
CSSMERR_CSSM_EMM_AUTHENTICATE_FAILED
CSSMERR_CSSM_ADDIN_AUTHENTICATE_FAILED
CSSMERR_CSSM_INVALID_SERVICE_MASK
CSSMERR_CSSM_MODULE_NOT_LOADED
CSSMERR_CSSM_INVALID_SUBSERVICEID
CSSMERR_CSSM_INVALID_KEY_HIERARCHY
CSSMERR_CSSM_INVALID_GUID
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_ModuleDetach*

CSSM_ModuleDetach

NAME

CSSM_ModuleDetach – Detach application from service provider module (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_ModuleDetach
(CSSM_MODULE_HANDLE ModuleHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`ModuleHandle` (*input*)

The handle that describes the service provider module.

DESCRIPTION

This function detaches the application from the service provider module.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_ModuleAttach*

CSSM_ModuleLoad

NAME

CSSM_ModuleLoad – Initialize the security service module (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_ModuleLoad
(const CSSM_GUID *ModuleGuid,
CSSM_KEY_HIERARCHY KeyHierarchy,
CSSM_API_ModuleEventHandler AppNotifyCallback,
void* AppNotifyCallbackCtx)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`ModuleGuid` (*input*)

The GUID of the module selected for loading.

`KeyHierarchy` (*input*)

The CSSM_KEY_HIERARCHY option directing CSSM what embedded key to use when verifying integrity of the named module.

`AppNotifyCallback` (input/optional)

The event notification function provided by the caller. This defines the callback for event notifications from the loaded (and later attached) service module.

`AppNotifyCallbackCtx` (input/optional)

When the selected service module raises an event, this context is passed as an input to the event handler specified by `AppNotifyCallback`. CSSM does not interpret or modify the value of `AppNotifyCallbackCtx`.

DESCRIPTION

This function initializes the security service module. Initialization includes registering the application's module-event handler and enabling events with the security service service module. The application can choose to provide an event handler function to receive notification of insert, remove, and fault events. The specified event handler is the single callback point for all attached sessions with the specified service module.

The function `CSSM_Init()` must be invoked prior to calling `CSSM_ModuleLoad()`. The function `CSSM_ModuleAttach()` can be invoked multiple times per call to `CSSM_ModuleLoad()`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_INVALID_GUID
CSSMERR_CSSM_ADDIN_LOAD_FAILED
CSSMERR_CSSM_EMM_LOAD_FAILED
CSSMERR_CSSM_INVALID_KEY_HIERARCHY
```

SEE ALSO

Intel CDSA Application Developer's Guide

CSSM_ModuleUnload

NAME

CSSM_ModuleUnload – Deregister event notification callbacks (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_ModuleUnload
(const CSSM_GUID *ModuleGuid,
CSSM_API_ModuleEventHandler AppNotifyCallback,
void* AppNotifyCallbackCtx)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`ModuleGuid` (*input*)

The GUID of the module selected for unloading.

`AppNotifyCallback` (input/optional)

The event notification function to be deregistered. The function must have been provided by the caller in `CSSM_ModuleLoad()`.

`AppNotifyCallbackCtx` (input/optional)

The event notification context that was provided in the corresponding call to `CSSM_ModuleLoad()`.

DESCRIPTION

The function deregisters event notification callbacks for the caller identified by `ModuleGuid`. The `CSSM_ModuleUnload()` function is the analog call to `CSSM_ModuleLoad()`. If all callbacks registered with CSSM are removed, then CSSM unloads the service module that was loaded by calls to `CSSM_ModuleLoad()`. Calls to `CSSM_ModuleUnload()` that are not matched with a previous call to `CSSM_ModuleLoad()` result in an error.

The CSSM uses the three input parameters `ModuleGuid`, `AppNotifyCallback`, and `AppNotifyCallbackCtx` to uniquely identify registered callbacks.

This function should be invoked after all necessary calls to `CSSM_ModuleDetach()` have been performed.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_ADDIN_UNLOAD_FAILED
CSSMERR_CSSM_EMM_UNLOAD_FAILED
CSSMERR_CSSM_EVENT_NOTIFICATION_CALLBACK_NOT_FOUND
```

SEE ALSO

Intel CDSA Application Developer's Guide

cssm_ReleaseAttachFunctions

NAME

cssm_ReleaseAttachFunctions – Release lock on the SP function table (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI cssm_ReleaseAttachFunctions
(CSSM_MODULE_HANDLE hAddIn)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`hAddIn` (*input*)

The handle identifying the attach-session whose function table is to be released by this function.

DESCRIPTION

This function releases the lock on the SP function table for the service module identified by the module handle. The SPI function table was obtained by the elective module manager through the `cssm_GetAttachFunctions()` operation.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

CSSM_SetContext

NAME

CSSM_SetContext – Replace all context information (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_SetContext
(CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CCHandle` (*input*)

The handle to the context.

`Context` (*input*)

The context data describing the service to replace the current service associated with context handle CCHandle.

DESCRIPTION

This function replaces all context information associated with an existing context specified by `CCHandle`. The contents of the basic context structure and all attributes included in that structure are replaced by the context structure and attribute values contained in the `Context` input parameter.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_INVALID_ATTRIBUTE
CSSMERR_CSSM_INVALID_CONTEXT_HANDLE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_GetContext*

CSSM_SetPrivilege

NAME

CSSM_SetPrivilege – Store privilege value in CSSM framework (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_SetPrivilege
(CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`Privilege` (*input*)

The CSSM_PRIVILEGE value to be applied to subsequent calls to CSSM interfaces.

DESCRIPTION

The `CSSM_SetPrivilege()` function accepts as input a privilege value and stores it in the CSSM framework. The integrity credentials of the module calling `CSSM_SetPrivilege()` must be verified by CSSM before the privilege value is updated. Integrity credentials are established using `CSSM_Introduce()`. CSSM will perform a pointer validation check to ensure the caller has been previously introduced. The `CSSM_SetPrivilege()` function will fail if no integrity information can be found for the caller.

After pointer validation checks, CSSM verifies the requested privilege is authorized. This is done by comparing `Privilege` with the set of privileges contained in the caller manifest. If `Privilege` is not a member, the `CSSM_SetPrivilege()` call fails.

Subsequent calls to the framework that require privileges inherit the privilege value previously established by `CSSM_SetPrivilege()`. CSSM will perform pointer validation checks on the API caller before servicing the API call. If OK, then the `Privilege` value is supplied to the SPI function.

Internally, CSSM builds and maintains privilege information based on the chosen scope of the implementation. The scope may be dictated by the capabilities of the platform hosting the CSSM. If threading is available, the privilege value can be associated with the thread ID of the currently executing thread. In this scenario, CSSM can manage a table of tuples consisting of `threadID` and privilege value. If threading is not available, the privilege value can be global to the process.

Because the selected privilege value is shared, the application programmer should take precautions to reset the privilege value whenever program flow leaves the caller's module and again when control flow returns. In general, any time there is a possibility for `CSSM_SetPrivilege()` to be called while within the context of the security critical section, `CSSM_SetPrivilege()` should be called again. Otherwise, the module receiving execution control could have called `CSSM_SetPrivilege()`, resulting in the privilege value being reset.

Data structures used to maintain the global privilege value should be initialized in `CSSM_Init()`. This includes lock initialization and preliminary resource allocation. The `CSSM_Init()` function is assumed to be idempotent with respect to shared structure initialization. This means `CSSM_Init()` will ensure a single thread initializes the shared structure and subsequent calls to `CSSM_Init()` will not reinitialize it. A reference count of calls to `CSSM_Init()` is needed to ensure matching calls to `CSSM_Terminate()` are handled.

Resource cleanup is performed at `CSSM_Terminate()` after the reference count falls to zero. The last call to `CSSM_Terminate()` results in shared resources being freed and lock structures being released.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

CSSM_SPI_ModuleAttach

NAME

CSSM_SPI_ModuleAttach – Attach a service provider module(CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMSPI CSSM_SPI_ModuleAttach
(const CSSM_GUID *ModuleGuid,
const CSSM_VERSION *Version,
uint32 SubserviceID,
CSSM_SERVICE_TYPE SubServiceType,
CSSM_ATTACH_FLAGS AttachFlags,
CSSM_MODULE_HANDLE ModuleHandle,
CSSM_KEY_HIERARCHY KeyHierarchy,
const CSSM_GUID *CssmGuid,
const CSSM_GUID *ModuleManagerGuid,
const CSSM_GUID *CallerGuid,
const CSSM_UPCALLS *Upcalls,
CSSM_MODULE_FUNCS_PTR *FuncTbl)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`ModuleGuid` (*input*)

The CSSM_GUID of the invoked service provider module.

`Version` (*input*)

The major and minor version number of the required level of system services and features. The service module must determine whether its services are compatible with the required version.

`SubserviceId` (*input*)

The identifier for the requested subservice within this module. If only one service is provided by the module, then `subserviceId` can be zero.

`SubServiceType` (*output*)

A CSSM_SERVICE_MASK indicating the type of services provided by the service module and the ordering of the function table returned in the output parameter `FuncTbl`.

`AttachFlags` (*input*)

A mask representing the caller's request for session-specific services.

`ModuleHandle` (*input*)

The CSSM_HANDLE value assigned by CSSM and associated with the attach session being created by this function.

`KeyHierarchy` (*input*)

The CSSM_KEY_HIERARCHY option directing CSSM which embedded key or keys to use when verifying integrity of the named modules.

`CssmGuid` (*input*)

The CSSM_GUID of the CSSM invoking this function.

`ModuleManagerGuid` (*input*)

The CSSM_GUID of the module that will route calls to the service provider.

`CallerGuid` (*input*)

The CSSM_GUID of the caller who invoked `CSSM_ModuleAttach()`, which resulted in CSSM invoking this function.

`Upcalls` (*input*)

A set of function pointers the service module must use to obtain selected CSSM services and to manage application memory. The memory management functions are provided when the application invokes `CSSM_ModuleAttach()`. CSSM forwards these function pointers with CSSM service function pointers to the module.

`FuncTbl` (*output*)

A CSSM_MODULE_FUNCS table containing pointers to the service module functions the caller can use. CSSM uses this table to proxy calls from an application caller to the add-in service module.

DESCRIPTION

This function is invoked by CSSM once for each invocation of `CSSM_ModuleAttach()`, specifying the module identified by `ModuleGuid`. Four entities are stakeholders in this function and each is identified by a CSSM_GUID value:

Service Module

The executing service provider performing the `CSSM_SPI_ModuleAttach()` operation. The module is identified by `ModuleGuid`.

CSSM

The CSSM that invoked the service module. CSSM is identified by `CssmGuid`.

`ModuleManagerGuid`

The module that will be routing calls to the service provider. This value will be the same as `CssmGuid` if CSSM is managing the calls to this service provider.

Caller

The entity that invoked CSSM through the `CSSM_ModuleAttach()` function. The caller is identified by `CallerGuid`.

The service provider module should perform an integrity check of CSSM. `CssmGuid` can be used to locate CSSM's signed manifest credentials. The service provider can require an integrity check of the Caller. The `CallerGuid` parameter can be used to locate the Caller's signed manifest credentials. The `KeyHierarchy` flag identifies the class of embedded public keys CSSM will use to check the integrity of the service provider. If the manifest for the target module does not encounter an embedded key for all the key classes in `KeyHierarchy`, the integrity cross-check fails.

The service module must verify compatibility with the system version level specified by `Version`. If the version is not compatible, then this function fails. The service module should perform all initializations required to support the new attached session and should return a function table for the SPI entry points that can be invoked by CSSM in response to API invocations by `CallerGuid`. CSSM uses this function table to dispatch requests for the attach session created by this function. Each attach session has its own function table.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_SPI_ModuleDetach*, *CSSM_SPI_ModuleLoad*

CSSM_SPI_ModuleDetach

NAME

CSSM_SPI_ModuleDetach – Notify service module of a context event (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMSPI CSSM_SPI_ModuleDetach
(CSSM_MODULE_HANDLE ModuleHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`ModuleHandle` (*input*)

The CSSM_HANDLE value associated with the attach session being terminated by this function.

DESCRIPTION

This function is invoked by CSSM once for each invocation of `CSSM_ModuleDetach()` specifying the attach-session identified by `ModuleHandle`. The function entry point for `CSSM_SPI_ModuleDetach` is included in the module function table `CSSM_MODULE_FUNCS` returned to CSSM as output of a successful `CSSM_SPI_ModuleAttach`.

The service module must perform all cleanup operations associated with the specified attach handle.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_SPI_ModuleAttach, CSSM_SPI_ModuleUnload*

CSSM_SPI_ModuleLoad

NAME

CSSM_SPI_ModuleLoad – Initialize process between CSSM and the add-in service module (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMSPI CSSM_SPI_ModuleLoad
(const CSSM_GUID *CssmGuid,
const CSSM_GUID *ModuleGuid,
CSSM_SPI_ModuleEventHandler CssmNotifyCallback,
void* CssmNotifyCallbackCtx)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CssmGuid` (*input*)

The CSSM_GUID of the caller. Used to locate the caller's signed manifest credentials.

`ModuleGuid` (*input*)

The CSSM_GUID of the invoked service provider module. Used to locate the module's signed manifest credentials.

`CssmNotifyCallback` (*input*)

A function pointer for the CSSM event handler that manages events of type CSSM_MODULE_EVENT.

`CssmNotifyCallbackCtx` (*input*)

The context to be returned to CSSM as input on each callback to the event handler defined by `CssmNotifyCallback`.

DESCRIPTION

This function completes the module initialization process between CSSM and the add-in service module. Before invoking this function, CSSM verifies the add-in service module's manifest credentials. If the credentials verify this module is loaded (physically if required), the `CSSM_SPI_ModuleLoad()` function is invoked.

The `CssmGuid` parameter identifies the caller and should be used by the module to locate the caller's signed manifest credentials and to complete integrity verification and secure linkage checks on the caller. The `ModuleGuid` identifies the invoked module and should be used by the module to locate its credentials and to complete an integrity self-check.

The `CssmNotifyCallback` and `CssmNotifyCallbackCtx` parameters define a callback and callback context respectively. The module must retain this information for later use. The module should use the callback to notify CSSM of module events of type CSSM_MODULE_EVENT in any ongoing, attached sessions.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_SPI_ModuleAttach, CSSM_SPI_ModuleUnload*

CSSM_SPI_ModuleUnload

NAME

CSSM_SPI_ModuleUnload – Disable events and deregister CSSM event notification (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMSPI CSSM_SPI_ModuleUnload
(const CSSM_GUID *CssmGuid,
const CSSM_GUID *ModuleGuid,
CSSM_SPI_ModuleEventHandler CssmNotifyCallback,
void* CssmNotifyCallbackCtx)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CssmGuid` (*input*)

The CSSM_GUID of the caller.

`ModuleGuid` (*input*)

The CSSM_GUID of the invoked service provider module.

`CssmNotifyCallback` (*input*)

A function pointer for the CSSM event handler that manages events of type CSSM_MODULE_EVENT.

`CssmNotifyCallbackCtx` (*input*)

The context to be returned to CSSM as input on each callback to the event handler defined by `CssmNotifyCallback`.

DESCRIPTION

This function disables events and deregisters the CSSM `event-notification` function. The add-in service module can perform cleanup operations, reversing the initialization performed in `CSSM_SPI_ModuleLoad()`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_SPI_ModuleDetach*, *CSSM_SPI_ModuleLoad*

CSSM_Terminate

NAME

CSSM_Terminate – Terminate the use of CSSM (CDSA)

SYNOPSIS

include <cssm.h>

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

None

DESCRIPTION

This function terminates the caller's use of CSSM. CSSM can clean up all internal states associated with the calling application. This function must be called once by each application.

`CSSM_Terminate()` must be called one time for each time `CSSM_Init()` was previously called.

CSSM services remain available to the program until the final call to `CSSM_Terminate()` completes. After that final call, all information introduced by the caller (including privileges, handles, contexts, introduced libraries, and so forth) is lost, and it is an error to subsequently call any CSSM API function other than `CSSM_Init()`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions *CSSM_Init*

CSSM_TP_RetrieveCredResult

NAME

CSSM_TP_RetrieveCredResult – Return the results of the credentials request (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_TP_RetrieveCredResult
(CSSM_TP_HANDLE TPHandle,
const CSSM_DATA *ReferenceIdentifier,
const CSSM_TP_CALLERAUTH_CONTEXT *CallerAuthCredentials,
sint32 *EstimatedTime,
CSSM_BOOL *ConfirmationRequired,
CSSM_TP_RESULT_SET_PTR *RetrieveOutput)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the certification authority module used to perform this function.

`ReferenceIdentifier` (*input*)

A reference identifier that uniquely identifies the `CSSM_TP_SubmitCredRequest()` call that initiated the certificate service request whose results are returned by this function. The identifier persists across application executions and becomes undefined when all local processing of the request has completed.

Local processing is completed in one of two ways:

- For certificate services that do not require explicit confirmation by the requester, the reference identifier is invalidated when the corresponding `CSSM_TP_RetrieveCredResult()` function completes (by returning valid results or by failure, which blocks returned results).
- For certificate services that require explicit confirmation by the requester, the reference identifier is invalidated by successfully invoking the function `CSSM_TP_ConfirmCredResult()`.

`CallerAuthCredentials` (input/optional)

This structure contains a set of caller authentication credentials. The authentication information can be a passphrase, a PIN, a completed registration form, a certificate, or a template of user-specific data. The required set of credentials is defined by the service provider module and recorded in a record in the MDS Primary relation. Multiple credentials can be required. If the local service provider module does not require credentials from a caller, then the `Credentials` field of this verification context structure can be NULL. The structure optionally contains additional credentials that can be used to support the

authentication process. Authentication credentials required by the authority should be included in the `RequestInput`. The local TP module can forward information from `CallerAuthCredentials` to the authority, as appropriate, but is not required to do so.

`EstimatedTime` (*output*)

The number of seconds estimated before the results of a requested service will be returned to the requester. When the local TP module or the authority process cannot estimate the time required to perform the requested service, the output value for estimated time is CSSM_ESTIMATED_TIME_UNKNOWN.

`ConfirmationRequired` (*output*)

A Boolean value indicating whether the caller must invoke `CSSM_TP_ConfirmCredResult()` to acknowledge retrieving the results of the service request. CSSM_TRUE indicates the caller must call `CSSM_TP_ConfirmCredResult()`. CSSM_FALSE indicates that the caller must not call `CSSM_TP_ConfirmCredResult()`. The value of this output parameter is not applicable until `CSSM_TP_RetrieveCredResult()` completes by returning results of the request or terminates in unrecoverable failure.

`RetrieveOutput` (*output*)

A pointer to the results returned by the authority in response to the service requests submitted by `CSSM_TP_SubmitCredRequest()`. The output results are ordered corresponding to the requests. The structure of the response set is determined by the type of request. The caller and the service provider must retain knowledge of the request type associated with the `ReferenceIdentifier`.

DESCRIPTION

This function returns the results of a `CSSM_TP_SubmitCredRequest()` call.

The single identifier `ReferenceIdentifier` denotes the `CSSM_TP_SubmitCredRequest()` invocation that initiated the request.

It is possible that the results are not ready to be retrieved when this call is made. In that case, an `EstimatedTime` to complete processing is returned. The caller must attempt to retrieve the results again after the estimated time to completion has elapsed.

This function can fail in total for any one of the following reasons:

- The reference identifier is invalid.
- The TP process cannot be located.
- The TP process encountered a fatal error when attempting to process the requests.

When this function completes, the set of return results is ordered corresponding to the order of the originating request.

Some certificate services require the requester to confirm retrieval of the results. The `ConfirmationRequired` parameter indicates whether the caller must confirm completion of `CSSM_TP_RetrieveCredResult()` by calling `CSSM_TP_ConfirmCredResult()`.

RETURN VALUE

A CSSM_RETURN value combined with estimated time to indicate one of three results:

Complete Function	Function Return	RetrieveOutput	EstimatedTime
Result	Value		
Request results returned to caller	CSSM_OK	Non-NULL pointer	NA
Request results not ready, but expected in the future	CSSM_OK	NULL pointer	CSSM_ESTIMATED_TIME_ UNKNOWN or <estimated seconds>
Fatal Error, results will never be returned	(!CSSM_OK)	NA	NA

The (!CSSM_OK) return value represents a specific error code.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_IDENTIFIER_POINTER
CSSMERR_TP_INVALID_IDENTIFIER
CSSMERR_TP_INVALID_CALLERAUTH_CONTEXT_POINTER
CSSMERR_TP_INVALID_POLICY_IDENTIFIERS
CSSMERR_TP_INVALID_TIMESTRING
CSSMERR_TP_INVALID_STOP_ON_POLICY
CSSMERR_TP_INVALID_CALLBACK
CSSMERR_TP_INVALID_ANCHOR_CERT
CSSMERR_TP_CERTGROUP_INCOMPLETE
CSSMERR_TP_INVALID_DL_HANDLE
CSSMERR_TP_INVALID_DB_HANDLE
CSSMERR_TP_INVALID_DB_LIST_POINTER
CSSMERR_TP_INVALID_DB_LIST
CSSMERR_TP_AUTHENTICATION_FAILED
CSSMERR_TP_INSUFFICIENT_CREDENTIALS
CSSMERR_TP_NOT_TRUSTED
CSSMERR_TP_CERT_REVOKED
CSSMERR_TP_CERT_SUSPENDED
CSSMERR_TP_CERT_EXPIRED
CSSMERR_TP_CERT_NOT_VALID_YET
CSSMERR_TP_INVALID_CERT_AUTHORITY
CSSMERR_TP_INVALID_SIGNATURE
CSSMERR_TP_INVALID_NAME
CSSMERR_TP_REQUEST_LOST
CSSMERR_TP_REQUEST_REJECTED
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_SubmitCredRequest

Functions for the TP SPI:

TP_SubmitCredRequest

CSSM_Unintroduce

NAME

CSSM_Unintroduce – Remove module (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSM_Unintroduce
(const CSSM_GUID *ModuleID)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`ModuleID` (*input*)

The CSSM_GUID of the calling library or other library that can call CDSA interfaces. The GUID is used to locate the module integrity and privilege information. If the `ModuleID` is NULL, then the caller will be unintroduced.

DESCRIPTION

The `CSSM_Unintroduce()` function removes the module referenced by `ModuleID` from the list of module information maintained by the CSSM framework.

A caller can unintroduce modules other than itself if the caller has been previously introduced.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSSM_INVALID_GUID`

SEE ALSO

Intel CDSA Application Developer's Guide

CSSM_UpdateContextAttributes

NAME

CSSM_UpdateContextAttributes – Update context attribute values (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_UpdateContextAttributes
(CSSM_CC_HANDLE CCHandle,
uint32 NumberOfAttributes,
const CSSM_CONTEXT_ATTRIBUTE *ContextAttributes)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CCHandle` (*input*)

The handle to the existing context.

`NumberOfAttributes` (*input*)

The number of CSSM_CONTEXT_ATTRIBUTE structures to allocate.

`ContextAttributes` (*input*)

Pointer to data that describes the attributes to be associated with this context.

DESCRIPTION

This function updates one or more context attribute values stored as part of an existing context specified by `CCHandle`. The basic context structure is not modified by this function. Only the context attributes are updated.

The `NumberOfAttributes` parameter specifies the number of attributes to update. The new attribute values are specified in `ContextAttributes`. If an attribute provided in `ContextAttributes` is already present in the existing context, the existing value is replaced by the new value. If an attribute provided in `ContextAttributes` is not present in the existing context, then the new attribute is added. Attribute values are never deleted from the existing context.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_INVALID_CONTEXT_HANDLE
CSSMERR_CSSM_INVALID_ATTRIBUTE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_DeleteContextAttributes*, *CSSM_GetContextAttribute*

Decode_CDSA_Error

NAME

Decode_CDSA_Error – Accepts a CDSA numeric error code and returns two strings: the ASCII name of the error and a description of the error

SYNOPSIS

```
#include <cssmerr.h>

API:

void Decode_CDSA_Error(Error_Code,Error_Label_String,Error_String)
     CSSM_RETURN Error_Code;
     char *Error_Label_String;
     char *Error_String;
```

DESCRIPTION

This function accepts a CDSA numeric error code and returns two strings: the ASCII name of the error and a description of the error.

RETURN VALUE

None.

DecryptData

NAME

DecryptData: CSSM_DecryptData, CSP_DecryptData – Decrypt buffer data (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DecryptData
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CipherBufs,
uint32 CipherBufCount,
CSSM_DATA_PTR ClearBufs,
uint32 ClearBufCount,
uint32 *bytesDecrypted,
CSSM_DATA_PTR RemData)
SPI:
CSSM_RETURN CSSMCSPI CSP_DecryptData
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
const CSSM_DATA *CipherBufs,
uint32 CipherBufCount,
CSSM_DATA_PTR ClearBufs,
uint32 ClearBufCount,
uint32 *bytesDecrypted,
CSSM_DATA_PTR RemData,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`CipherBufs` (*input*)

A pointer to a vector of CSSM_DATA structures that contain the data to be decrypted.

`CipherBufCount` (*input*)

The number of `CipherBufs`.

`ClearBufs` (*output*)

A pointer to a vector of CSSM_DATA structures that contain the decrypted data resulting from the decryption operation.

`ClearBufCount` (*input*)

The number of `ClearBufs`.

`bytesDecrypted` (*output*)

A pointer to `uint32` for the size of the decrypted data in bytes.

`RemData` (*output*)

A pointer to the CSSM_DATA structure for the remaining plain text if there is not enough buffer space available in the output data structures.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

A pointer to CSSM_CONTEXT structure that describes the attributes with this context.

`Privilege` (*input*)

The export privilege to be applied during the cryptographic operation. This parameter is forwarded to the CSP after CSSM verifies the caller and service provider privilege set includes the specified PRIVILEGE.

DESCRIPTION

This function decrypts all data contained in the set of input buffers using information in the context. The `CSSM_QuerySize()` (CSSM API), or `CSP_QuerySize()` (CSP SPI), function can be used to estimate the output buffer size required. The minimum number of buffers required to contain the resulting plain text is produced as output. If the plain text result does not fit within the set of output buffers, the remaining plain text is returned in the single output buffer `RemData`.

The CSP can require that the cryptographic context include access credentials for authentication and authorization checks when using a private key or a secret key.

NOTES FOR API

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, pre-allocated output buffer, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value greater than zero and a non-NULL data pointer field value. To specify automatic output buffer allocation by the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed. In-place decryption can be done by supplying the same input and output buffers.

NOTES FOR SPI

The output is returned to the caller as specified in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_BLOCK_SIZE_MISMATCH`
`CSSMERR_CSP_OUTPUT_LENGTH_ERROR`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_QuerySize, *CSSM_EncryptData*, *CSSM_DecryptDataInit*, *CSSM_DecryptDataUpdate*, *CSSM_DecryptDataFinal*, *CSSM_DecryptP*, *CSSM_DecryptDataInitP*

Functions for the CSP SPI:

CSP_QuerySize, *CSP_EncryptData*, *CSP_DecryptDataInit*, *CSP_DecryptDataUpdate*, *CSP_DecryptDataFinal*

DecryptDataFinal

NAME

DecryptDataFinal: CSSM_DecryptDataFinal, CSP_DecryptDataFinal – Finalize staged decryption process (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DecryptDataFinal
(CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR RemData)
SPI:
CSSM_RETURN CSSMCSPI CSP_DecryptDataFinal
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR RemData)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`RemData` (*output*)

A pointer to the CSSM_DATA structure for the last decrypted block, if necessary.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function finalizes the staged decryption process by returning any remaining plain text not returned in the previous staged decryption call. The plain text is returned in a single buffer.

NOTES FOR API

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, pre-allocated output buffer, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value greater than zero and a non-NULL data pointer field value. To specify automatic output buffer

allocation by the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed.

NOTES FOR SPI

The output is returned to the caller as specified in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_BLOCK_SIZE_MISMATCH`
`CSSMERR_CSP_OUTPUT_LENGTH_ERROR`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DecryptData, *CSSM_DecryptDataInit*, *CSSM_DecryptDataUpdate*

Functions for the CSP SPI:

CSP_DecryptData, *CSP_DecryptDataInit*, *CSP_DecryptDataUpdate*

DecryptDataInit

NAME

DecryptDataInit: CSSM_DecryptDataInit, CSP_DecryptDataInit – Initialize the staged decrypt function(CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DecryptDataInit
(CSSM_CC_HANDLE CCHandle)
SPI:
CSSM_RETURN CSSMCSPI CSSM_CSP_DecryptDataInit
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

`Privilege` (*input*)

The export privilege to be applied during the cryptographic operation. This parameter is forwarded to the CSP after CSSM verifies the caller and service provider privilege set includes the specified PRIVILEGE.

DESCRIPTION

This function initializes the staged decrypt function.

The CSP can require that the cryptographic context include access credentials for authentication and authorization checks when using a private key or a secret key.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DecryptData, *CSSM_DecryptDataUpdate*, *CSSM_DecryptDataFinal*, *CSSM_DecryptDataP*, *CSSM_DecryptDataInitP*

Functions for the CSP SPI:

CSP_DecryptData, *CSP_DecryptDataUpdate*, *CSP_DecryptDataFinal*

DecryptDataInitP

NAME

DecryptDataInitP – Intialize the staged decrypt function with privilege (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_DecryptDataInitP
(CSSM_CC_HANDLE CCHandle,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`Privilege` (*input*)

The privilege to be applied during the cryptographic operation.

See `CSSM_DecryptDataInit()` for other parameters.

DESCRIPTION

This function is similar to `CSSM_DecryptDataInit()`. It also accepts a USEE tag as a privilege request parameter. CSSM checks that either its own privilege set or the application's privilege set (if the application is signed) includes the tag. If the tag is found and the service provider privilege set indicates that it is supported, the tag is forwarded to the service provider.

For staged operations using privilege initialization functions `CSSM_DecryptDataInitP()`, the completion functions `CSSM_DecryptDataUpdate()` and `CSSM_DecryptDataFinalize()` are used.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_DecryptData*, *CSSM_EncryptDataInit*, *CSSM_EncryptDataUpdate*, *CSSM_EncryptDataFinal*, *CSSM_EncryptDataP*, *CSSM_EncryptDataInitP*, *CSSM_DecryptP*, *CSSM_DecryptDataInitP*, *CSSM_QuerySize*

DecryptDataP

NAME

DecryptDataP – Decrypt data with privilege (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_DecryptDataP
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CipherBufs,
uint32 CipherBufCount,
CSSM_DATA_PTR ClearBufs,
uint32 ClearBufCount,
uint32 *bytesDecrypted,
CSSM_DATA_PTR RemData,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`Privilege` (*input*)

The privilege to be applied during the cryptographic operation.

See `CSSM_DecryptData()` for other parameters.

DESCRIPTION

This function is similar to `CSSM_DecryptData()`. It also accepts a USEE tag as a privilege request parameter. CSSM checks that either its privilege set or the application's privilege set (if the application is signed) includes the tag. If the tag is found and the service provider privilege set indicates that it is supported, the tag is forwarded to the service provider.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSP_BLOCK_SIZE_MISMATCH
CSSMERR_CSP_OUTPUT_LENGTH_ERROR
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_DecryptData*, *CSSM_EncryptDataInit*, *CSSM_EncryptDataUpdate*, *CSSM_EncryptDataFinal*, *CSSM_EncryptDataP*, *CSSM_EncryptDataInitP*, *CSSM_DecryptP*, *CSSM_DecryptDataInitP*, *CSSM_QuerySize*

DecryptDataUpdate

NAME

DecryptDataUpdate: CSSM_DecryptDataUpdate, CSP_DecryptDataUpdate – Continue the staged decryption process (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DecryptDataUpdate
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CipherBufs,
uint32 CipherBufCount,
CSSM_DATA_PTR ClearBufs,
uint32 ClearBufCount,
uint32 *bytesDecrypted)
SPI:
CSSM_RETURN CSSMCSPI CSP_DecryptDataUpdate
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CipherBufs,
uint32 CipherBufCount,
CSSM_DATA_PTR ClearBufs,
uint32 ClearBufCount,
uint32 *bytesDecrypted)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`CipherBufs` (*input*)

A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

`CipherBufCount` (*input*)

The number of `CipherBufs`.

`ClearBufs` (*output*)

A pointer to a vector of CSSM_DATA structures that contain the decrypted data resulting from the decryption operation.

`ClearBufCount` (*input*)

The number of `ClearBufs`.

`bytesDecrypted` (*output*)

A pointer to `uint32` for the size of the decrypted data in bytes.

SPI PARAMETER

`CSPHandle (`*input*`)`

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function continues the staged decryption process over all data in the set of input buffers. There can be algorithm-specific and token-specific rules restricting the lengths of data in `CSSM_DecryptUpdate()` calls, but multiple input buffers are supported. The minimum number of buffers required to contain the resulting plain text is produced as output. Excess output buffer space is not remembered across staged decryption calls. Each staged call begins filling one or more new output buffers. The `CSSM_QuerySize()` (CSSM API), or `CSP_QuerySize()` (CSP SPI), function can be used to estimate the output buffer size required for each update call.

NOTES FOR API

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, preallocated output buffer, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value greater than zero and a non-NULL data pointer field value. To specify automatic output buffer allocation by the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed. In-place decryption can be done by supplying the same input and output buffers.

NOTES FOR SPI

The output is returned to the caller as specified in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_QuerySize, *CSSM_DecryptData*, *CSSM_DecryptDataInit*, *CSSM_DecryptDataFinal*

Functions for the CSP SPI:

CSP_QuerySize, *CSP_DecryptData*, *CSP_DecryptDataInit*, *CSP_DecryptDataFinal*

DeriveKey

NAME

DeriveKey: CSSM_DeriveKey, CSP_DeriveKey – Derive new symmetric key (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DeriveKey
(CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR Param,
uint32 KeyUsage,
uint32 KeyAttr,
const CSSM_DATA *KeyLabel,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry,
CSSM_KEY_PTR DerivedKey)
SPI:
CSSM_RETURN CSSMCSPI CSP_DeriveKey
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
CSSM_DATA_PTR Param,
uint32 KeyUsage,
uint32 KeyAttr,
const CSSM_DATA *KeyLabel,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry,
CSSM_KEY_PTR DerivedKey)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation.

`Param` (input/output)

This parameter varies depending on the derivation algorithm. Password based derivation algorithms use this parameter to return a cipher block chaining initialization vector. Concatenation algorithms use this parameter to get the second item to concatenate.

`KeyUsage` (*input*)

A bit mask indicating all permitted uses for the new derived key.

`KeyAttr` (*input*)

A bit mask defining other attribute values for the new derived key.

`KeyLabel` (input/optional)

Pointer to a byte string that will be used as the label for the derived key.

`CredAndAclEntry` (input/optional)

A structure containing one or more credentials authorized for creating a key and the prototype ACL entry that will control future use of the newly created key. The credentials and ACL entry prototype can be presented as immediate values or callback functions can be provided for use by the CSP to acquire the credentials and/or the subject of the ACL entry interactively. If the CSP provides public access for creating a key, then the credentials can be NULL. If the CSP defines a default initial ACL entry for the new key, then the ACL entry prototype can be empty.

`DerivedKey` (*output*)

A pointer to a CSSM_KEY structure that returns the derived key.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DESCRIPTION

This function derives a new symmetric key using the context and/or information from the base key in the context. The CSP can require that the cryptographic context include access credentials for authentication and authorization checks when using a private key or a secret key.

Authorization policy can restrict the set of callers who can create a new resource. In this case, the caller must present a set of access credentials for authorization. Upon successfully authenticating the credentials, the template that verified the presented samples identifies the ACL entry that will be used in the authorization computation. If the caller is authorized, the new resource is created.

The caller must provide an initial ACL entry to be associated with the newly created resource. This entry is used to control future access to the new resource and (since the subject is deemed to be the "Owner") exercise control over its associated ACL. The caller can specify the following items for initializing an ACL entry:

Subject

A CSSM_LIST structure, containing the type of the subject and a template value that can be used to verify samples that are presented in credentials when resource access is requested.

Delegation flag

A value indicating whether the Subject can delegate the permissions recorded in the `AuthorizationTag`. (This item only applies to public key subjects).

Authorization tag

The set of permissions that are granted to the Subject.

Validity period

The start time and the stop time for which the ACL entry is valid.

ACL entry tag

A user-defined string value associated with the ACL entry.

The service provider can modify the caller-provided initial ACL entry to conform to any innate resource-access policy that the service provider may be required to enforce. If the initial ACL entry provided by the caller contains values or permissions that are not supported by the service provider, then the service provider can modify the initial ACL appropriately or can fail the request to create the new resource. Service providers list their supported `AuthorizationTag` values in their Module Directory Services primary record.

The CSP can require that the cryptographic context include access credentials for authentication and authorization checks when using a private key or a secret key.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_KEY_LABEL_ALREADY_EXISTS`

COMMENTS

The `KeyData` field of the CSSM_KEY structure is allocated by the CSP. The application is required to free this memory using the `CSSM_FreeKey()` (CSSM API), or `CSP_FreeKey()` (CSP SPI) call, or with the memory functions registered for the `CSPHandle`.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_CSP_CreateDeriveKeyContext*

DigestData

NAME

DigestData: CSSM_DigestData, CSP_DigestData – Compute message digest (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DigestData
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount,
CSSM_DATA_PTR Digest)
SPI:
CSSM_RETURN CSSMCSPI CSP_DigestData
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
const CSSM_DATA *DataBufs,
uint32 DataBufCount,
CSSM_DATA_PTR Digest)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`DataBufs` (*input*)

A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

`DataBufCount` (*input*)

The number of DataBufs.

`Digest` (*output*)

A pointer to the CSSM_DATA structure for the message digest.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DESCRIPTION

This function computes a message digest for all data contained in the set of input buffers.

NOTES FOR API

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, preallocated output buffer, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value greater than zero and a non-NULL data pointer field value. To specify automatic output buffer allocation by the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed.

NOTES FOR SPI

The output is returned to the caller as specifed in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_OUTPUT_LENGTH_ERROR`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DigestDataInit, *CSSM_DigestDataUpdate*, *CSSM_DigestDataFinal*, *CSSM_DigestDataClone*

Functions for the CSP SPI:

CSP_DigestDataInit, *CSP_DigestDataUpdate*, *CSP_DigestDataFinal*, *CSP_DigestDataClone*

DigestDataClone

NAME

DigestDataClone: CSSM_DigestDataClone, CSP_DigestDataClone – Clone a staged message digest (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DigestDataClone
(CSSM_CC_HANDLE CCHandle,
CSSM_CC_HANDLE *ClonednewCCHandle)
SPI:
CSSM_RETURN CSSMCSPI CSP_DigestDataClone
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_CC_HANDLE ClonednewCCHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of a staged message digest operation.

`ClonednewCCHandle` (*output*)

The cloned digest context handle. The handle will be set to CSSM_INVALID_HANDLE if the function fails.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function clones a given staged message digest context with its cryptographic attributes and intermediate result.

NOTES

When a digest context is cloned, a new context is created with data associated with the parent context. Changes made to the parent context after calling this function will not be reflected in the cloned context. The cloned context could be used with the `CSSM_DigestDataUpdate()` and `CSSM_DigestDataFinal()` functions.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DigestData, *CSSM_DigestDataInit*, *CSSM_DigestDataUpdate*, *CSSM_DigestDataFinal*

Functions for the CSP SPI:

CSP_DigestData, *CSP_DigestDataInit*, *CSP_DigestDataUpdate*, *CSP_DigestDataFinal*

DigestDataFinal

NAME

DigestDataFinal: CSSM_DigestDataFinal, CSP_DigestDataFinal – Finalize the staged message digest (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DigestDataFinal
(CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR Digest)
SPI:
CSSM_RETURN CSSMCSPI CSP_DigestDataFinal
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR Digest)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`Digest` (*output*)

A pointer to the CSSM_DATA structure for the message digest.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function finalizes the staged message digest function.

NOTES FOR API

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, preallocated output buffer, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value greater than zero and a non-NULL data pointer field value. To specify automatic output buffer allocation by

the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed.

NOTES FOR SPI

The output is returned to the caller as specified in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard. .

`CSSMERR_CSP_OUTPUT_LENGTH_ERROR`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DigestData, *CSSM_DigestDataInit*, *CSSM_DigestDataUpdate*, *CSSM_DigestDataClone*

Functions for the CSP SPI:

CSP_DigestData, *CSP_DigestDataInit*, *CSP_DigestDataUpdate*, *CSP_DigestDataClone*

DigestDataInit

NAME

DigestDataInit: CSSM_DigestDataInit, CSP_DigestDataInit – Initialize the staged message digest (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DigestDataInit
(CSSM_CC_HANDLE CCHandle)
SPI:
CSSM_RETURN CSSMCSPI CSP_DigestDataInit
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DESCRIPTION

This function initializes the staged message digest function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DigestData, *CSSM_DigestDataUpdate*, *CSSM_DigestDataClone*, *CSSM_DigestDataFinal*

Functions for the CSP SPI:

CSP_DigestData, *CSP_DigestDataUpdate*, *CSP_DigestDataClone*, *CSP_DigestDataFinal*

DigestDataUpdate

NAME

DigestDataUpdate: CSSM_DigestDataUpdate – Continue the staged process of digesting (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DigestDataUpdate
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount)
SPI:
CSSM_RETURN CSSMCSPI CSP_DigestDataUpdate
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`DataBufs` (*input*)

A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

`DataBufCount` (*input*)

The number of DataBufs.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function continues the staged process of digesting all data contained in the set of input buffers. The resulting digest value will be returned as part of the staged digesting process.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DigestData, *CSSM_DigestDataInit*, *CSSM_DigestDataClone*, *CSSM_DigestDataFinal*

Functions for the CSP SPI:

Functions: *CSP_DigestData*, *CSP_DigestDataInit*, *CSP_DigestDataClone*, *CSP_DigestDataFinal*

DL_Authenticate

NAME

DL_Authenticate: CSSM_DL_Authenticate – Provide authentication credentials (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_Authenticate
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_DB_ACCESS_TYPE AccessRequest,
const CSSM_ACCESS_CREDENTIALS *AccessCred)
SPI:
CSSM_RETURN CSSMDLI DL_Authenticate
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_DB_ACCESS_TYPE AccessRequest,
const CSSM_ACCESS_CREDENTIALS *AccessCred)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that describes the add-in data storage library module used to perform this function and the data store to which access is being requested. If the form of authentication being requested is authentication to the DL module in general, then the data store handle must be NULL.

`AccessRequest` (*input*)

An indicator of the requested access mode for the data store or DL module in general.

`AccessCred` (*input*)

A pointer to the set of one or more credentials being presented for authentication by the caller. The credentials can apply to the DL module in general or to a particular data store managed by this service module. The credentials required for creating new data stores is defined by the DL and recorded in a record in the MDS Primary DL relation. The required set of credentials to access a particular data store is defined by the `DbInfo` record containing meta-data for the specified data store.

The credentials structure can contain multiple types of credentials, as required for multi-factor authentication. The credential data can be an immediate value, such as a passphrase, PIN, certificate, or template of user-specific data, or the caller can specify a callback function the DL can use to obtain one or more credentials.

DESCRIPTION

This function allows the caller to provide authentication credentials to the DL module at a time other than data store creation, deletion, open, import, and export. AccessRequest defines the type of access to be associated with the caller. If the authentication credential applies to access and use of a DL module in general, then the data store handle specified in the `DLDBHandle` must be NULL. When the authorization credential is to apply to a specific data store, the handle for that data store must be specified in the `DLDBHandle` pair.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_DL_INVALID_ACCESS_REQUEST`
`CSSMERR_DL_INVALID_DB_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

DL_ChangeDbAcl

NAME

DL_ChangeDbAcl: CSSM_DL_ChangeDbAcl – Edit stored ACL (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_ChangeDbAcl
(CSSM_DL_DB_HANDLE DLDBHandle,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_ACL_EDIT *AclEdit)
SPI:
CSSM_RETURN CSSMDLI DL_ChangeDbAcl
(CSSM_DL_DB_HANDLE DLDBHandle,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_ACL_EDIT *AclEdit)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that describes the data storage library module to be used to perform this function, and the open data store whose associated ACL entries are to be updated.

`AccessCred` (*input*)

A pointer to the set of one or more credentials used to authenticate and validate the caller's authorization to modify the ACL associated with the target data base. Required credentials can include zero or more certificates, zero or more caller names, and one or more samples. If certificates and/or caller names are provided as input these must be provided as immediate values in this structure. The samples can be provided as immediate values or can be obtained through a callback function included in the `AccessCred` structure.

`AclEdit` (*input*)

A structure containing information that defines the edit operation. Valid operations include adding, replacing and deleting entries in the set of ACL entries managed by the service provider. The `AclEdit` can contain information for a new ACL entry and a unique handle identifying an existing ACL entry. The information controls the edit operation as follows:

Value of AclEdit.EditMode	Use of AclEdit.NewEntry and AclEdit.OldEntryHandle
CSSM_ACL_EDIT_MODE_ADD	Adds a new ACL entry to the set of ACL entries associated with the specified data base. The new ACL entry is created from the prototype ACL entry contained in `NewEntry`. `OldEntryHandle` is ignored for this `EditMode`.

Value of AclEdit.EditMode	Use of AclEdit.NewEntry and AclEdit.OldEntryHandle
CSSM_ACL_EDIT_MODE_DELETE	Deletes the ACL entry identified by `OldEntryHandle` and associated with the specified data base. `NewEntry` is ignored for this `EditMode`.
CSSM_ACL_EDIT_MODE_REPLACE	Replaces the ACL entry identified by `OldEntryHandle` and associated with the specified data base. The existing ACL is replaced based on the ACL entry prototype contained in `NewEntry`.

When replacing an existing ACL entry, the caller must replace all of the items in an ACL entry. The replacement prototype includes:

Subject type and value

A CSSM_LIST structure containing a typed Subject. The Subject identifies the entity authorized by this ACL entry.

Delegation flag

A CSSM_BOOL value indicating whether the subject can delegate the permissions recorded in the authorization array.

Authorization array

A CSSM_AUTHORIZATIONGROUP structure defining the set of operations for which permission is granted to the Subject.

Validity period

A CSSM_ACL_VALIDITY_PERIOD structure containing two elements, the start time and the stop time for which the ACL entry is valid.

ACL entry tag

A CSSM_STRING containing a user-defined value associated with the ACL entry.

DESCRIPTION

This function edits the stored ACL associated with the target data base identified by `DLDBHandle.DBHandle`. The ACL is modified according to the edit mode and information provided in `AclEdit`.

The caller must be authorized to modify the target ACL. Caller authentication and authorization to edit the ACL is determined based on the caller-provided `AccessCred`.

The caller must be authorized to add, delete or replace the ACL entries associated with the target data base. When adding or replacing an ACL entry, the service provider must reject the creation of duplicate ACL entries.

When adding a new ACL entry to an ACL, the caller must provide a complete ACL entry prototype. All ACL entry items, except the ACL entry `TypedSubject` must be provided as an immediate value in `AclEdit->NewEntry`. The ACL entry Subject can be provided as an immediate value, from a verifier with a protected data path, from an external authentication or authorization service, or through a callback function specified in `AclEdit->NewEntry->Callback`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_DL_INVALID_DB_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_GetDbAcl

Functions for the DL SPI:

DL_GetDbAcl

DL_ChangeDbOwner

NAME

DL_ChangeDbOwner: CSSM_DL_ChangeDbOwner – Define a new data base owner (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_ChangeDbOwner
(CSSM_DL_DB_HANDLE DLDBHandle,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_ACL_OWNER_PROTOTYPE *NewOwner)
SPI:
CSSM_RETURN CSSMDLI DL_ChangeDbOwner
(CSSM_DL_DB_HANDLE DLDBHandle,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_ACL_OWNER_PROTOTYPE *NewOwner)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that describes the data storage library module to be used to perform this function, and the open data store whose associated Owner is to be updated.

`AccessCred` (*input*)

A pointer to the set of one or more credentials used to prove the caller is the current Owner of the Data Base. Required credentials can include zero or more certificates, zero or more caller names, and one or more samples. If certificates and/or caller names are provided as input these must be provided as immediate values in this structure. The samples can be provided as immediate values or can be obtained through a callback function included in the `AccessCred` structure.

`NewOwner` (*input*)

A `CSSM_ACL_OWNER_PROTOTYPE` defining the new Owner of the Data Base.

DESCRIPTION

This function takes a `CSSM_ACL_OWNER_PROTOTYPE` defining the new Owner of the Data Base.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_INVALID_DB_HANDLE
CSSMERR_DL_INVALID_NEW_OWNER
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_GetDbOwner

Functions for the DL SPI:

DL_GetDbOwner

DL_CreateRelation

NAME

DL_CreateRelation: CSSM_DL_CreateRelation – Create a new persistent relation (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_CreateRelation
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_DB_RECORDTYPE RelationID,
const char *RelationName,
uint32 NumberOfAttributes,
const CSSM_DB_SCHEMA_ATTRIBUTE_INFO *pAttributeInfo,
uint32 NumberOfIndexes,
const CSSM_DB_SCHEMA_INDEX_INFO *pIndexInfo)
SPI:
CSSM_RETURN CSSMDLI DL_CreateRelation
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_DB_RECORDTYPE RelationID,
const char *RelationName,
uint32 NumberOfAttributes,
const CSSM_DB_SCHEMA_ATTRIBUTE_INFO *pAttributeInfo,
uint32 NumberOfIndexes,
const CSSM_DB_SCHEMA_INDEX_INFO *pIndexInfo)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that describes the add-in data storage library module to be used to perform this function and the open data store in which to insert the new relation record. The database should be opened in administrative mode using the CSSM_DB_ACCESS_PRIVILEGED flag.

`RelationID` (*input*)

Indicates the type of relation record being added to the data store.

`RelationName` (*input*)

Indicates the name of the relation being added to the data store.

`NumberOfAttributes` (*input*)

Indicates the number of attributes specified in `pAttributeInfo`.

`pAttributeInfo` (*input*)

A list of structures containing the meta information (schema) describing the attributes for the relation being added to the specified data store. The list contains at most one entry per attribute in the specified record type.

`NumberOfIndexes` (*input*)

Indicates the number of indexes specified in `pIndexInfo`.

`pIndexInfo` (*input*)

A list of structures containing the meta information (schema) describing the indexes for the relation being added to the specified data store. The list contains at most one entry per index in the specified record type.

DESCRIPTION

This function creates a new persistent relation of the specified type by inserting it into the specified data store. The `pAttributeInfo` and `pIndexInfo` specify the values contained in the new relation record.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_FIELD_SPECIFIED_MULTIPLE
CSSMERR_DL_INVALID_ATTRIBUTE_INFO
CSSMERR_DL_INVALID_DB_HANDLE
CSSMERR_DL_INVALID_INDEX_INFO
CSSMERR_DL_INVALID_RECORDTYPE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_DestroyRelation

Functions for the DL SPI:

DL_DestroyRelation

DL_DataAbortQuery

NAME

DL_DataAbortQuery: CSSM_DL_DataAbortQuery – Terminate DL_DataGetFirst query (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_DataAbortQuery
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_HANDLE ResultsHandle)
SPI:
CSSM_RETURN CSSMDLI DL_DataAbortQuery
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_HANDLE ResultsHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that describes the add-in data storage library module to be used to perform this function and the open data store from which records were selected by the initiating query.

`ResultsHandle` (*input*)

The selection handle returned from the initial query function.

DESCRIPTION

This function terminates the query initiated by `DL_DataGetFirst()` and allows a DL to release all intermediate state information associated with the query, and release any locks on the resource. The user/application must call `CSSM_DL_DataAbortQuery()` at the termination.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_INVALID_DB_HANDLE
CSSMERR_DL_INVALID_RESULTS_HANDLE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_DataGetFirst, *CSSM_DL_DataGetNext*

Functions for the DL SPI:

DL_DataGetFirst, *dL_DataGetNext*

DL_DataDelete

NAME

DL_DataDelete: CSSM_DL_DataDelete – Remove data record (CDSA)

SYNOPSIS

#include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_DataDelete
(CSSM_DL_DB_HANDLE DLDBHandle,
const CSSM_DB_UNIQUE_RECORD *UniqueRecordIdentifier)
SPI:
CSSM_RETURN CSSMDLI DL_DataDelete
(CSSM_DL_DB_HANDLE DLDBHandle,
const CSSM_DB_UNIQUE_RECORD *UniqueRecordIdentifier)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that describes the add-in data storage library module to be used to perform this function and the open data store from which to delete the specified data record.

`UniqueRecordIdentifier` (*input*)

A pointer to a CSSM_DB_UNIQUE_RECORD identifier containing unique identification of the data record to be deleted from the data store. Once the associated record has been deleted, this unique record identifier cannot be used in future references, except as an argument to `DL_FreeUniqueRecord()` which must still be called.

DESCRIPTION

This function removes the data record specified by the unique record identifier from the specified data store.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_INVALID_DB_HANDLE
CSSMERR_DL_INVALID_RECORD_UID
CSSMERR_DL_RECORD_NOT_FOUND
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_DataInsert

Functions for the DL SPI:

DL_DataInsert

DL_DataGetFirst

NAME

DL_DataGetFirst: CSSM_DL_DataGetFirst – Get first data record (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_DataGetFirst
(CSSM_DL_DB_HANDLE DLDBHandle,
const CSSM_QUERY *Query,
CSSM_HANDLE_PTR ResultsHandle,
CSSM_DB_RECORD_ATTRIBUTE_DATA_PTR Attributes,
CSSM_DATA_PTR Data,
CSSM_DB_UNIQUE_RECORD_PTR *UniqueId)
SPI:
CSSM_RETURN CSSMDLI DL_DataGetFirst
(CSSM_DL_DB_HANDLE DLDBHandle,
const CSSM_QUERY *Query,
CSSM_HANDLE_PTR ResultsHandle,
CSSM_DB_RECORD_ATTRIBUTE_DATA_PTR Attributes,
CSSM_DATA_PTR Data,
CSSM_DB_UNIQUE_RECORD_PTR *UniqueId)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that describes the add-in data storage library module to be used to perform this function and the open data store to search for records satisfying the query.

`Query` (input/optional)

The query structure specifying the selection predicate(s) used to query the data store. The structure contains meta information about the search fields and the relational and conjunctive operators forming the selection predicate. The comparison values to be used in the search are specified in the Attributes field of this Query structure. If a search attribute is of type CSSM_DB_ATTRIBUTE_FORMAT_STRING and the search value specified for that string includes a null-terminator, then the length count for that string should include the terminating character. (If null-terminators are used they should be used consistently, storing the terminator as part of the string in the data store, otherwise selection predicates will not locate expected matches.) The Query structure attributes also identify the particular attributes to be searched by this query. If no query is specified, the DL module can return the first record in the data store, performing sequential retrieval, or return an error. If no selection predicates are specified, the DL module can return the first record in the data store, performing sequential retrieval, or return an error (CSSM_DL_UNSUPPORTED_NUM_SELECTION_PREDS). When selection predicates are

specified, the NumberOfValues of the Attribute of each selection predicate must be 1. If any selection predicate does not satisfy this requirement, the error CSSMERR_DL_INVALID_QUERY is returned.

ResultsHandle (*output*)

This handle should be used to retrieve subsequent records that satisfied this query.

Attributes (optional-input/output)

If the Attributes structure pointer is NULL, no values are returned.

Otherwise, the DataRecordType, NumberOfAttributes and AttributeData fields are read. AttributeData must be an array of NumberOfAttributes CSSM_DB_RECORD_ATTRIBUTE elements. Only the Info field of each element is used on input. The AttributeFormat field of the Info field is ignored on input.

On output, a CSSM_DB_RECORD_ATTRIBUTE structure containing a list of all or the requested attribute values (subset) from the retrieved record. The SemanticInformation field is set. For each CSSM_DB_ATTRIBUTE_DATA contained in the AttributeData array, the NumberOfValues field is set to reflect the size of the Value array which is allocated by the DL using the application specified allocators. Each CSSM_DATA in the Value array will have it's Data field as a pointer to data allocated using the application specified allocators containing the attributes value, and have it's Length set to the length of the value.

All values for an attribute are returned (this could be 0). All fields in the Info field of the CSSM_DB_ATTRIBUTE_DATA are left unchanged except for the AttributeFormat field, which is set to reflect the schema.

Data (optional-input/output)

Data values contained in the referenced memory are ignored during processing and are overwritten with the retrieved opaque object. On output, a CSSM_DATA structure containing the opaque object stored in the retrieved record.

UniqueId (*output*)

If successful and (at least) a record satisfying the query has been found, then this parameter returns a pointer to a CSSM_UNIQUE_RECORD_PTR structure containing a unique identifier associated with the retrieved record. This unique identifier structure can be used in future references to this record using this DLDBHandle pairing. It may not be valid for other DLHandles targeted to this DL module or to other DBHandles targeted to this data store. If there are no records satisfying the query, then this pointer is NULL and CSSM_DL_DataGetFirst() must return CSSM_DL_ENDOFDATA; in this case a normal termination condition has occurred. The CSSM_DL_FreeUniqueRecord() must be used to deallocate this structure.

DESCRIPTION

This function retrieves the first data record in the data store that matches the selection criteria. The selection criteria (including selection predicate and comparison values) is specified in the Query structure. If the Query specifies an attribute that is not defined in the database's meta-information, an error condition is returned. The DL module can use internally-managed indexing structures to enhance the performance of the retrieval operation. This function selects the first record satisfying the query based on the list of Attributes and the opaque Data object. The output buffers for the retrieved record are allocated by this function using the memory management functions provided during the module attach operation. This function also returns a results handle to be used when retrieving subsequent records satisfying the query.

Additional matching records are iteratively retrieved using the `CSSM_DL_DataGetNext()` function . The data storage module supports one of two retrieval models:

- Transactional - all query results are determined at initial query evaluation. Results do not change during an incremental retrieval process.
- File System Scan - query results are selected during the incremental retrieval process. Records matching the query may be added to or deleted from the underlying data store during the iterative retrieval. The caller may receive the new matching records and not received the deleted records.

The caller can determine which retrieval model is supported by examining the encapsulated product description for this data storage module.

If the query selection criteria also specifies time for space limits for executing the query, those limits also apply ro retrieval of the additional selected data records retrieved using the `CSSM_DL_DataGetNext()` function. Finally, this function returns a unique record identifier associated with the retrieved record. This structure can be used in future references to the retrieved data record. Once a user has finished using a certain query, it must call `CSSM_DataAbortQuery()` for releasing resources that CSSM uses. If all records satisfying the query have been retrieved, then query is automatically terminated.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_ENDOFDATA
CSSMERR_DL_FIELD_SPECIFIED_MULTIPLE
CSSMERR_DL_INCOMPATIBLE_FIELD_FORMAT
CSSMERR_DL_INVALID_DB_HANDLE
CSSMERR_DL_INVALID_FIELD_NAME
CSSMERR_DL_INVALID_PARSING_MODULE
CSSMERR_DL_INVALID_QUERY
CSSMERR_DL_INVALID_RECORDTYPE
CSSMERR_DL_INVALID_RECORD_UID
CSSMERR_DL_UNSUPPORTED_FIELD_FORMAT
CSSMERR_DL_UNSUPPORTED_NUM_SELECTION_PREDS
CSSMERR_DL_UNSUPPORTED_OPERATOR
CSSMERR_DL_UNSUPPORTED_QUERY
CSSMERR_DL_UNSUPPORTED_QUERY_LIMITS
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_DataGetNext, *CSSM_DL_DataAbortQuery*

Functions for the DL SPI:

DL_DataGetNext, *DL_DataAbortQuery*

DL_DataGetFromUniqueRecordId

NAME

DL_DataGetFromUniqueRecordId: CSSM_DL_DataGetFromUniqueRecordId – Get data record (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_DataGetFromUniqueRecordId
(CSSM_DL_DB_HANDLE DLDBHandle,
const CSSM_DB_UNIQUE_RECORD_PTR UniqueRecord,
CSSM_DB_RECORD_ATTRIBUTE_DATA_PTR Attributes,
CSSM_DATA_PTR Data)
SPI:
CSSM_RETURN CSSMDLI DL_DataGetFromUniqueRecordId
(CSSM_DL_DB_HANDLE DLDBHandle,
const CSSM_DB_UNIQUE_RECORD_PTR UniqueRecord,
CSSM_DB_RECORD_ATTRIBUTE_DATA_PTR Attributes,
CSSM_DATA_PTR Data)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that describes the add-in data storage library module to be used to perform this function and the open data store to search for the data record.

`UniqueRecord` (*input*)

The pointer to a unique record structure returned from a `DL_DataInsert`, `DL_DataGetFirst`, or `DL_DataGetNext` operation.

`Attributes` (optional-input/output)

If the `Attributes` structure pointer is NULL, no values are returned.

Otherwise, the `DataRecordType`, `NumberOfAttributes` and `AttributeData` fields are read. `AttributeData` must be an array of `NumberOfAttributes` CSSM_DB_RECORD_ATTRIBUTE elements. Only the `Info` field of each element is used on input. The `AttributeFormat` field of the `Info` field is ignored on input.

On output, a CSSM_DB_RECORD_ATTRIBUTE structure containing a list of all or the requested attribute values (subset) from the retrieved record. The `SemanticInformation` field is set. For each CSSM_DB_ATTRIBUTE_DATA contained in the `AttributeData` array, the `NumberOfValues` field is set to reflect the size of the `Value` array which is allocated by the DL using the application specified allocators. Each CSSM_DATA in the `Value` array will have it's `Data` field as a pointer to data allocated using the application specified allocators containing the attributes value, and have it's `Length` set to the length of the value.

All values for an attribute are returned (this could be 0). All fields in the `Info` field of the CSSM_DB_ATTRIBUTE_DATA are left unchanged except for the `AttributeFormat` field, which is set to reflect the schema.

`Data` (optional-input/output)

Data values contained in the referenced memory are ignored during processing and are overwritten with the retrieved opaque object. On output, a CSSM_DATA structure containing the opaque object stored in the retrieved record. If the pointer is data structure pointer is NULL, the opaque object is not returned.

DESCRIPTION

This function retrieves the data record and attributes associated with this unique record identifier. The `Attributes` parameter can specify a subset of the attributes to be returned. If `Attributes` specifies an attribute that is not defined in the database's meta-information, an error condition is returned. The output buffers for the retrieved record are allocated by this function using the memory management functions provided during the module attach operation. The DL module can use an indexing structure identified in the `UniqueRecordId` to enhance the performance of the retrieval operation.

The DL should assume that the value of CSSM_QUERY_FLAGS is when performing this operation. In particular this means that if the data of a key record is being retrieved, the DL will return a CSSM_KEY structure with a key reference.

If the record referenced by `UniqueRecordIdentifier` has been modified since the last time it was retrieved, the error (warning) CSSMERR_DL_RECORD_MODIFIED is returned but the requested attributes and data of the new record is returned. The caller should be advised that other attributes (or the data) might have changed that were not fetched from the DL with this call.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_FIELD_SPECIFIED_MULTIPLE
CSSMERR_DL_INCOMPATIBLE_FIELD_FORMAT
CSSMERR_DL_INVALID_DB_HANDLE
CSSMERR_DL_INVALID_FIELD_NAME
CSSMERR_DL_INVALID_RECORDTYPE
CSSMERR_DL_INVALID_RECORD_UID
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_DataInsert, *CSSM_DL_DataGetFirst*, *CSSM_DL_DataGetNext*

Functions for the DL SPI:

CSSM_DL_DataInsert, *CSSM_DL_DataGetFirst*, *CSSM_DL_DataGetNext*

DL_DataGetNext

NAME

DL_DataGetNext: CSSM_DL_DataGetNext – Get next data record (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_DataGetNext
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_HANDLE ResultsHandle,
CSSM_DB_RECORD_ATTRIBUTE_DATA_PTR Attributes,
CSSM_DATA_PTR Data,
CSSM_DB_UNIQUE_RECORD_PTR *UniqueId)
SPI:
CSSM_RETURN CSSMDLI DL_DataGetNext
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_HANDLE ResultsHandle,
CSSM_DB_RECORD_ATTRIBUTE_DATA_PTR Attributes,
CSSM_DATA_PTR Data,
CSSM_DB_UNIQUE_RECORD_PTR *UniqueId)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that describes the add-in data storage library module to be used to perform this function, and the open data store from which records were selected by the initiating query.

`ResultsHandle` (*input*)

The handle identifying a set of records retrieved by a query executed by the `CSSM_DL_DataGetFirst()` function.

`Attributes` (optional-input/output)

If the `Attributes` structure pointer is NULL, no values are returned.

Otherwise, the `DataRecordType`, `NumberOfAttributes` and `AttributeData` fields are read. `AttributeData` must be an array of `NumberOfAttributes` CSSM_DB_RECORD_ATTRIBUTE elements. Only the `Info` field of each element is used on input. The `AttributeFormat` field of the `Info` field is ignored on input.

On output, a CSSM_DB_RECORD_ATTRIBUTE structure containing a list of all or the requested attribute values (subset) from the retrieved record. The `SemanticInformation` field is set. For each CSSM_DB_ATTRIBUTE_DATA contained in the `AttributeData` array, the `NumberOfValues` field is set to reflect the size of the `Value` array which is allocated by the DL using the application specified allocators. Each CSSM_DATA in the

Value array will have it's Data field as a pointer to data allocated using the application specified allocators containing the attributes value, and have it's Length set to the length of the value.

All values for an attribute are returned (this could be 0). All fields in the Info field of the CSSM_DB_ATTRIBUTE_DATA are left unchanged except for the AttributeFormat field, which is set to reflect the schema.

Data (optional-input/output)

Data values contained in the referenced memory are ignored during processing and are overwritten with the retrieved opaque object. On output, a CSSM_DATA structure containing the opaque object stored in the retrieved record. If the pointer is data structure pointer is NULL, the opaque object is not returned.

UniqueId (*output*)

If successful and (at least) a record satisfying the query has been found, then this parameter returns a pointer to a CSSM_UNIQUE_RECORD_PTR structure containing a unique identifier associated with the retrieved record. This unique identifier structure can be used in future references to this record using this DLDBHandle pairing. It may not be valid for other DLHandles targeted to this DL module or to other DBHandles targeted to this data store. If there are no more records satisfying the query, then this pointer is NULL and CSSM_DL_DataGetNext() must return CSSM_DL_ENDOFDATA; in this case a normal termination condition has occurred. The CSSM_DL_FreeUniqueRecord() must be used to deallocate this structure.

DESCRIPTION

This function returns the next data record referenced by the ResultsHandle. The ResultsHandle references a set of records selected by an invocation of the DataGetFirst function. The Attributes parameter can specify a subset of the attributes to be returned. If Attributes specifies an attribute that is not defined in the database's meta-information, an error condition is returned. The record values are returned in the Attributes and Data parameters. The output buffers for the retrieved record are allocated by this function using the memory management functions provided during the module attach operation. The function also returns a unique record identifier for the return record.

The data storage module supports one of two retrieval models: transactional or file system scan. The transactional model freezes the set of records to be retrieved at query initiation. The file system scan model selects from a potentially changing set of records during the retrieval process. The EndOfDataStore() function indicates when all matching records have been retrieved. The caller can determine which retrieval model is supported by examining the encapsulated product description for this data storage module. Once a user has finished using a certain query, it must call CSSM_DataAbortQuery() for releasing resources that CSSM uses. If all records satisfying the query have been retrieved, then query is automatically terminated.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_ENDOFDATA
CSSMERR_DL_FIELD_SPECIFIED_MULTIPLE
CSSMERR_DL_INCOMPATIBLE_FIELD_FORMAT
CSSMERR_DL_INVALID_DB_HANDLE
CSSMERR_DL_INVALID_FIELD_NAME
CSSMERR_DL_INVALID_RECORDTYPE
CSSMERR_DL_INVALID_RECORD_UID
CSSMERR_DL_INVALID_RESULTS_HANDLE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_DataGetFirst, *CSSM_DL_DataAbortQuery*

Functions for the DL SPI:

DL_DataGetFirst, *DL_DataAbortQuery*

DL_DataInsert

NAME

DL_DataInsert: CSSM_DL_DataInsert – Create new persistent data record (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_DataInsert
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_DB_RECORDTYPE RecordType,
const CSSM_DB_RECORD_ATTRIBUTE_DATA *Attributes,
const CSSM_DATA *Data,
CSSM_DB_UNIQUE_RECORD_PTR *UniqueId)
SPI:
CSSM_RETURN CSSMDLI DL_DataInsert
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_DB_RECORDTYPE RecordType,
const CSSM_DB_RECORD_ATTRIBUTE_DATA *Attributes,
const CSSM_DATA *Data,
CSSM_DB_UNIQUE_RECORD_PTR *UniqueId)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that describes the add-in data storage library module to be used to perform this function and the open data store in which to insert the new data record.

`RecordType` (*input*)

Indicates the type of data record being added to the data store.

`Attributes` (input/optional)

A list of structures containing the attribute values to be stored in that attribute, and the meta information (schema) describing those attributes. The list contains at most one entry per attribute in the specified record type. The specified `AttributeFormat` for each attribute must match that of the database schema, otherwise the error CSSMERR_DL_INCOMPATIBLE_FIELD_FORMAT is returned. If an attribute is of type CSSM_DB_ATTRIBUTE_FORMAT_STRING and the value specified for that string includes a null-terminator, then the length count in the CSSM_DATA structure containing the input string should include the terminating character. (If null-terminators are used, they should be used consistently when storing, searching, and retrieving the string value, otherwise selection predicates will not locate expected matches.) For those attributes that are not assigned values by the caller, the DL module may assume the values to be the empty set, or assume default values, or return an error. If the specified record type does not contain any attributes, this parameter must be NULL.

`Data` (input/optional)

A pointer to the CSSM_DATA structure which contains the opaque data object to be stored in the new data record. If the specified record type does not contain an opaque data object, this parameter must be NULL.

`UniqueId` (*output*)

A pointer to a CSSM_DB_UNIQUE_RECORD_PTR containing a unique identifier associated with the new record. This unique identifier structure can be used in future references to this record during the current open data base session. The pointer will be set to NULL if the function fails. The `CSSM_DL_FreeUniqueRecord()` function must be used to deallocate this structure.

DESCRIPTION

This function creates a new persistent data record of the specified type by inserting it into the specified data store. The values contained in the new data record are specified by the `Attributes` and the `Data`. The attribute value list contains zero or more attribute values. The `Attributes` parameter also specifies a record type. This type must be the same as the type specified by the `RecordType` input parameter. The DL module may require initial values for the CSSM pre-defined attributes. The DL module can assume default values for any unspecified attribute values or can return an error condition when DLM-required attribute values are not specified by the caller. The Data is an opaque object to be stored in the new data record.

If a primary key (concatination of all unique indexes in the relation) exists, the error CSSMERR_DL_INVALID_UNIQUE_INDEX_DATA is returned. The client should call `CSSM_DL_DataGetFirst()`, followed by `CSSM_DL_DataModify()` to change an existing record.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_FIELD_SPECIFIED_MULTIPLE
CSSMERR_DL_INCOMPATIBLE_FIELD_FORMAT
CSSMERR_DL_INVALID_FIELD_NAME
CSSMERR_DL_INVALID_DB_HANDLE
CSSMERR_DL_INVALID_PARSING_MODULE
CSSMERR_DL_INVALID_RECORDTYPE
CSSMERR_DL_INVALID_RECORD_UID
CSSMERR_DL_INVALID_UNIQUE_INDEX_DATA
CSSMERR_DL_INVALID_VALUE
CSSMERR_DL_MISSING_VALUE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_DataDelete

Functions for the DL SPI:

DL_DataDelete

DL_DataModify

NAME

DL_DataModify: CSSM_DL_DataModify – Modify persistent data record (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_DataModify
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_DB_RECORDTYPE RecordType,
CSSM_DB_UNIQUE_RECORD_PTR UniqueRecordIdentifier,
const CSSM_DB_RECORD_ATTRIBUTE_DATA *AttributesToBeModified,
const CSSM_DATA *DataToBeModified,
CSSM_DB_MODIFY_MODE ModifyMode)
SPI:
CSSM_RETURN CSSMDLI DL_DataModify
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_DB_RECORDTYPE RecordType,
CSSM_DB_UNIQUE_RECORD_PTR UniqueRecordIdentifier,
const CSSM_DB_RECORD_ATTRIBUTE_DATA *AttributesToBeModified,
const CSSM_DATA *DataToBeModified,
CSSM_DB_MODIFY_MODE ModifyMode)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that describes the add-in data storage library module to be used to perform this function and the open data store to search for records satisfying the query.

`RecordType` (*input*)

Indicates the type of data record being modified.

`UniqueRecordIdentifier` (input/output)

A pointer to a CSSM_DB_UNIQUE_RECORD containing a unique identifier associated with the record to modify. If the modification succeeds, the `UniqueRecordIdentifier` points to a CSSM_DB_UNIQUE_RECORD containing a unique identifier associated with the updated record. If the modification fails, the `UniqueRecordIdentifier` is not modified.

`AttributesToBeModified` (input/optional)

A list of structures containing the attribute values to be stored in that attribute and the meta information (schema) describing those attributes. The list contains at most one entry per attribute in the specified record type. The specified `AttributeFormat` for each attribute must match that of the database schema, otherwise the error CSSMERR_DL_INCOMPATIBLE_FIELD_FORMAT is returned. If an attribute is of type CSSM_DB_ATTRIBUTE_FORMAT_STRING and the value specified for that string

includes a null-terminator, then the length count in the CSSM_DATA structure containing the input string should include the terminating character. (If null-terminators are used, they should be used consistently when storing, searching, and retrieving the string value, otherwise selection predicates will not locate expected matches.) Each attribute specified is modified according to the value of ModifyMode (see table in the DESCRIPTION section of this definition). Those attributes that are not specified as part of this parameter remain unchanged. If the AttributesToBeModified parameter is NULL, no attribute modification occurs.

DataToBeModified (input/optional)

A pointer to the CSSM_DATA structure which contains the opaque data object to be stored in the data record. If this parameter is NULL, no Data modification occurs.

ModifyMode (*input*)

A CSSM_DB_MODIFY_MODE value indicating the type of modification to be performed on the record attributes identified by AttributesToBeModified. If no attributes are specified, then this value must be CSSM_DB_MODIFY_ATTRIBUTE_NONE.

DESCRIPTION

This function modifies the persistent data record identified by the UniqueRecordIdentifier. The modifications are specified by the Attributes and Data parameters. The ModifyMode indicates how the attributes are to be updated. The ModifyMode has no affect on updating the data blob contained in the record. If the data blob is the only record attribute being updated by this function call, then the modification mode must be 0. The current modification modes behave as follows:

ModifyMode Value	Function Behavior
CSSM_DB_MODIFY_ATTRIBUTE_NONE	No Attributes are being updated.
CSSM_DB_MODIFY_ATTRIBUTE_ADD	The specified values are added to the set of current values for each attribute. If 0 values are specified then the error CSSMERR_DL_INVALID_MODIFY_MODE is returned. If a DL does not support multiple values per attribute, the error CSSMERR_DL_MULTIPLE_VALUES_UNSUPPORTED is returned.
CSSM_DB_MODIFY_ATTRIBUTE_DELETE	The specified values are removed from the set of current values for each attribute. If 0 values are specified then all values are deleted or the attributes value is replaced with the default for this attribute. If a DL does not support multiple values per attribute, the error CSSMERR_DL_MULTIPLE_VALUES_UNSUPPORTED is returned.

CSSM_DB_MODIFY_ATTRIBUTE_REPLACE	The values for each attribute are replaced with the specified set of values for each attribute. If no values are specified then all values are deleted or the attributes value is replaced with the default for this attribute. If a DL does not support multiple values per attribute, the error CSSMERR_DL_MULTIPLE_VALUES_UNSUPPORTED is returned when more than 1 value is specified.

If the attribute lists specifies an attribute that is not defined in the database's meta-information, an error condition is returned. For each attribute-value pair, the value replaces the corresponding attribute value in the record. If a data value is specified, the record's data value is replaced with the specified value. A record's data value or attribute values can be set to NULL or zero to represent deletion or the lack of a known value.

If the record referenced by `UniqueRecordIdentifier` has been modified since the last time it was updated, the error CSSMERR_DL_STALE_UNIQUE_RECORD is returned and no modification takes place.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_FIELD_SPECIFIED_MULTIPLE
CSSMERR_DL_INCOMPATIBLE_FIELD_FORMAT
CSSMERR_DL_INVALID_DB_HANDLE
CSSMERR_DL_INVALID_FIELD_NAME
CSSMERR_DL_INVALID_MODIFY_MODE
CSSMERR_DL_INVALID_RECORDTYPE
CSSMERR_DL_INVALID_RECORD_UID
CSSMERR_DL_INVALID_UNIQUE_INDEX_DATA
CSSMERR_DL_INVALID_VALUE
CSSMERR_DL_MULTIPLE_VALUES_UNSUPPORTED
CSSMERR_DL_STALE_UNIQUE_RECORD
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_DataInsert, *CSSM_DL_DataDelete*

Functions for the DL SPI:

DL_DataInsert, *DL_DataDelete*

DL_DbClose

NAME

DL_DbClose: CSSM_DL_DbClose – Close open data store (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_DbClose
(CSSM_DL_DB_HANDLE DLDBHandle)
SPI:
CSSM_RETURN CSSMDLI DL_DbClose
(CSSM_DL_DB_HANDLE DLDBHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

A handle structure containing the DL handle for the attached DL module and the DB handle for an open data store managed by the DL. This specifies the open data store to be closed.

DESCRIPTION

This function closes an open data store.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_DL_INVALID_DB_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_DbOpen

Functions for the DL SPI:

DL_DbOpen

DL_DbCreate

NAME

DL_DbCreate: CSSM_DL_DbCreate – Create and open new data store (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_DbCreate
(CSSM_DL_HANDLE DLHandle,
const char *DbName,
const CSSM_NET_ADDRESS *DbLocation,
const CSSM_DBINFO *DBInfo,
CSSM_DB_ACCESS_TYPE AccessRequest,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry,
const void *OpenParameters,
CSSM_DB_HANDLE *DbHandle)
SPI:
CSSM_RETURN CSSMDLI DL_DbCreate
(CSSM_DL_HANDLE DLHandle,
const char *DbName,
const CSSM_NET_ADDRESS *DbLocation,
const CSSM_DBINFO *DBInfo,
CSSM_DB_ACCESS_TYPE AccessRequest,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry,
const void *OpenParameters,
CSSM_DB_HANDLE *DbHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLHandle` (*input*)

The handle that describes the add-in data storage library module used to perform this function.

`DbName` (*input*)

The logical name for the new data store.

`DbLocation` (input/optional)

A pointer to a network address directly or indirectly identifying the location of the storage service process. If the input is NULL, the module can assume a default storage service process location. If the `DbName` does not distinguish the storage service process, the service cannot be performed and the operation fails.

`DBInfo` (*input*)

A pointer to a structure describing the format/schema of each record type that will be stored in the new data store.

`AccessRequest` (*input*)

An indicator of the requested access mode for the data store, such as read-only or read-write.

`CredAndAclEntry` (input/optional)

A structure containing one or more credentials authorized for creating a data base and the prototype ACL entry that will control future use of the newly created key. The credentials and ACL entry prototype can be presented as immediate values or callback functions can be provided for use by the DL to acquire the credentials and/or the ACL entry interactively. If the DL provides public access for creating a data base, then the credentials can be NULL. If the DL defines a default initial ACL entry for the new data base, then the ACL entry prototype can be an empty list.

`OpenParameters` (input/optional)

A pointer to a module-specific set of parameters required to open the data store.

`DbHandle` (*output*)

The handle to the newly created and open data store. The value will be set to CSSM_INVALID_HANDLE if the function fails.

DESCRIPTION

This function creates and opens a new data store. The name of the new data store is specified by the input parameter `DbName`. The record schema for the data store is specified in the DBINFO structure. If any `RecordType` defined in the DBINFO structure does not have an associated parsing module, then the `ModuleSubserviceUid` specified for that record type must be zero.

The newly created data store is opened under the specified access mode. If user authentication credentials are required, they must be provided. Also, additional open parameters may be required and are supplied in `OpenParameters`. If user authentication credentials are required, they must be provided.

Authorization policy can restrict the set of callers who can create a new resource. In this case, the caller must present a set of access credentials for authorization. Upon successfully authenticating the credentials, the template that verified the presented samples identifies the ACL entry that will be used in the authorization computation. If the caller is authorized, the new resource is created.

The caller must provide an initial ACL entry to be associated with the newly created resource. This entry is used to control future access to the new resource and (since the subject is deemed to be the "Owner") exercise control over its associated ACL. The caller can specify the following items for initializing an ACL entry:

Subject

A CSSM_LIST structure, containing the type of the subject and a template value that can be used to verify samples that are presented in credentials when resource access is requested.

Delegation flag

A value indicating whether the Subject can delegate the permissions recorded in the `AuthorizationTag`. (This item only applies to public key subjects).

Authorization tag

The set of permissions that are granted to the Subject.

Validity period

The start time and the stop time for which the ACL entry is valid.

ACL entry tag

A user-defined string value associated with the ACL entry.

The service provider can modify the caller-provided initial ACL entry to conform to any innate resource-access policy that the service provider may be required to enforce. If the initial ACL entry provided by the caller contains values or permissions that are not supported by the service provider, then the service provider can modify the initial ACL appropriately or can fail the request to create the new resource. Service providers list their supported `AuthorizationTag` values in their Module Directory Services primary record.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_DATASTORE_ALREADY_EXISTS
CSSMERR_DL_FIELD_SPECIFIED_MULTIPLE
CSSMERR_DL_INCOMPATIBLE_FIELD_FORMAT
CSSMERR_DL_INVALID_ACCESS_REQUEST
CSSMERR_DL_INVALID_DB_LOCATION
CSSMERR_DL_INVALID_DB_NAME
CSSMERR_DL_INVALID_FIELD_NAME
CSSMERR_DL_INVALID_OPEN_PARAMETERS
CSSMERR_DL_INVALID_PARSING_MODULE
CSSMERR_DL_INVALID_RECORDTYPE
CSSMERR_DL_INVALID_RECORD_INDEX
CSSMERR_DL_UNSUPPORTED_FIELD_FORMAT
CSSMERR_DL_UNSUPPORTED_INDEX_INFO
CSSMERR_DL_UNSUPPORTED_LOCALITY
CSSMERR_DL_UNSUPPORTED_NUM_ATTRIBUTES
CSSMERR_DL_UNSUPPORTED_NUM_INDEXES
CSSMERR_DL_UNSUPPORTED_NUM_RECORDTYPES
CSSMERR_DL_UNSUPPORTED_RECORDTYPE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_DbOpen, *CSSM_DL_DbClose*, *CSSM_DL_DbDelete*

Functions for the DL SPI:

DL_DbOpen, *DL_DbClose*, *DL_DbDelete*

DL_DbDelete

NAME

DL_DbDelete: CSSM_DL_DbDelete – Delete all records (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_DbDelete
(CSSM_DL_HANDLE DLHandle,
const char *DbName,
const CSSM_NET_ADDRESS *DbLocation,
const CSSM_ACCESS_CREDENTIALS *AccessCred)
SPI:
CSSM_RETURN CSSMDLI DL_DbDelete
(CSSM_DL_HANDLE DLHandle,
const char *DbName,
const CSSM_NET_ADDRESS *DbLocation,
const CSSM_ACCESS_CREDENTIALS *AccessCred)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLHandle` (*input*)

The handle that describes the add-in data storage library module to be used to perform this function.

`DbName` (*input*)

A pointer to the string containing the logical name of the data store.

`DbLocation` (input/optional)

A pointer to a network address directly or indirectly identifying the location of the storage service process. If the input is NULL, the module can assume a default storage service process location. If the `DbName` does not distinguish the storage service process, the service cannot be performed and the operation fails.

`AccessCred` (input/optional)

A pointer to the set of one or more credentials being presented for authentication by the caller. These credentials are required to obtain access to the specified data store. The credentials structure can contain multiple types of credentials, as required for multi-factor authentication. The credential data can be an immediate value, such as a passphrase, PIN, certificate, or template of user-specific data, or the caller can specify a callback function the DL can use to obtain one or more credentials. The required set of credentials to access a particular data store is defined by the `DbInfo` record containing meta-data for the specified data store. If credentials are not required to access the specified data store, then this field can be NULL.

DESCRIPTION

This function deletes all records from the specified data store and removes all state information associated with that data store.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_DATASTORE_DOESNOT_EXIST
CSSMERR_DL_DATASTORE_IS_OPEN
CSSMERR_DL_INVALID_DB_LOCATION
CSSMERR_DL_INVALID_DB_NAME
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_DbCreate, *CSSM_DL_DbOpen*, *CSSM_DL_DbClose*

Functions for the DL SPI:

DL_DbCreate, *DL_DbOpen*, *DL_DbClose*

DL_DbOpen

NAME

DL_DbOpen: CSSM_DL_DbOpen – Open a data store (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_DbOpen
(CSSM_DL_HANDLE DLHandle,
const char *DbName,
const CSSM_NET_ADDRESS *DbLocation,
CSSM_DB_ACCESS_TYPE AccessRequest,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const void *OpenParameters,
CSSM_DB_HANDLE *DbHandle)
SPI:
CSSM_RETURN CSSMDLI DL_DbOpen
(CSSM_DL_HANDLE DLHandle,
const char *DbName,
const CSSM_NET_ADDRESS *DbLocation,
CSSM_DB_ACCESS_TYPE AccessRequest,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const void *OpenParameters,
CSSM_DB_HANDLE *DbHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLHandle` (*input*)

The handle that describes the add-in data storage library module to be used to perform this function.

`DbName` (*input*)

A pointer to the string containing the logical name of the data store.

`DbLocation` (input/optional)

A pointer to a network address directly or indirectly identifying the location of the storage service process. If the input is NULL, the module can determine a storage service process and its location based on the DbName (for existing data stores) or can assume a default storage service process location. If the DbName does not distinguish the storage service process, the service cannot be performed and the operation fails.

`AccessRequest` (*input*)

An indicator of the requested access mode for the data store, such as read-only or read-write.

`AccessCred` (input/optional)

A pointer to the set of one or more credentials being presented for authentication by the caller. These credentials are required to obtain access to the specified data store. The credentials structure can contain multiple types of credentials, as required for multi-factor authentication. The credential data can be an immediate value, such as a passphrase, PIN, certificate, or template of user-specific data, or the caller can specify a callback function the DL can use to obtain one or more credentials. The required set of credentials to access a particular data store is defined by the `DbInfo` record containing meta-data for the specified data store. If credentials are not required to access the specified data store, then this field can be NULL.

`OpenParameters` (input/optional)

A pointer to a module-specific set of parameters required to open the data store.

`DbHandle` (*output*)

The handle to the opened data store. The value will be set to CSSM_INVALID_HANDLE if the function fails.

DESCRIPTION

This function opens the data store with the specified logical name under the specified access mode. If user authentication credentials are required, they must be provided. Also, additional open parameters may be required to open a given data store, and are supplied in the `OpenParameters`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_DB_LOCKED
CSSMERR_DL_INVALID_ACCESS_REQUEST
CSSMERR_DL_INVALID_DB_LOCATION
CSSMERR_DL_INVALID_DB_NAME
CSSMERR_DL_DATASTORE_DOESNOT_EXIST
CSSMERR_DL_INVALID_PARSING_MODULE
CSSMERR_DL_INVALID_OPEN_PARAMETERS
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_DbClose

Functions for the DL SPI:

DL_DestroyRelation

NAME

DL_DestroyRelation: CSSM_DL_DestroyRelation – Destroy an existing relation (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_DestroyRelation
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_DB_RECORDTYPE RelationID)
SPI:
CSSM_RETURN CSSMDLI DL_DestroyRelation
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_DB_RECORDTYPE RelationID)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that describes the add-in data storage library module to be used to perform this function and the open data store from which to delete the relation record.

`RelationID` (*input*)

Indicates the type of relation record being deleted from the data store.

DESCRIPTION

This function destroys an existing relation of the specified type by removing its entry from the specified data store.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_INVALID_DB_HANDLE
CSSMERR_DL_INVALID_RECORDTYPE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_CreateRelation

Functions for the DL SPI:

DL_CreateRelation

DL_FreeNameList

NAME

DL_FreeNameList: CSSM_DL_FreeNameList – Free the list of the logical data store names (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_FreeNameList
(CSSM_DL_HANDLE DLHandle,
CSSM_NAME_LIST_PTR NameList)
SPI:
CSSM_RETURN CSSMDLI DL_FreeNameList
(CSSM_DL_HANDLE DLHandle,
CSSM_NAME_LIST_PTR NameList)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLHandle` (*input*)

The handle that describes the add-in data storage library module to be used to perform this function.

`NameList` (*input*)

A pointer to the CSSM_NAME_LIST.

DESCRIPTION

This function frees the list of the logical data store names that was returned by `CSSM_DL_GetDbNames`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_GetDbNames

Functions for the DL SPI:

DL_GetDbNames

DL_FreeUniqueRecord

NAME

DL_FreeUniqueRecord: CSSM_DL_FreeUniqueRecord – Free data store memory (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_FreeUniqueRecord
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_DB_UNIQUE_RECORD_PTR UniqueRecord)
SPI:
CSSM_RETURN CSSMDLI DL_FreeUniqueRecord
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_DB_UNIQUE_RECORD_PTR UniqueRecord)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that describes the add-in data storage library module to be used to perform this function and the open data store from which the UniqueRecord identifier was assigned.

`UniqueRecord`(*input*)

The pointer to the memory that describes the data store unique record structure.

DESCRIPTION

This function frees the memory associated with the data store unique record structure.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_INVALID_DB_HANDLE
CSSMERR_DL_INVALID_RECORD_UID
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_DataInsert, *CSSM_DL_DataGetFirst*, *CSSM_DL_DataGetNext*

Functions for the DL SPI:

DL_DataInsert, *DL_DataGetFirst*, *DL_DataGetNext*

DL_GetDbAcl

NAME

DL_GetDbAcl: CSSM_DL_GetDbAcl – Get ACL description (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_GetDbAcl
(CSSM_DL_DB_HANDLE DLDBHandle,
const CSSM_STRING *SelectionTag,
uint32 *NumberOfAclInfos,
CSSM_ACL_ENTRY_INFO_PTR *AclInfos)
SPI:
CSSM_RETURN CSSMDLI DL_GetDbAcl
(CSSM_DL_DB_HANDLE DLDBHandle,
const CSSM_STRING *SelectionTag,
uint32 *NumberOfAclInfos,
CSSM_ACL_ENTRY_INFO_PTR *AclInfos)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that identifies the Data Storage service provider to perform this operation and the target data store whose associated ACL entries are scanned and returned.

`SelectionTag` (input/optional)

A CSSM_STRING value matching the user-defined tag value associated with one or more ACL entries for the target data base. To retrieve a description of all ACL entries for the target data base, this parameter must be NULL.

`NumberOfAclInfos` (*output*)

The number of entries in the `AclInfos` array. If no ACL entry descriptions are returned, this value is zero.

`AclInfos` (*output*)

An array of CSSM_ACL_ENTRY_INFO structures. The unique handle contained in each structure can be used during the current attach session to reference the ACL entry for editing. The structure is allocated by the service provider and must be released by the caller when the structure is no longer needed. If no ACL entry descriptions are returned, this value is NULL.

DESCRIPTION

This function returns a description of zero or more ACL entries managed by the data storage service provider module and associated with the target database identified by `DLDBHandle.DBHandle`. The optional input `SelectionTag` restricts the returned descriptions to those ACL entries with a matching `EntryTag` value. If a `SelectionTag` value is specified and no matches are found, zero descriptions are returned. If no `SelectionTag` is specified, a description of all ACL entries associated with the target data base are returned by this function.

Each `AclInfo` structure contains:

- Public contents of an ACL entry
- ACL `EntryHandle`, which is a unique value defined and managed by the service provider

The public ACL entry information returned by this function includes:

The subject type

A CSSM_LIST structure containing one element identifying the type of subject stored in the ACL entry.

Delegation flag

A CSSM_BOOL value indicating whether the subject can delegate the permissions recorded in Authorization.

Authorization array

A CSSM_AUTHORIZATIONGROUP structure defining the set of operations for which permission is granted to the Subject.

Validity period

A CSSM_ACL_VALIDITY_PERIOD structure containing two elements, the start time and the stop time for which the ACL entry is valid.

ACL entry tag

A CSSM_STRING containing a user-defined value associated with the ACL entry.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_DL_INVALID_DB_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_ChangeDbAcl

Functions for the DL SPI:

DL_ChangeDbAcl

DL_GetDbNameFromHandle

NAME

DL_GetDbNameFromHandle: CSSM_DL_GetDbNameFromHandle – Get data source name (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_GetDbNameFromHandle
(CSSM_DL_DB_HANDLE DLDBHandle,
char **DbName)
SPI:
CSSM_RETURN CSSMDLI DL_GetDbNameFromHandle
(CSSM_DL_DB_HANDLE DLDBHandle,
char **DbName)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that identifies the add-in data storage library module and the open data store whose name should be retrieved.

`DbName` (*output*)

Returns a zero terminated string which contains a data store name. The memory is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function retrieves the data source name corresponding to an opened data store handle.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_DL_INVALID_DB_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_GetDbNames

Functions for the DL SPI:

DL_GetDbNames

DL_GetDbNames

NAME

DL_GetDbNames: CSSM_DL_GetDbNames – Get list of logical data store names (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_GetDbNames
(CSSM_DL_HANDLE DLHandle,
CSSM_NAME_LIST_PTR *NameList)
SPI:
CSSM_RETURN CSSMDLI DL_GetDbNames
(CSSM_DL_HANDLE DLHandle,
CSSM_NAME_LIST_PTR *NameList)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLHandle` (*input*)

The handle that describes the add-in data storage library module to be used to perform this function.

`NameList` (*output*)

Returns a list of data store names in a CSSM_NAME_LIST_PTR structure.

DESCRIPTION

This function returns the list of logical data store names for all data stores that are known by and accessible to the specified DL module. This list also includes the number of data store names in the return list.

The `CSSM_DL_FreeNameList()` function must be called to deallocate memory containing the list.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_GetDbNameFromHandle, *CSSM_DL_FreeNameList*

Functions for the DL SPI:

DL_GetDbNameFromHandle, *DL_FreeNameList*

DL_GetDbOwner

NAME

DL_GetDbOwner: CSSM_DL_GetDbOwner – Get data base owner (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_GetDbOwner
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_ACL_OWNER_PROTOTYPE_PTR Owner)
SPI:
CSSM_RETURN CSSMDLI DL_GetDbOwner
(CSSM_DL_DB_HANDLE DLDBHandle,
CSSM_ACL_OWNER_PROTOTYPE_PTR Owner)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETER

`DLDBHandle` (*input*)

The handle pair that describes the data storage library module to be used to perform this function, and the open data store whose associated Owner is to be retrieved.

`Owner` (*output*)

A CSSM_ACL_OWNER_PROTOTYPE describing the current Owner of the Data Base.

DESCRIPTION

This function returns a CSSM_ACL_OWNER_PROTOTYPE describing the current Owner of the Data Base.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_DL_INVALID_DB_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_DL_ChangeDbOwner

Functions for the DL SPI:

DL_ChangeDbOwner

DL_PassThrough

NAME

DL_PassThrough: CSSM_DL_PassThrough – Extend data storage module functionality (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_DL_PassThrough
(CSSM_DL_DB_HANDLE DLDBHandle,
uint32 PassThroughId,
const void *InputParams,
void **OutputParam)
SPI:
CSSM_RETURN CSSMDLI DL_PassThrough
(CSSM_DL_DB_HANDLE DLDBHandle,
uint32 PassThroughId,
const void *InputParams,
void **OutputParam)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`DLDBHandle` (*input*)

The handle pair that describes the add-in data storage library module to be used to perform this function and the open data store upon which the function is to be performed.

`PassThroughId` (*input*)

An identifier assigned by a DL module to indicate the exported function to be performed.

`InputParams` (*input*)

A pointer to a module implementation-specific structure containing parameters to be interpreted in a function-specific manner by the requested DL module.

`OutputParams` (*output*)

A pointer to a module, implementation-specific structure containing the output data. The service provider will allocate the memory for this structure. The application should free the memory for the structure.

DESCRIPTION

This function allows applications to call data storage library module-specific operations that have been exported. Such operations may include queries or services that are specific to the domain represented by a DL module.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_DL_INVALID_DB_HANDLE`
`CSSMERR_DL_INVALID_PASSTHROUGH_ID`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

EncryptData

NAME

EncryptData: CSSM_EncryptData, CSP_EncryptData – Encrypts all buffer data (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_EncryptData
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *ClearBufs,
uint32 ClearBufCount,
CSSM_DATA_PTR CipherBufs,
uint32 CipherBufCount,
uint32 *bytesEncrypted,
CSSM_DATA_PTR RemData)
SPI:
CSSM_RETURN CSSMCSPI CSP_EncryptData
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
const CSSM_DATA *ClearBufs,
uint32 ClearBufCount,
CSSM_DATA_PTR CipherBufs,
uint32 CipherBufCount,
uint32 *bytesEncrypted,
CSSM_DATA_PTR RemData,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`ClearBufs` (*input*)

A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

`ClearBufCount` (*input*)

The number of `ClearBufs`.

`CipherBufs` (*output*)

A pointer to a vector of CSSM_DATA structures that contain the results of the operation on the data.

`CipherBufCount` (*input*)

The number of `CipherBufs`.

`bytesEncrypted` (*output*)

A pointer to `uint32` for the size of the encrypted data in bytes.

`RemData` (*output*)

A pointer to the CSSM_DATA structure for the remaining cipher text if there is not enough buffer space available in the output data structures.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

`Privilege` (*input*)

The export privilege to be applied during the cryptographic operation. This parameter is forwarded to the CSP after CSSM verifies the caller and service provider privilege set includes the specified PRIVILEGE.

DESCRIPTION

This function encrypts all data contained in the set of input buffers using information in the context. The `CSSM_QuerySize()` function can be used to estimate the output buffer size required. The minimum number of buffers required to contain the resulting cipher text is produced as output. If the cipher text result does not fit within the set of output buffers, the remaining cipher text is returned in the single output buffer `RemData`.

The CSP can require that the cryptographic context include access credentials for authentication and authorization checks when using a private key or a secret key.

NOTES FOR API

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, preallocated output buffer, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value greater than zero and a non-NULL data pointer field value. To specify automatic output buffer allocation by the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL Data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed. In-place encryption can be done by supplying the same input and output buffers.

NOTES FOR SPI

The output is returned to the caller as specified in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_BLOCK_SIZE_MISMATCH`
`CSSMERR_CSP_OUTPUT_LENGTH_ERROR`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_QuerySize, *CSSM_DecryptData*, *CSSM_EncryptDataInit*, *CSSM_EncryptDataUpdate*, *CSSM_EncryptDataFinal*, *CSSM_EncryptDataP*, *CSSM_EncryptDataInitP*, *CSSM_DecryptP*, *CSSM_DecryptDataInitP*

Functions for the CSP SPI:

CSP_QuerySize, *CSP_DecryptData*, *CSP_EncryptDataInit*, *CSP_EncryptDataUpdate*, *CSP_EncryptDataFinal*

EncryptDataFinal

NAME

EncryptDataFinal: CSSM_EncryptDataFinal, CSP_EncryptDataFinal – Finalize staged encryption process (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_EncryptDataFinal
(CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR RemData)
SPI:
CSSM_RETURN CSSMCSPI CSP_EncryptDataFinal
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR RemData)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`RemData` (*output*)

A pointer to the CSSM_DATA structure for the last encrypted block containing padded data, if necessary.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function finalizes the staged encryption process by returning any remaining cipher text not returned in the previous staged encryption call. The cipher text is returned in a single buffer.

NOTES FOR API

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, preallocated output buffer, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value greater than zero and a non-NULL data pointer field value. To specify automatic output buffer allocation by

the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed.

NOTES FOR SPI

The output is returned to the caller as specified in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_BLOCK_SIZE_MISMATCH`
`CSSMERR_CSP_OUTPUT_LENGTH_ERROR`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_QuerySize, *CSSM_DecryptData*, *CSSM_EncryptDataInit*, *CSSM_EncryptDataUpdate*, *CSSM_EncryptDataFinal*, *CSSM_EncryptDataP*, *CSSM_EncryptDataInitP*, *CSSM_DecryptP*, *CSSM_DecryptDataInitP*

Functions for the CSP SPI:

CSP_EncryptData, *CSP_EncryptDataInit*, *CSP_EncryptDataUpdate*

EncryptDataInit

NAME

EncryptDataInit: CSSM_EncryptDataInit, CSP_EncryptDataInit – Initialize the staged encrypt funciton (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_EncryptDataInit
(CSSM_CC_HANDLE CCHandle)
SPI:
CSSM_RETURN CSSMCSPI CSP_EncryptDataInit
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

`Privilege` (*input*)

The export privilege to be applied during the cryptographic operation. This parameter is forwarded to the CSP after CSSM verifies the caller and service provider privilege set includes the specified PRIVILEGE.

DESCRIPTION

This function initializes the staged encrypt function. There may be algorithm-specific and token-specific rules restricting the lengths of data following data update calls making use of these parameters.

The CSP can require that the cryptographic context include access credentials for authentication and authorization checks when using a private key or a secret key.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_QuerySize, *CSSM_DecryptData*, *CSSM_EncryptDataInit*, *CSSM_EncryptDataUpdate*, *CSSM_EncryptDataFinal*, *CSSM_EncryptDataP*, *CSSM_EncryptDataInitP*, *CSSM_DecryptP*, *CSSM_DecryptDataInitP*

Functions for the CSP SPI:

CSP_QuerySize, *CSP_DecryptData*, *CSP_EncryptDataInit*, *CSP_EncryptDataUpdate*, *CSP_EncryptDataFinal*

EncryptDataInitP

NAME

EncryptDataInitP – Initialize the staged encrypt function with privilege (CDSA)

SYNOPSIS

```
# include <cssm.h>

CSSM_RETURN CSSMAPI CSSM_EncryptDataInitP
(CSSM_CC_HANDLE CCHandle,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`Privilege` (*input*)

The privilege to be applied during the cryptographic operation.

See *CSSM_EncryptDataInit* for other parameters.

DESCRIPTION

This function is similar to `CSSM_EncryptDataInit()`. It also accepts a USEE tag as a privilege request parameter. CSSM checks that either its privilege set or the application's privilege set (if the application is signed) includes the tag. If the tag is found and the service provider privilege set indicates that it is supported, the tag is forwarded to the service provider.

For staged operations using privilege initialization functions `CSSM_EncryptDataInitP()`, the completion functions `CSSM_EncryptDataUpdate()` and `CSSM_EncryptDataFinalize()` are used.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_DecryptData*, *CSSM_EncryptDataInit*, *CSSM_EncryptDataUpdate*, *CSSM_EncryptDataFinal*, *CSSM_EncryptDataP*, *CSSM_EncryptDataInitP*, *CSSM_DecryptP*, *CSSM_DecryptDataInitP*, *CSSM_QuerySize*

EncryptDataP

NAME

EncryptDataP – Encrypt data with privilege (CDSA)

SYNOPSIS

```
# include <cssm.h>

CSSM_RETURN CSSMAPI CSSM_EncryptDataP
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *ClearBufs,
uint32 ClearBufCount,
CSSM_DATA_PTR CipherBufs,
uint32 CipherBufCount,
uint32 *bytesEncrypted,
CSSM_DATA_PTR RemData,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`Privilege` (*input*)

The privilege to be applied during the cryptographic operation.

See *CSSM_EncryptData* for other parameters.

DESCRIPTION

This function is similar to `CSSM_EncryptData()`. It also accepts a USEE tag as a privilege request parameter. CSSM checks that either its own privilege set or the application's privilege set (if the application is signed) includes the tag. If the tag is found and the service provider privilege set indicates that it is supported, the tag is forwarded to the service provider.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSP_BLOCK_SIZE_MISMATCH
CSSMERR_CSP_OUTPUT_LENGTH_ERROR
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_DecryptData*, *CSSM_EncryptDataInit*, *CSSM_EncryptDataUpdate*, *CSSM_EncryptDataFinal*, *CSSM_EncryptDataP*, *CSSM_EncryptDataInitP*, *CSSM_DecryptP*, *CSSM_DecryptDataInitP*, *CSSM_QuerySize*

EncryptDataUpdate

NAME

EncryptDataUpdate: CSSM_EncryptDataUpdate, CSP_EncryptDataUpdate – Continue the staged encryption process (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_EncryptDataUpdate
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *ClearBufs,
uint32 ClearBufCount,
CSSM_DATA_PTR CipherBufs,
uint32 CipherBufCount,
uint32 *bytesEncrypted)
SPI:
CSSM_RETURN CSSMCSPI CSP_EncryptDataUpdate
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *ClearBufs,
uint32 ClearBufCount,
CSSM_DATA_PTR CipherBufs,
uint32 CipherBufCount,
uint32 *bytesEncrypted)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`ClearBufs` (*input*)

A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

`ClearBufCount` (*input*)

The number of `ClearBufs`.

`CipherBufs` (*output*)

A pointer to a vector of CSSM_DATA structures that contain the encrypted data resulting from the encryption operation.

`CipherBufCount` (*input*)

The number of `CipherBufs`.

`bytesEncrypted` (*output*)

A pointer to `uint32` for the size of the encrypted data in bytes.

SPI PARAMETERS

`CSPHandle (`*`input`*`)`

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function continues the staged encryption process over all data in the set of input buffers. There can be algorithm-specific and token-specific rules restricting the lengths of data in `CSSM_EncryptUpdate()` calls, but multiple input buffers are supported. The minimum number of buffers required to contain the resulting cipher text is produced as output. Excess output buffer space is not remembered across staged encryption calls. Each staged call begins filling one or more new output buffers. The `CSSM_QuerySize()` function can be used to estimate the output buffer size required for each update call.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

NOTES FOR API

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, preallocated output buffer, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value greater than zero and a non-NULL Data pointer field value. To specify automatic output buffer allocation by the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed. In-place encryption can be done by supplying the same input and output buffers.

NOTES FOR SPI

The output is returned to the caller as specified in Buffer Management for Cryptographic Services.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_QuerySize, *CSSM_DecryptData*, *CSSM_EncryptDataInit*, *CSSM_EncryptDataUpdate*, *CSSM_EncryptDataFinal*, *CSSM_EncryptDataP*, *CSSM_EncryptDataInitP*, *CSSM_DecryptP*, *CSSM_DecryptDataInitP*

Functions for the CSP SPI:

CSP_QuerySize, *CSP_DecryptData*, *CSP_EncryptDataInit*, *CSP_EncryptDataFinal*

FreeKey

NAME

FreeKey: CSSM_FreeKey, CSP_FreeKey – Clean up keys (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_FreeKey
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
CSSM_KEY_PTR KeyPtr,
CSSM_BOOL Delete)
SPI:
CSSM_RETURN CSSMCSPI CSP_FreeKey
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
CSSM_KEY_PTR KeyPtr,
CSSM_BOOL Delete)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the module to perform this operation.

`AccessCred` (input/optional)

If the target key referenced by `KeyPtr` is protected and `Delete` has the value CSSM_TRUE, this parameter must contain the certificates and samples required to access the target key. The certificates must be presented as immediate values in the input structure. The samples can be immediate values, be obtained through a protected mechanism, or be obtained through a `callback` function.

`KeyPtr` (*input*)

The key whose associated keying material can be discarded at this time.

`Delete` (*input*)

If this value is CSSM_TRUE, the key data in the key structure will be removed and any internal storage related to that key will also be removed. In this case the key no longer exists in any form, unless previously wrapped out of the CSP by the application. If this value is CSSM_FALSE, then only the resources related to the key structure are released. The key may still be accessible by other means internally to the CSP.

DESCRIPTION

This function requests the Cryptographic Service Provider to clean up any key material associated with the key, and to possibly delete the key from the CSP completely. This function also releases the internal storage referenced by the KeyData field of the key structure, which can hold the actual key value. The key reference by `KeyPtr` can be a persistent key or a transient key. This function clears the cached copy of the key and can have an effect on the long term persistence or transience of the key.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

GenerateAlgorithmParams

NAME

GenerateAlgorithmParams: CSSM_GenerateAlgorithmParams, CSP_GenerateAlgorithmParams – Generate algorithm parameters (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_GenerateAlgorithmParams
(CSSM_CC_HANDLE CCHandle,
uint32 ParamBits,
CSSM_DATA_PTR Param)
SPI:
CSSM_RETURN CSSMCSPI CSP_GenerateAlgorithmParams
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
uint32 ParamBits,
CSSM_DATA_PTR Param,
uint32 *NumberOfUpdatedAttributes,
CSSM_CONTEXT_ATTRIBUTE_PTR *UpdatedAttributes)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`ParamBits` (*input*)

Used to generate parameters for the algorithm (for example, Diffie-Hellman).

`Param` (*output*)

Pointer to a CSSM_DATA structure used to provide information to the parameter generation process, or to receive information resulting from the generation process that is not required as a parameter to the algorithm. For instance, phase 2 of the KEA algorithm requires a private random value, rA, and a public version, Ra, to be generated. The private value, rA, is added to the context and the public value, Ra, is returned to the caller. In some cases, when both input and output is required, a data structure is passed to the algorithm. In this situation, `Param->Data` references the structure and `Param->Length` is set to the length of the structure.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context. Modifying this structure has no effect on the internal structure maintained by the CSSM. It is only a copy of the actual data. Changes to the context attributes must be returned using the `UpdatedAttributes` return parameter.

`NumberOfUpdatedAttributes` (*output*)

The number of CSSM_CONTEXT_ATTRIBUTE structures contained in the `UpdatedAttributes` array. If this value is zero, `UpdatedAttributes` should be set to NULL.

`UpdatedAttributes` (*output*)

An array of attributes that will be added to the context should be returned using this parameter. Memory for the attribute structures should be allocated using the CSSM_UPCALLS callbacks provided to the service provider module when `CSSM_SPI_ModuleAttach()` is called.

DESCRIPTION

This function generates algorithm parameters for the specified context. These parameters include `Diffie-Hellman` key agreement parameters and DSA key generation parameters. In most cases the algorithm parameters will be added directly to the cryptographic context (by returning an array of CSSM_CONTEXT_ATTRIBUTE structures), but an algorithm may return some data to the caller via the `Param` parameter. The generated parameters are added to the context as an attribute of type CSSM_ATTRIBUTE_ALG_PARAMS. Other attributes returned are added to the context, or replace existing values in the context.

NOTES FOR API

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, pre-allocated output buffer, the caller must provide an array of one or more CSSM_DATA structures each, containing a Length field value greater than zero and a non-NULL data pointer field value. To specify automatic output buffer allocation by the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed.

NOTES FOR SPI

The output is returned to the caller as specified in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

GenerateKey

NAME

GenerateKey: CSSM_GenerateKey, CSP_GenerateKey – Generate a symmetric key (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_GenerateKey
(CSSM_CC_HANDLE CCHandle,
uint32 KeyUsage,
uint32 KeyAttr,
const CSSM_DATA *KeyLabel,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry,
CSSM_KEY_PTR Key)
SPI:
CSSM_RETURN CSSMCSPI CSP_GenerateKey
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
uint32 KeyUsage,
uint32 KeyAttr,
const CSSM_DATA *KeyLabel,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry,
CSSM_KEY_PTR Key)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`KeyUsage` (*input*)

A bit mask indicating all permitted uses for the new key.

`KeyAttr` (*input*)

A bit mask defining attribute values for the new key.

`KeyLabel` (input/optional)

Pointer to a byte string that will be used as the label for the key.

`CredAndAclEntry` (input/optional)

A structure containing one or more credentials authorized for creating a key and the prototype ACL entry that will control future use of the newly created key. The credentials and ACL entry prototype can be presented as immediate values or callback functions can be provided for use by the CSP to acquire the credentials and/or the ACL entry interactively. If

the CSP provides public access for creating a key, then the credentials can be NULL. If the CSP defines a default initial ACL entry for the new key, then the ACL entry prototype can be an empty list.

`Key (output)`

Pointer to CSSM_KEY structure used to hold the new key. The CSSM_KEY structure should be empty upon input to this function. The CSP will ignore any values residing in this structure at function invocation. Input values should be supplied in the cryptographic context, `KeyUsage`, `KeyAttr`, and `KeyLabel` input parameters.

SPI PARAMETERS

`CSPHandle (input)`

The handle that describes the add-in Cryptographic Service Provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

`Context (input)`

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

`Key (output)`

Pointer to CSSM_KEY structure used to obtain the key. Upon function invocation, any values in the CSSM_Key structure should be ignored. All input values should be supplied in the cryptographic `Context`, `KeyUsage`, `KeyAttr`, and `KeyLabel` input parameters.

DESCRIPTION

This function generates a symmetric key. The `KeyUsage`, and `KeyAttr` are used to initialize the keyheader for the newly created key. These values are not retained in the cryptographic Context, which contains additional parameters for this operation. The CSP may cache keying material associated with the new symmetric key. When the symmetric key is no longer in active use, the application can invoke the `CSSM_FreeKey()` interface to allow cached keying material associated with the symmetric key to be removed.

Authorization policy can restrict the set of callers who can create a new resource. In this case, the caller must present a set of access credentials for authorization. Upon successfully authenticating the credentials, the template that verified the presented samples identifies the ACL entry that will be used in the authorization computation. If the caller is authorized, the new resource is created.

The caller must provide an initial ACL entry to be associated with the newly created resource. This entry is used to control future access to the new resource and (since the subject is deemed to be the "Owner") exercise control over its associated ACL. The caller can specify the following items for initializing an ACL entry:

- Subject - A CSSM_LIST structure, containing the type of the subject and a template value that can be used to verify samples that are presented in credentials when resource access is requested.
- Delegation flag - A value indicating whether the Subject can delegate the permissions recorded in the `AuthorizationTag`. (This item only applies to public key subjects).
- Authorization tag - The set of permissions that are granted to the Subject.
- Validity period - The start time and the stop time for which the ACL entry is valid.
- ACL entry tag - A user-defined string value associated with the ACL entry.

  The service provider can modify the caller-provided initial ACL entry to conform to any innate resource-access policy that the service provider may be required to enforce. If the initial ACL entry provided by the caller contains values or permissions that are not supported by the service provider, then

the service provider can modify the initial ACL appropriately or can fail the request to create the new resource. Service providers list their supported `AuthorizationTag` values in their Module Directory Services primary record.

NOTES

The `KeyData` field of the CSSM_KEY structure is allocated by the CSP. The application is required to free this memory using the `CSSM_FreeKey()` (CSSM API), or `CSP_FreeKey()` (CSP SPI), function or with the memory functions registered for the `CSPHandle`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_KEY_LABEL_ALREADY_EXISTS`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_GenerateRandom, *CSSM_GenerateKeyPair*

Functions for the CSP SPI:

CSP_GenerateRandom, *CSP_GenerateKeyPair*

GenerateKeyP

NAME

GenerateKeyP – Generate a key with privilege (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_GenerateKeyP
(CSSM_CC_HANDLE CCHandle,
uint32 KeyUsage,
uint32 KeyAttr,
const CSSM_DATA *KeyLabel,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry,
CSSM_KEY_PTR Key,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`Privilege` (*input*)

The privilege to be applied during the cryptographic operation.

See *CSSM_GenerateKey* for other parameters.

DESCRIPTION

This function is similar to the `CSSM_GenerateKey()` function. It also accepts a USEE tag as a privilege request parameter. CSSM checks that either its own privilege set or the application's privilege set (if the application is signed) includes the tag. If the tag is found and the service provider privilege set indicates that it is supported, the tag is forwarded to the service provider.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_GenerateKeyPairP*, *CSSM_GenerateRandom*

GenerateKeyPair

NAME

GenerateKeyPair: CSSM_GenerateKeyPair, CSP_GenerateKeyPair – Generate an asymmetric key pair (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_GenerateKeyPair
(CSSM_CC_HANDLE CCHandle,
uint32 PublicKeyUsage,
uint32 PublicKeyAttr,
const CSSM_DATA *PublicKeyLabel,
CSSM_KEY_PTR PublicKey,
uint32 PrivateKeyUsage,
uint32 PrivateKeyAttr,
const CSSM_DATA *PrivateKeyLabel,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry,
CSSM_KEY_PTR PrivateKey)
SPI:
CSSM_RETURN CSSMCSPI CSP_GenerateKeyPair
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
uint32 PublicKeyUsage,
uint32 PublicKeyAttr,
const CSSM_DATA *PublicKeyLabel,
CSSM_KEY_PTR PublicKey,
uint32 PrivateKeyUsage,
uint32 PrivateKeyAttr
const CSSM_DATA *PrivateKeyLabel,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry,
CSSM_KEY_PTR PrivateKey,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`PublicKeyUsage` (*input*)

A bit mask indicating all permitted uses for the new public key.

`PublicKeyAttr` (*input*)

A bit mask defining attribute values for the new public key.

`PublicKeyLabel` (input/optional)

Pointer to a byte string that will be used as the label for the public key.

`PublicKey` (*output*)

Pointer to CSSM_KEY structure used to hold the new public key. The CSSM_KEY structure should be empty upon input to this function. The CSP will ignore any values residing in this structure at function invocation. Input values should be supplied in the cryptographic `Context`, `PublicKeyUsage`, `PublicKeyAttr`, and `PublicKeyLabel` input parameters.

`PrivateKeyUsage` (*input*)

A bit mask indicating all permitted uses for the new private key.

`PrivateKeyAttr` (*input*)

A bit mask defining attribute values for the new private key.

`PrivateKeyLabel` (input/optional)

Pointer to a byte string that will be used as the label for the private key.

`CredAndAclEntry` (input/optional)

A structure containing one or more credentials authorized for creating a key and the prototype ACL entry that will control future use of the newly created key. The credentials and ACL entry prototype can be presented as immediate values or callback functions can be provided for use by the CSP to acquire the credentials and/or the ACL entry interactively. If the CSP provides public access for creating a key, then the credentials can be NULL. If the CSP defines a default initial ACL entry for the new key, then the ACL entry prototype can be an empty list.

`PrivateKey` (*output*)

Pointer to CSSM_KEY structure used to obtain the private key. Upon function invocation, any values in the CSSM_Key structure should be ignored. All input values should be supplied in the cryptographic `Context`, `PrivateKeyUsage`, `PrivateKeyAttr`, and `PrivateKeyLabel` input parameters.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

`Privilege` (*input*)

The export privilege to be applied during the cryptographic operation. This parameter is forwarded to the CSP after CSSM verifies the caller and service provider privilege set includes the specified privilege.

DESCRIPTION

This function generates an asymmetric key pair. The CSP may cache keying material associated with the new asymmetric keypair. When one or both of the keys are no longer in active use, the application can invoke the `CSSM_FreeKey()` interface to allow cached keying material associated with the key to be removed.

Authorization policy can restrict the set of callers who can create a new resource. In this case, the caller must present a set of access credentials for authorization. Upon successfully authenticating the credentials, the template that verified the presented samples identifies the ACL entry that will be used in the authorization computation. If the caller is authorized, the new resource is created.

The caller must provide an initial ACL entry to be associated with the newly created resource. This entry is used to control future access to the new resource and (since the subject is deemed to be the "Owner") exercise control over its associated ACL. The caller can specify the following items for initializing an ACL entry:

- Subject - A CSSM_LIST structure, containing the type of the subject and a template value that can be used to verify samples that are presented in credentials when resource access is requested.
- Delegation flag - A value indicating whether the Subject can delegate the permissions recorded in the `AuthorizationTag`. (This item only applies to public key subjects).
- Authorization tag - The set of permissions that are granted to the Subject.
- Validity period - The start time and the stop time for which the ACL entry is valid.
- ACL entry tag - A user-defined string value associated with the ACL entry.

  The service provider can modify the caller-provided initial ACL entry to conform to any innate resource-access policy that the service provider may be required to enforce. If the initial ACL entry provided by the caller contains values or permissions that are not supported by the service provider, then the service provider can modify the initial ACL appropriately or can fail the request to create the new resource. Service providers list their supported `AuthorizationTag` values in their Module Directory Services primary record.

NOTES

The `KeyData` fields of the CSSM_KEY structures are allocated by the CSP. The application is required to free this memory using the `CSSM_FreeKey()` (CSSM API), or `CSP_FreeKey()` (CSP SPI), function or with the memory functions registered for the `CSPHandle`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_KEY_LABEL_ALREADY_EXISTS`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_GenerateKey, *CSSM_GenerateRandom*

Functions for the CSP SPI:

CSP_GenerateKey, *CSP_GenerateRandom*

GenerateKeyPairP

NAME

GenerateKeyPairP – Generate an asymmetric key pair with privilege (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_GenerateKeyPairP
(CSSM_CC_HANDLE CCHandle,
uint32 PublicKeyUsage,
uint32 PublicKeyAttr,
const CSSM_DATA *PublicKeyLabel,
CSSM_KEY_PTR PublicKey,
uint32 PrivateKeyUsage,
uint32 PrivateKeyAttr,
const CSSM_DATA *PrivateKeyLabel,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry,
CSSM_KEY_PTR PrivateKey,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`Privilege` (*input*)

The privilege to be applied during the cryptographic operation.

See *CSSM_GenerateKeyPair*.

DESCRIPTION

This function is similar to the `CSSM_GenerateKeyPair()` function. It also accepts a USEE tag as a privilege request parameter. CSSM checks that either its own privilege set or the application's privilege set (if the application is signed) includes the tag. If the tag is found and the service provider privilege set indicates that it is supported, the tag is forwarded to the service provider.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_KEY_LABEL_ALREADY_EXISTS`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *CSSM_GenerateKeyPair*

GenerateMac

NAME

GenerateMac: CSSM_GenerateMac, CSP_GenerateMac – Compute a message authentication code (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_GenerateMac
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount,
CSSM_DATA_PTR Mac)
SPI:
CSSM_RETURN CSSMCSPI CSP_GenerateMac
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
const CSSM_DATA *DataBufs,
uint32 DataBufCount,
CSSM_DATA_PTR Mac)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`DataBufs` (*input*)

A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

`DataBufCount` (*input*)

The number of DataBufs.

`Mac` (*output*)

A pointer to the CSSM_DATA structure for the Message Authentication Code.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DESCRIPTION

This function computes a message authentication code for all data contained in the set of input buffers.

NOTES ON API

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, preallocated output buffer, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value greater than zero and a non-NULL data pointer field value. To specify automatic output buffer allocation by the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed.

NOTES ON SPI

The output is returned to the caller as specified in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_OUTPUT_LENGTH_ERROR`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_GenerateMacInit, *CSSM_GenerateMacUpdate*, *CSSM_GenerateMacFinal*

Functions for the CSP SPI:

CSP_GenerateMacInit, *CSP_GenerateMacUpdate*, *CSP_GenerateMacFinal*

GenerateMacFinal

NAME

GenerateMacFinal: CSSM_GenerateMacFinal, CSP_GenerateMacFinal – Finalize the staged message authentication code (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_GenerateMacFinal
(CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR Mac)
SPI:
CSSM_RETURN CSSMCSPI CSP_GenerateMacFinal
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR Mac)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`Mac` (*output*)

A pointer to the CSSM_DATA structure for the message authentication code.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function finalizes the staged message authentication code function.

NOTES ON API

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, preallocated output buffer, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value greater than zero and a non-NULL data pointer field value. To specify automatic output buffer allocation by

the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed.

NOTES ON SPI

The output is returned to the caller as specified in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_OUTPUT_LENGTH_ERROR`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_GenerateMac, CSSM_GenerateMacInit, CSSM_GenerateMacUpdate

Functions for the CSP SPI:

CSP_GenerateMac, CSP_GenerateMacInit, CSP_GenerateMacUpdate

GenerateMacInit

NAME

GenerateMacInit: CSSM_GenerateMacInit, CSP_GenerateMacInit – Initialize the staged message authentication code (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_GenerateMacInit
(CSSM_CC_HANDLE CCHandle)
SPI:
CSSM_RETURN CSSMCSPI CSP_GenerateMacInit
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DESCRIPTION

This function initializes the staged message authentication code function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_GenerateMac, *CSSM_GenerateMacUpdate*, *CSSM_GenerateMacFinal*

Functions for the CSP SPI:

CSP_GenerateMac, *CSP_GenerateMacUpdate*, *CSP_GenerateMacFinal*

GenerateMacUpdate

NAME

GenerateMacUpdate: CSSM_GenerateMacUpdate, CSP_GenerateMacUpdate – Continue the staged process of computing a message authentication code (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_GenerateMacUpdate
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount)
SPI:
CSSM_RETURN CSSMCSPI CSP_GenerateMacUpdate
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`DataBufs` (*input*)

A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

`DataBufCount` (*input*)

The number of DataBufs.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function continues the staged process of computing a message authentication code over all data contained in the set of input buffers. The authentication code will be returned as a result of the final code generation step.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_GenerateMac, CSSM_GenerateMacInit, CSSM_GenerateMacFinal

Functions for the CSP SPI:

CSP_GenerateMac, CSP_GenerateMacInit, CSP_GenerateMacFinal

GenerateRandom

NAME

GenerateRandom: CSSM_GenerateRandom, CSP_GenerateRandom function – Generate random data (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_GenerateRandom
(CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR RandomNumber)
SPI:
CSSM_RETURN CSSMCSPI CSP_GenerateRandom
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
CSSM_DATA_PTR RandomNumber)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`RandomNumber` (*output*)

Pointer to CSSM_DATA structure used to obtain the random number and the size of the random number in bytes.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DESCRIPTION

This function generates random data.

NOTES ON API

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, preallocated output buffer, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value greater than zero and a non-NULL data pointer field value. To specify automatic output buffer allocation by the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed.

NOTES ON SPI

The output is returned to the caller as specified in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

GetOperationalStatistics

NAME

GetOperationalStatistics: CSSM_CSP_GetOperationalStatistics, CSP_GetOperationalStatistics – Get operational values of a subservice (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CSP_GetOperationalStatistics
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CSP_OPERATIONAL_STATISTICS *Statistics)
SPI:
CSSM_RETURN CSSMCSPI CSSM_CSP_GetOperationalStatistics
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CSP_OPERATIONAL_STATISTICS *Statistics)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

Handle of the Cryptographic Service Provider that will perform the operation.

`Statistics` (*output*)

Structure containing the subservice's current statistics.

DESCRIPTION

Obtain the current operational values of a subservice. The information is returned in a structure of type CSSM_CSP_OPERATIONAL_STATISTICS. This information includes login status and available storage space. The data structure to hold the returned results must be provided by the caller. The CSP does not allocate memory on behalf of the caller.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

GetTimeValue

NAME

GetTimeValue: CSSM_GetTimeValue, CSP_GetTimeValue – Get a CSP time value (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_GetTimeValue
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ALGORITHMS TimeAlgorithm,
CSSM_DATA *TimeData)
SPI:
CSSM_RETURN CSSMCSPI CSP_GetTimeValue
(CSSM_CSP_HANDLE CSPHandle,
CSSM_ALGORITHMS TimeAlgorithm,
kCSSM_DATA *TimeData)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

Handle of the Cryptographic Service Provider that will perform the operation.

`TimeAlgorithm` (*input*)

A CSSM algorithm type that indicates the method for fetching the time. The following algorithm types are supported:

CSSM_ALGID_UTC Returns a time value in the form YYYYMMDDhhmmss (4 characters for the year; 2 characters each for the month, the day, the hour, the minute, and the second). The time returned is GMT.

CSSM_ALGID_RUNNING_COUNTER The current value of a running hardware counter that operates while the device is in operation. This value can be read from a processor counter provided by some platform architectures.

`TimeData` (*output*)

The time value of counter value returned in response to the request.

DESCRIPTION

This function returns a time value maintained by a CSP. This feature will be supported primarily by hardware tokens with an onboard real time clock.

NOTES

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, preallocated output buffer, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value greater than zero and a non-NULL data pointer field value. To specify automatic output buffer allocation by the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed.

Some tokens require authentication before returning a time value.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

MDS_Initialize

NAME

MDS_Initialize – Initiate service context with MDS (CDSA)

SYNOPSIS

include <cdsa/mds.h>

```
CSSM_RETURN CSSMAPI MDS_Initialize
(const CSSM_GUID *pCallerGuid,
const CSSM_DATA *pCallerManifest,
const CSSM_MEMORY_FUNCS *pMemoryFunctions,
MDS_FUNCS_PTR pDlFunctions,
MDS_HANDLE *hMds)
```

LIBRARY

Module Directory Services library (`cdsa$mds300_shr.exe`)

PARAMETERS

`pCallerGuid` (input/optional)

The GUID of the module calling MDS.

`pCallerManifest` (input/optional)

The Manifest of the module calling MDS.

`pMemoryFunctions` (*input*)

The memory-management routines MDS uses to allocate query results on behalf of the caller.

`pDlFunctions` (*output*)

The function table containing MDS programming interfaces for database access.

`hMds` (*output*)

A new handle that can be used to interact with the MDS. The value will be set to CSSM_INVALID_HANDLE if the function fails.

DESCRIPTION

This function initiates a service context with MDS and returns an opaque handle corresponding to that context. The caller provides memory functions that MDS can use to manage memory in the caller's space on behalf of the caller. The caller also provides input/output table pDlFunctions to get access to MDS databases.

If the caller is a CDSA service provider that will require write-access to an MDS database, (such as a module that supports dynamic insertion and removal events), then the caller can provide the caller's GUID as input parameter `pCallerGuid`. When provided as input, the GUID is associated with the MDS handle and is used during `DbOpen` processing. If write-access is requested during `DbOpen`, MDS uses the associated GUID to locate the service provider's signed manifest credentials in the `DS Common` relation. The service provider module and its credentials are verified to ensure that write-access is permitted on this database by this module.

The installers will have to provide the `pCallerManifest` instead of `pCallerGuid`, as GUID cannot be used to locate an application unless it is installed. Only one of the two parameters `pCallerGuid` and `pCallerManifest` should be non NULL in an `MDS_Initialize()` call, otherwise an error will be returned.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_INVALID_POINTER
CSSMERR_DL_INTERNAL_ERROR
CSSMERR_DL_MEMORY_ERROR
CSSMERR_DL_FUNCTION_FAILED
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

MDS_Install

NAME

MDS_Install – Create the object directory database (CDSA)

SYNOPSIS

#include <cdsa/mds.h>

```
CSSM_RETURN CSSMAPI MDS_Install
(MDS_HANDLE MdsHandle)
```

LIBRARY

Module Directory Services library (`cdsa$mds300_shr.exe`)

PARAMETERS

`MdsHandle` (*input*)

The MDS handle identifying an MDS context.

DESCRIPTION

This function creates the Object Directory database containing the `Object` relation, and the CDSA Directory database containing the set of CDSA-specific relations defined in this specification. The `MdsHandle` identifies an MDS context created by invoking `MDS_Initialize()`. The context contains information about the access rights of the caller. Write-access is required to perform this operation.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_INVALID_DL_HANDLE
CSSMERR_DL_DATASTORE_ALREADY_EXISTS
CSSMERR_DL_INVALID_ACCESS_REQUEST
CSSMERR_DL_INVALID_DB_LOCATION
CSSMERR_DL_INVALID_DB_NAME
CSSMERR_DL_INVALID_OPEN_PARAMETERS
CSSMERR_DL_INVALID_RECORD_INDEX
CSSMERR_DL_INVALID_RECORDTYPE
CSSMERR_DL_INVALID_FIELD_NAME
CSSMERR_DL_UNSUPPORTED_FIELD_FORMAT
CSSMERR_DL_UNSUPPORTED_INDEX_INFO
CSSMERR_DL_UNSUPPORTED_LOCALITY
CSSMERR_DL_UNSUPPORTED_NUM_ATTRIBUTES
CSSMERR_DL_UNSUPPORTED_NUM_INDEXES
CSSMERR_DL_UNSUPPORTED_NUM_RECORDTYPES
```

```
CSSMERR_DL_UNSUPPORTED_RECORDTYPE
CSSMERR_DL_FIELD_SPECIFIED_MULTIPLE
CSSMERR_DL_INCOMPATIBLE_FIELD_FORMAT
CSSMERR_DL_INVALID_PARSING_MODULE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

MDS_Terminate

NAME

MDS_Terminate – Terminate the MDS service context (CDSA)

SYNOPSIS

include <cdsa/mds.h>

```
CSSM_RETURN CSSMAPI MDS_Terminate
(MDS_HANDLE MdsHandle)
```

LIBRARY

Module Directory Services library (`cdsa$mds300_shr.exe`)

PARAMETERS

`MdsHandle` (*input*)

The MDS handle corresponding to the context being terminated.

DESCRIPTION

This function terminates the MDS service context identified by the opaque `MdsHandle`. The MDS handle is invalidated and MDS frees all internal resources associated with the context.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_DL_INVALID_DL_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

MDS_Uninstall

NAME

MDS_Uninstall – Delete the object directory database (CDSA)

SYNOPSIS

include <cdsa/mds.h>

```
CSSM_RETURN CSSMAPI MDS_Uninstall
(MDS_HANDLE MdsHandle)
```

LIBRARY

Module Directory Services library (`cdsa$mds300_shr.exe`)

PARAMETERS

`MdsHandle` (*input*)

The MDS handle identifying a valid MDS context.

DESCRIPTION

This function deletes the Object Directory database containing the `Object` relation, and the CDSA Directory database containing the set of CDSA-specific relations defined in this specification. The `MdsHandle` identifies the MDS context created by invoking `MDS_Initialize()`. The context contains information about the access rights of the caller. Write-access is required to perform this operation.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_DL_INVALID_DL_HANDLE CSSMERR_DL_DATASTORE_IS_OPEN
CSSMERR_DL_INVALID_DB_LOCATION CSSMERR_DL_INVALID_DB_NAME
CSSMERR_DL_DATASTORE_DOESNOT_EXIST
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

MDSUTIL_FreeModuleInfo

NAME

MDSUTIL_FreeModuleInfo – Frees memory associated with the MDSUTIL_GetModuleInfo function.

SYNOPSIS

include <mds_util_api.h>
include <mds_util_helper.h>

```
CSSM_RETURN CSSMAPI MDSUTIL_FreeModuleInfo
(MDSUTIL_MODULE_INFO_PTR ModuleInfo)
```

LIBRARY

Module Directory Services library (`cdsa$mds300_shr.exe`)

PARAMETERS

`ModuleInfo` (*input*)

A pointer to the data to be freed.

DESCRIPTION

This routine frees the list of module information that was returned by MDSUTIL_GetModuleInfo. All substructures within the info structure are freed by this function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition.The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_INVALID_POINTER
CSSMERR_CSSM_NOT_INITIALIZED

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *MDSUTIL_ModuleInstall, MDSUTIL_ModuleUninstall, MDSUTIL_ListModules, MDSUTIL_GetModuleInfo, MDSUTIL_GetCredLocationFromGUID, MDSUTIL_FreeModuleList, MDSUTIL_ListModuleManagers, MDSUTIL_GetModuleManagerInfo, MDSUTIL_ModuleManagerInstall, MDSUTIL_ModuleManagerUninstall, MDSUTIL_Init, MDSUTIL_Term*

MDSUTIL_FreeModuleList

NAME

MDSUTIL_FreeModuleList – Frees the list of add-in modules that was returned by MDSUTIL_ListModules.

SYNOPSIS

```
# include <mds_util_api.h>
# include <mds_util_helper.h>

CSSM_RETURN CSSMAPI MDSUTIL_FreeModuleList
(MDSUTIL_LIST_PTR List)
```

LIBRARY

Module Directory Services library (`cdsa$mds300_shr.exe`)

PARAMETERS

`List` (*input*)

A pointer to a MDSUTIL_LIST pointer.

DESCRIPTION

This routine frees the list of add-in modules that was returned by MDSUTIL_ListModules.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition.The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_INVALID_POINTER

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *MDSUTIL_ModuleInstall, MDSUTIL_ModuleUninstall, MDSUTIL_ListModules, MDSUTIL_GetModuleInfo, MDSUTIL_GetCredLocationFromGUID, MDSUTIL_FreeModuleInfo, MDSUTIL_ListModuleManagers, MDSUTIL_GetModuleManagerInfo, MDSUTIL_ModuleManagerInstall, MDSUTIL_ModuleManagerUninstall, MDSUTIL_Init, MDSUTIL_Term*

MDSUTIL_GetCredLocationFromGUID

NAME

MDSUTIL_GetCredLocationFromGUID – Returns the location of the add-in module, and the associated credentials file for the add-in module.

SYNOPSIS

include <mds_util_api.h>
include <mds_util_helper.h>

```
CSSM_RETURN CSSMAPI MDSUTIL_GetCredLocationFromGUID
(const CSSM_GUID *ModuleGUID,
CSSM_DATA *pModulePath,
CSSM_DATA *pModuleCredentialPath,
CSSM_API_MEMORY_FUNCS_PTR MemoryFuncs)
```

LIBRARY

Module Directory Services library (`cdsa$mds300_shr.exe`)

PARAMETERS

`ModuleGUID` (*input*)

A pointer to the module's Globally Unique ID.

`pModulePath` (*output*)

A pointer to the module's full filespec location.

`pModuleCredentialPath` (*output*)

A pointer to the module's credential full filespec location.

`MemoryFuncs` (*input*)

The memory-management routines MDS uses to allocate query results on behalf of the caller.

DESCRIPTION

This function returns the location of the add-in module, and the associated credentials file for the add-in module. The caller is responsible for freeing the memory in the output parameters.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_INVALID_GUID
CSSMERR_CSSM_MDS_ERROR

CSSM_ERRCODE_INVALID_OUTPUT_POINTER
CSSM_ERRCODE_MEMORY_ERROR

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *MDSUTIL_ModuleInstall, MDSUTIL_ModuleUninstall, MDSUTIL_ListModules, MDSUTIL_GetModuleInfo, MDSUTIL_FreeModuleInfo, MDSUTIL_FreeModuleList, MDSUTIL_ListModuleManagers, MDSUTIL_GetModuleManagerInfo, MDSUTIL_ModuleManagerInstall, MDSUTIL_ModuleManagerUninstall, MDSUTIL_Init, MDSUTIL_Term*

MDSUTIL_GetModuleInfo

NAME

MDSUTIL_GetModuleInfo – Gets information from the MDS registry for the add-in module.

SYNOPSIS

```
# include <mds_util_api.h>
# include <mds_util_helper.h>

CSSM_RETURN CSSMAPI MDSUTIL_GetModuleInfo
(const CSSM_GUID *ModuleGUID,
CSSM_SERVICE_MASK UsageMask,
uint32 SubserviceID,
CSSM_USEE_TAG USEERequest,
MDSUTIL_MODULE_INFO_PTR *pModuleInfo)
```

LIBRARY

Module Directory Services library (`cdsa$mds300_shr.exe`)

PARAMETERS

`ModuleGUID` (*input*)

A pointer to the CSSM_GUID structure containing the Globally Unique ID of the add-in module.

`UsageMask` (*input*)

A bit mask specifying the module usage types used to restrict the capabilities information returned by this function. An input value of zero specifies all usages for the specified module.

`SubserviceID` (*input*)

A single subservice ID. Note that the operation may already be limited by a service mask. If so, the subservice ID applies to all service categories selected by the service mask.

`USEERequest` (*input*)

United States Export Exemption tag; should be set to CSSM_USEE_NONE.

`pModuleInfo` (*output*)

A pointer to the module information.

DESCRIPTION

This function gets a list of descriptive information from the MDS registry for the add-in module identified by the ModuleGUID. The information returned can include all of the capability information for each of the subservices for each of the service types implemented by the selected module. The request for information can be limited to a particular set of services, as specified by the UsageMask. The request may be further limited to one or all of the subservices implemented in one or all of the service categories. The MDSUTIL_FreeModuleInfo function must be called to deallocate memory containing the list.

RETURN VALUE

NULL — Error in retrieving information from the MDS registry.

Not NULL — A pointer to a module info structure containing a pointer to an array of zero or more service information structures. Each structure contains type information identifying the service description as representing certificate library services, data storage library services, and so on. The service descriptions are sub-classed into subservice descriptions that describe the attributes and capabilities of a subservice.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_INVALID_POINTER
CSSMERR_CSSM_INVALID_GUID
CSSM_INVALID_SUBSERVICEID
CSSMERR_CSSM_MEMORY_ERROR
CSSMERR_CSSM_NOT_INITIALIZED
CSSM_ERRCODE_MDS_ERROR

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *MDSUTIL_ModuleInstall, MDSUTIL_ModuleUninstall, MDSUTIL_ListModules, MDSUTIL_GetCredLocationFromGUID, MDSUTIL_FreeModuleInfo, MDSUTIL_FreeModuleList, MDSUTIL_ListModuleManagers, MDSUTIL_GetModuleManagerInfo, MDSUTIL_ModuleManagerInstall, MDSUTIL_ModuleManagerUninstall, MDSUTIL_Init, MDSUTIL_Term*

MDSUTIL_GetModuleManagerInfo

NAME

MDSUTIL_GetModuleManagerInfo – Returns descriptive information about the elective module manager identified by the GUID or the service mask.

SYNOPSIS

include <mds_util_api.h>
include <mds_util_helper.h>

```
CSSM_RETURN CSSMAPI MDSUTIL_GetModuleManagerInfo
(const CSSM_GUID *ModuleGUID,
CSSM_SERVICE_MASK ServiceType,
MDSUTIL_MODULE_MANAGER_INFO_PTR *ModuleManagerInfo)
```

LIBRARY

Module Directory Services library (`cdsa$mds300_shr.exe`)

PARAMETERS

`ModuleGUID` (*input*)

A pointer to a GUID identifying the module manager.

`ServiceType` (*input*)

A unique service mask identifying the module manager.

`ModuleManagerInfo` (*output*)

A pointer to the returned module manager information.

DESCRIPTION

This function returns descriptive information about the elective module manager identified by the GUID or the service mask.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. NULL indicates that the routine was unable to get the module manager information.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_INVALID_POINTER
CSSMERR_CSSM_INVALID_GUID
CSSM_ERRCODE_MDS_ERROR
CSSMERR_CSSM_INVALID_SERVICE_MASK
CSSM_ERRCODE_MEMORY_ERROR
CSSMERR_CSSM_MEMORY_ERROR

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *MDSUTIL_ModuleInstall, MDSUTIL_ModuleUninstall, MDSUTIL_ListModules, MDSUTIL_GetModuleInfo, MDSUTIL_GetCredLocationFromGUID, MDSUTIL_FreeModuleInfo, MDSUTIL_FreeModuleList, MDSUTIL_ListModuleManagers, MDSUTIL_ModuleManagerInstall, MDSUTIL_ModuleManagerUninstall, MDSUTIL_Init, MDSUTIL_Term*

MDSUTIL_Init

NAME

MDSUTIL_Init – Initializes the MDS registry in preparation for a series of MDSUTIL operations.

SYNOPSIS

```
# include <mds_util_api.h>
# include <mds_util_helper.h>

CSSM_RETURN CSSMAPI MDSUTIL_Init
(CSSM_BOOL ReadWrite)
```

LIBRARY

Module Directory Services library (`cdsa$mds300_shr.exe`)

PARAMETERS

`ReadWrite(`*input*`)`

A Boolean flag indicating whether the MDS registry is to be enabled for writing as well as reading. CSSM_TRUE indicates that the registry should be enabled for writing.

DESCRIPTION

This function initializes the MDS registry in preparation for a series of MDSUTIL operations.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition.The value CSSM_OK indicates success. All other values indicate an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *MDSUTIL_ModuleInstall, MDSUTIL_ModuleUninstall, MDSUTIL_ListModules, MDSUTIL_GetModuleInfo, MDSUTIL_GetCredLocationFromGUID, MDSUTIL_FreeModuleInfo, MDSUTIL_FreeModuleList, MDSUTIL_ListModuleManagers, MDSUTIL_GetModuleManagerInfo, MDSUTIL_ModuleManagerInstall, MDSUTIL_ModuleManagerUninstall, MDSUTIL_Term*

MDSUTIL_ListModuleManagers

NAME

MDSUTIL_ListModuleManagers – Returns the number of module managers and a list of GUIDs associated with those module managers.

SYNOPSIS

```
# include <mds_util_api.h>
# include <mds_util_helper.h>

CSSM_RETURN CSSMAPI MDSUTIL_ListModuleManagers
(CSSM_GUID_PTR *ModuleManagerGuids,
uint32 *NumberOfModuleManagers)
```

LIBRARY

Module Directory Services library (`cdsa$mds300_shr.exe`)

PARAMETERS

`ModuleManagerGuids` (*output*)

A pointer to a list of GUIDs.

`NumberOfModuleManagers` (*output*)

A pointer to the number of module managers.

DESCRIPTION

This function returns the number of module managers and a list of GUIDs associated with those module managers. The caller is responsible for freeing the memory associated with the GUID list.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition.The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_INVALID_POINTER
CSSM_ERRCODE_MEMORY_ERROR

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *MDSUTIL_ModuleInstall, MDSUTIL_ModuleUninstall*, *MDSUTIL_ListModules*, *MDSUTIL_GetModuleInfo*, *MDSUTIL_GetCredLocationFromGUID*, *MDSUTIL_FreeModuleInfo*, *MDSUTIL_FreeModuleList*, *MDSUTIL_GetModuleManagerInfo*, *MDSUTIL_ModuleManagerInstall*, *MDSUTIL_ModuleManagerUninstall*, *MDSUTIL_Init*, *MDSUTIL_Term*

MDSUTIL_ListModules

NAME

MDSUTIL_ListModules – Returns a list containing the GUID/version/name for each of the currently installed service provider modules that provide services in any of the CSSM functional categories selected in the usage mask. The MDSUTIL_FreeModuleList function must be called to deallocate memory containing the list.

SYNOPSIS

include <mds_util_api.h>
include <mds_util_helper.h>

```
CSSM_RETURN CSSMAPI MDSUTIL_ListModules
(CSSM_SERVICE_MASK UsageMask,
CSSM_BOOL MatchAll,
MDSUTIL_LIST_PTR *pList)
```

LIBRARY

Module Directory Services library (`cdsa$mds300_shr.exe`)

PARAMETERS

`UsageMask` (*input*)

A bit mask selecting CSSM functional categories of interest for selecting information about potential service provider modules.

`MatchAll` (*input*)

A Boolean value to indicate if the add-in has to match all of the conditions expressed in UsageMask. TRUE means all conditions must be met. FALSE means one or more conditions must be met.

`pList` (*output*)

Pointer to a list of modules. Each item contains a CSSM_GUID, the module version, and a descriptive string name of the module.

DESCRIPTION

This function returns a list containing the GUID/version/name for each of the currently installed service provider modules that provide services in any of the CSSM functional categories selected in the usage mask. The MDSUTIL_FreeModuleList function must be called to deallocate memory containing the list.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition.The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSM_ERRCODE_MDS_ERROR
CSSMERR_CSSM_INVALID_POINTER
CSSM_ERRCODE_INVALID_OUTPUT_POINTER
CSSM_ERRCODE_MEMORY_ERROR
CSSMERR_NOT_INITIALIZED

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *MDSUTIL_ModuleInstall*, *MDSUTIL_ModuleUninstall*, *MDSUTIL_GetModuleInfo*, *MDSUTIL_GetCredLocationFromGUID*, *MDSUTIL_FreeModuleInfo*, *MDSUTIL_FreeModuleList*, *MDSUTIL_ListModuleManagers*, *MDSUTIL_GetModuleManagerInfo*, *MDSUTIL_ModuleManagerInstall*, *MDSUTIL_ModuleManagerUninstall*, *MDSUTIL_Init*, *MDSUTIL_Term*

MDSUTIL_ModuleInstall

NAME

MDSUTIL_ModuleInstall – Updates the MDS registry with information on the add-in module

SYNOPSIS

include <mds_util_api.h>
include <mds_util_helper.h>

```
CSSM_RETURN CSSMAPI MDSUTIL_ModuleInstall
(const char *ModuleName,
const char *ModuleFileNames,
const char *ModulePathName,
const char *ModuleCredentialName,
const char *ModulCredentialPath,
const CSSM_GUID *GUID,
const MDSUTIL_MODULE_INFO *ModuleDescription,
const void *Reserved1,
const CSSM_DATA *Reserved2)
```

LIBRARY

Module Directory Services utility API library (`cdsa$mds_util_api.olb`)

PARAMETERS

`ModuleName` (*input*)

The name of the add-in module.

`ModuleFileNames` (*input*)

The name of the file implementing the add-in module.

`ModulePathName` (*input*)

The path to the file implementing the add-in module.

`ModuleCredentialName` (*input*)

The name of the credential file for the add-in module.

`ModuleCredentialPath` (*input*)

The path to the credential file for the add-in module.

`GUID` (*input*)

The Globally Unique ID of the add-in module.

`ModuleDescription` (*input*)

A pointer to a structure that describes the add-in module.

`Reserved1` (*input*)

Reserved data.

`Reserved2` (*input*)

Reserved data.

DESCRIPTION

This function updates the MDS registry with information on the add-in module.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition.The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_INVALID_POINTER
CSSMERR_DL_OS_ACCESS_DENIED
CSSMERR_CSSM_INTERNAL_ERROR
CSSM_ERRCODE_INTERNAL_ERROR

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *MDSUTIL_ModuleUninstall*, *MDSUTIL_ListModules*, *MDSUTIL_GetModuleInfo*, *MDSUTIL_GetCredLocationFromGUID*, *MDSUTIL_FreeModuleInfo*, *MDSUTIL_FreeModuleList*, *MDSUTIL_ListModuleManagers*, *MDSUTIL_GetModuleManagerInfo*, *MDSUTIL_ModuleManagerInstall*, *MDSUTIL_ModuleManagerUninstall*, *MDSUTIL_Init*, *MDSUTIL_Term*

MDSUTIL_ModuleManagerInstall

NAME

MDSUTIL_ModuleManagerInstall – Updates the MDS registry with information about the Extensible Module Manager

SYNOPSIS

include <mds_util_api.h>
include <mds_util_helper.h>

```
CSSM_RETURN CSSMAPI MDSUTIL_ModuleManagerInstall
(const char *ModuleManagerName,
const char *ModuleManagerFileName,
const char *ModuleManagerPathName,
const char *ModuleManager CredentialName,
const char *ModuleManagerCredentialPath,
const CSSM_GUID *ModuleManagerGuid,
const MDSUTIL_MODULE_MANAGER_INFO *ModuleManagerDescription,
const void *Reserved1,
const CSSM_DATA *Reserved2)
```

LIBRARY

Module Directory Services utility API library (`cdsa$mds_util_api.olb`)

PARAMETERS

`ModuleManagerName` (*input*)

A pointer to the name of the Extensible Module Manager (EMM).

`ModuleManagerFileName` (*input*)

A pointer to the filename of the Extensible Module Manager.

`ModuleManagerPathname` (*input*)

A pointer to the directory path to the file implementing the EMM.

`ModuleManagerCredentialName` (*input*)

A pointer to the name of the credential file of the EMM.

`ModuleManagerCredentialPath` (*input*)

A pointer to the directory path to the credential file of the EMM.

`ModuleManagerGuid` (*input*)

A pointer to the Globally Unique ID of the EMM.

`ModuleManagerDescription` (*input*)

A pointer to the structure that describes the EMM.

DESCRIPTION

This function updates the MDS registry with information about the Extensible Module Manager.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition.The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_INVALID_POINTER
CSSMERR_CSSM_INTERNAL_ERROR
CSSMERR_CSSM_FUNCTION_FAILED

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *MDSUTIL_ModuleInstall*, *MDSUTIL_ModuleUninstall*, *MDSUTIL_ListModules*, *MDSUTIL_GetModuleInfo*, *MDSUTIL_GetCredLocationFromGUID*, *MDSUTIL_FreeModuleInfo*, *MDSUTIL_FreeModuleList*, *MDSUTIL_ListModuleManagers*, *MDSUTIL_GetModuleManagerInfo*, *MDSUTIL_ModuleManagerUninstall*, *MDSUTIL_Init*, *MDSUTIL_Term*

MDSUTIL_ModuleManagerUninstall

NAME

MDSUTIL_ModuleManagerUninstall – Removes from the MDS registry the information associated with the Globally Unique ID of the EMM

SYNOPSIS

include <mds_util_api.h>
include <mds_util_helper.h>

```
CSSM_RETURN CSSMAPI MDSUTIL_ModuleManagerUninstall
(const CSSM_GUID *ModuleManagerGuid)
```

LIBRARY

Module Directory Services utility API library (`cdsa$mds_util_api.olb`)

PARAMETERS

`ModuleManagerGuid` (*input*)

A pointer to the Globally Unique ID of the Extensible Module Manager.

DESCRIPTION

This function removes from the MDS registry the information associated with the Globally Unique ID of the EMM.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition.The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_INVALID_GUID
CSSMERR_CSSM_FUNCTION_FAILED
CSSM_ERRCODE_MDS_ERROR

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *MDSUTIL_ModuleInstall, MDSUTIL_ModuleUninstall*, *MDSUTIL_ListModules*, *MDSUTIL_GetModuleInfo*, *MDSUTIL_GetCredLocationFromGUID*, *MDSUTIL_FreeModuleInfo*, *MDSUTIL_FreeModuleList*, *MDSUTIL_ListModuleManagers*, *MDSUTIL_GetModuleManagerInfo*, *MDSUTIL_ModuleManagerInstall*, *MDSUTIL_Init*, *MDSUTIL_Term*

MDSUTIL_ModuleUninstall

NAME

MDSUTIL_ModuleUninstall – Removes from the MDS registry the information associated with GUID

SYNOPSIS

```
# include <mds_util_api.h>
# include <mds_util_helper.h>

CSSM_RETURN CSSMAPI MDSUTIL_ModuleUninstall
(const CSSM_GUID *ModuleGUID)
```

LIBRARY

Module Directory Services utility API library (`cdsa$mds_util_api.olb`)

PARAMETERS

`ModuleGUID` (*input*)

The Globally Unique ID of the add-in module.

DESCRIPTION

This function removes from the MDS registry the information associated with GUID.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_MEMORY_ERROR

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *MDSUTIL_ModuleInstall*, *MDSUTIL_ListModules*, *MDSUTIL_GetModuleInfo*, *MDSUTIL_GetCredLocationFromGUID*, *MDSUTIL_FreeModuleInfo*, *MDSUTIL_FreeModuleList*, *MDSUTIL_ListModuleManagers*, *MDSUTIL_GetModuleManagerInfo*, *MDSUTIL_ModuleManagerInstall*, *MDSUTIL_ModuleManagerUninstall*, *MDSUTIL_Init*, *MDSUTIL_Term*

MDSUTIL_Term

NAME

MDSUTIL_Term – Closes the MDS registry after a series of operations.

SYNOPSIS

```
# include <mds_util_api.h>
# include <mds_util_helper.h>
void CSSMAPI MDSUTIL_Term()
```

LIBRARY

Module Directory Services library (`cdsa$mds300_shr.exe`)

PARAMETERS

None

DESCRIPTION

This function closes the MDS registry after a series of operations.

RETURN VALUE

None

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *MDSUTIL_ModuleInstall, MDSUTIL_ModuleUninstall, MDSUTIL_ListModules, MDSUTIL_GetModuleInfo, MDSUTIL_GetCredLocationFromGUID, MDSUTIL_FreeModuleInfo, MDSUTIL_FreeModuleList, MDSUTIL_ListModuleManagers, MDSUTIL_GetModuleManagerInfo, MDSUTIL_ModuleManagerInstall, MDSUTIL_ModuleManagerUninstall, MDSUTIL_Init*

ObtainPrivateKeyFromPublicKey

NAME

ObtainPrivateKeyFromPublicKey: CSSM_CSP_ObtainPrivateKeyFromPublicKey, CSP_ObtainPrivateKeyFromPublicKey – Convert public key to private key (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CSP_ObtainPrivateKeyFromPublicKey
(CSSM_CSP_HANDLE CSPHandle
const CSSM_KEY *PublicKey,
CSSM_KEY_PTR PrivateKey)
SPI:
CSSM_RETURN CSSMCSPI CSP_ObtainPrivateKeyFromPublicKey
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_KEY *PublicKey,
CSSM_KEY_PTR PrivateKey)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the module to perform this operation.

`PublicKey` (*input*)

The public key corresponding to the private key being sought.

`PrivateKey` (*output*)

A reference to the private key corresponding to the public key.

DESCRIPTION

Given a public key this function returns a reference to the private key. The private key and its associated passphrase can be used as an input to any function requiring a private key value.

NOTES

The `KeyData` field of the CSSM_KEY structure is allocated by the CSP. The application is required to free this memory using the `CSSM_FreeKey()` (CSSM API), or `CSP_FreeKey()` (CSP SPI), function or with the memory functions registered for the `CSPHandle`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_PRIVATE_KEY_NOT_FOUND`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

PassThrough

NAME

PassThrough: CSSM_CSP_PassThrough, CSP_PassThrough – Extend crypto functionality (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_CSP_PassThrough
(CSSM_CC_HANDLE CCHandle,
uint32 PassThroughId,
const void *InData,
void **OutData)
SPI:
CSSM_RETURN CSSMCSPI CSP_PassThrough
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
uint32 PassThroughId,
const void *InData,
void **OutData)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation.

`PassThroughId` (*input*)

An identifier specifying the custom function to be performed.

`InData` (*input*)

A pointer to a module, implementation-specific structure containing the input data.

`OutData` (*output*)

A pointer to a module, implementation-specific structure containing the output data. The service provider will allocate the memory for this structure. The application should free the memory for the structure.

SPI PARAMETERS

`CSPHandle` (*input*)

Handle of the CSP supporting the `PassThrough` function.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this custom context structure.

DESCRIPTION

The `CSSM_CSP_PassThrough()` (CSSM API), or `CSP_PassThrough()` (CSP SPI), function is provided to allow CSP developers to extend the crypto functionality of the CSSM API.

NOTES

The `CSP_EventNotify()` function is used by the CSSM Core to interact with the CSP module.

Because this function is only exposed to CSSM as a function pointer, the function name internal to the CSP can be assigned at the discretion of the CSP module developer. However, the parameter list and return value types must match those defined for this function.

The error codes given in this section constitute the generic error codes, which may be used by all CSP libraries to describe common error conditions. CSP module developers may also define their own module-specific error codes.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_INVALID_PASSTHROUGH_ID`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Print_CDSA_Error

NAME

Print_CDSA_Error – Output the CDSA error strings to SYS$OUTPUT

LIBRARY

Common Security Services Manager library (CDSA$INCSSM300_SHR.EXE)

PARAMETERS

Error_Code (input) The numeric error code return by CDSA routines.

SYNOPSIS

include <cssm.h>

```
void Print_CDSA_Error( CSSM_RETURN  Error_Code);
```

DESCRIPTION

This routine outputs the strings returned by `Decode_CDSA_Error` to `SYS$OUTPUT`. It provides a simple way report CDSA errors from a user program.

RETURN VALUE

None.

ERRORS

None.

QueryKeySizeInBits

NAME

QueryKeySizeInBits: CSSM_QueryKeySizeInBits, CSP_QueryKeySizeInBits – Get CSP logical and effective sizes (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_QueryKeySizeInBits
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_KEY *Key,
CSSM_KEY_SIZE_PTR KeySize)
SPI:
CSSM_RETURN CSSMCSPI CSP_QueryKeySizeInBits
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
const CSSM_KEY *Key,
CSSM_KEY_SIZE_PTR KeySize)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CSPHandle` (input/optional)

The handle that describes the Cryptographic Service Provider module used to perform this function.

For the API, this parameter is ignored if a valid cryptographic context handle is specified.

`CCHandle` (input/optional)

A handle to a context that describes a cryptographic operation. The cryptographic context should contain a handle to the CSP that is being queried and the key about which key-size information is being requested.

`Key` (input/optional)

A pointer to a CSSM_KEY structure containing the key about which key-size information is being requested. This parameter is ignored if a valid cryptographic context handle is specified.

`KeySize` (*output*)

Pointer to a CSSM_KEY_SIZE data structure. The logical and effective sizes (in bits) for the key are returned in this structure.

For the API, if no context handle is provided, only the CSSM_KEY_SIZE `LogicalKeySizeInBits` field is set.

SPI PARAMETERS

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DESCRIPTION

This function queries a Cryptographic Service Provider (CSP) for the logical and effective sizes of a specified key.

The Cryptographic Service Provider (handle) and the key can be specified either in the cryptographic context or as parameters to the function call. If a valid cryptographic context handle parameter is specified, the CSP handle and key parameters are ignored.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_QUERY_SIZE_UNKNOWN`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_GenerateRandom, *CSSM_GenerateKeyPair*, *CSSM_GenerateKey*

Functions for the CSP SPI:

CSP_GenerateRandom, *CSP_GenerateKeyPair*, *CSP_GenerateKey*

QuerySize

NAME

QuerySize: CSSM_QuerySize, CSP_QuerySize – Get size of the output data (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_QuerySize
(CSSM_CC_HANDLE CCHandle,
CSSM_BOOL Encrypt,
uint32 QuerySizeCount,
CSSM_QUERY_SIZE_DATA_PTR DataBlockSizes)
SPI:
CSSM_RETURN CSSMCSPI CSP_QuerySize
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
CSSM_BOOL Encrypt,
uint32 QuerySizeCount,
CSSM_QUERY_SIZE_DATA_PTR DataBlockSizes)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle for an encryption and decryption context.

`Encrypt` (*input*)

A boolean indicating whether encryption is the operation for which the output data size should be calculated. If CSSM_TRUE, the operation is encryption. If CSSM_FALSE the operation is decryption.

`QuerySizeCount` (*input*)

The number of entries in the array of DataBlockSizes.

`DataBlockSizes` (input/output)

An array of data block input sizes and corresponding entries for the data block output sizes that are returned by this function.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DESCRIPTION

This function queries for the size of the output data for a cryptographic operation. If the context is an encryption or decryption context type then the `Encrypt` parameter will determine which operation is being performed. If `Encrypt` is set to CSSM_TRUE then it is an encrypt operation, otherwise it is a decrypt operation. For all other context types the `Encrypt` parameter is ignored. This function can also be used to query the output size requirements for the intermediate steps of a staged cryptographic operation. There may be algorithm-specific and token-specific rules restricting the lengths of data following data update calls.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_QUERY_SIZE_UNKNOWN`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_EncryptData, *CSSM_EncryptDataUpdate*, *CSSM_DecryptData*, *CSSM_DecryptDataUpdate*, *CSSM_SignData*, *CSSM_VerifyData*, *CSSM_DigestData*, *CSSM_GenerateMac*

Functions for the CSP SPI:

CSP_EncryptData, *CSP_EncryptDataUpdate*, *CSP_DecryptData*, *CSP_DecryptDataUpdate*, *CSP_SignData*, *CSP_VerifyData*, *CSP_DigestData*, *CSP_GenerateMac*

RetrieveCounter

NAME

RetrieveCounter: CSSM_RetrieveCounter, CSP_RetrieveCounter – Get the value of a tamper resistant clock (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_RetrieveCounter
(CSSM_CSP_HANDLE CSPHandle,
CSSM_DATA_PTR Counter)
SPI:
CSSM_RETURN CSSMCSPI CSP_RetrieveCounter
(CSSM_CSP_HANDLE CSPHandle,
CSSM_DATA_PTR Counter)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

`Counter` (*output*)

Pointer to CSSM_DATA structure that contains data of the tamper resistant clock/counter of the cryptographic device.

DESCRIPTION

This function returns the value of a tamper resistant clock/counter of the cryptographic device.

NOTES ON SPI

The output is returned to the caller as specified in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

RetrieveUniqueId

NAME

RetrieveUniqueId: CSSM_RetrieveUniqueId, CSP_RetrieveUniqueId – Get identifier (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_RetrieveUniqueId
(CSSM_CSP_HANDLE CSPHandle,
CSSM_DATA_PTR UniqueID)
SPI:
CSSM_RETURN CSSMCSPI CSP_RetrieveUniqueId
(CSSM_CSP_HANDLE CSPHandle,
CSSM_DATA_PTR UniqueID)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

`UniqueID` (*output*)

Pointer to CSSM_DATA structure that contains data that uniquely identifies the cryptographic device.

DESCRIPTION

This function returns an identifier that could be used to uniquely differentiate the cryptographic device from all other devices from the same vendor or different vendors.

NOTES ON SPI

The output is returned to the caller as specified in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

SignData

NAME

SignData: CSSM_SignData, CSP_SignData – Sign all buffer data (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_SignData
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount,
CSSM_ALGORITHMS DigestAlgorithm,
CSSM_DATA_PTR Signature)
SPI:
CSSM_RETURN CSSMCSPI CSP_SignData
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
const CSSM_DATA *DataBufs,
uint32 DataBufCount,
CSSM_ALGORITHMS DigestAlgorithm,
CSSM_DATA_PTR Signature)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`DataBufs` (*input*)

A pointer to a vector of CSSM_DATA structures that contain the data to be signed.

`DataBufCount` (*input*)

The number of DataBufs to be signed.

`DigestAlgorithm` (*input*)

If signing just a digest, specifies the type of digest. In this case, the context should only specify the encryption algorithm. If not signing just a digest, it must be CSSM_ALGID_NONE. In this case, the context should specify the combination digest/encryption algorithm.

`Signature` (*output*)

A pointer to the CSSM_DATA structure for the signature.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DESCRIPTION

This function signs all data contained in the set of input buffers using the private key specified in the context. The CSP can require that the cryptographic context include access credentials for authentication and authorization checks when using a private key or a secret key.

Signing can include digesting the data and encrypting the digest or signing just the digest (already calculated by the application). If digesting the data and encrypting the digest, then the context should specify the combination digest/encryption algorithm (for example, CSSM_ALGID_MD5WithRSA). In this case, the `DigestAlgorithm` parameter must be set to CSSM_ALGID_NONE. If signing just the digest, then the context should specify just the encryption algorithm and the `DigestAlgorithm` parameter should specify the type of digest (for example, CSSM_ALGID_MD5). Also, `DataBufCount` must be 1.

If the signing algorithm is not reversible or strictly limits the size of the signed data, then the algorithm can specify signing without digesting. In this case, the sign operation is performed on the input data and the size of the input data is restricted by the service provider.

NOTES ON API

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, preallocated output buffer, the caller must provide an array of one or more CSSM_DATA structures each, containing a Length field value greater than zero and a non-NULL data pointer field value. To specify automatic output buffer allocation by the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed.

NOTES ON SPI

The output is returned to the caller as specifed in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_OUTPUT_LENGTH_ERROR`
`CSSMERR_CSP_INVALID_DIGEST_ALGORITHM`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_VerifyData, *CSSM_SignDataInit*, *CSSM_SignDataUpdate*, *CSSM_SignDataFinal*

Functions for the CSP SPI:

CSP_VerifyData, *CSP_SignDataInit*, *CSP_SignDataUpdate*, *CSP_SignDataFinal*

SignDataFinal

NAME

SignDataFinal: CSSM_SignDataFinal, CSP_SignDataFinal – Complete the final stage of the sign data (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_SignDataFinal
(CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR Signature)
SPI:
CSSM_RETURN CSSMCSPI CSP_SignDataFinal
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_DATA_PTR Signature)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`Signature` (*output*)

A pointer to the CSSM_DATA structure for the signature.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function completes the final stage of the sign data function.

NOTES ON API

The output is returned to the caller either by filling the caller-specified buffer or by using the application's declared memory allocation functions to allocate buffer space. To specify a specific, preallocated output buffer, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value greater than zero and a non-NULL data pointer field value. To specify automatic output buffer allocation by

the CSP, the caller must provide an array of one or more CSSM_DATA structures, each containing a Length field value equal to zero and a NULL data pointer field value. The application is always responsible for deallocating the memory when it is no longer needed.

NOTES ON SPI

The output is returned to the caller as specifed in Buffer Management for Cryptographic Services.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_OUTPUT_LENGTH_ERROR`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_SignData, CSSM_SignDataInit, CSSM_SignDataUpdate

Functions for the CSP SPI:

CSP_SignData, CSP_SignDataInit, CSP_SignDataUpdate

SignDataInit

NAME

SignDataInit: CSSM_SignDataInit, CSP_SignDataInit – Initialize the staged sign data (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_SignDataInit
(CSSM_CC_HANDLE CCHandle)
SPI:
CSSM_RETURN CSSMCSPI CSP_SignDataInit
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DESCRIPTION

This function initializes the staged sign data function.

For staged operations, a combination operation selecting both a digesting algorithm and a signing algorithm must be specified.

The CSP can require that the cryptographic context include access credentials for authentication and authorization checks when using a private key or a secret key.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_SignData, *CSSM_SignDataUpdate*, *CSSM_SignDataFinal*

Functions for the CSP SPI:

CSP_SignData, *CSP_SignDataUpdate*, *CSP_SignDataFinal*

SignDataUpdate

NAME

SignDataUpdate: CSSM_SignDataUpdate, CSP_SignDataUpdate – Continue the staged signing process input buffer data (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_SignDataUpdate
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount)
SPI:
CSSM_RETURN CSSMCSPI CSP_SignDataUpdate
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`DataBufs` (*input*)

A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

`DataBufCount` (*input*)

The number of DataBufs to be signed.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function continues the staged signing process over all data contained in the set of input buffers. Signing is performed using the private key specified in the context.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_SignData, *CSSM_SignDataInit*, *CSSM_SignDataFinal*

Functions for the CSP SPI:

Funcitons: *CSP_SignData*, *CSP_SignDataInit*, *CSP_SignDataFinal*

TP_ApplyCrlToDb

NAME

TP_ApplyCrlToDb: CSSM_TP_ApplyCrlToDb – Update persistent storage (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_ApplyCrlToDb
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_ENCODED_CRL *CrlToBeApplied,
const CSSM_CERTGROUP *SignerCertGroup,
const CSSM_TP_VERIFY_CONTEXT *ApplyCrlVerifyContext,
CSSM_TP_VERIFY_CONTEXT_RESULT_PTR ApplyCrlVerifyResult)
SPI:
CSSM_RETURN CSSMTPI TP_ApplyCrlToDb
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_ENCODED_CRL *CrlToBeApplied,
const CSSM_CERTGROUP *SignerCertGroup,
const CSSM_TP_VERIFY_CONTEXT *ApplyCrlVerifyContext,
CSSM_TP_VERIFY_CONTEXT_RESULT_PTR ApplyCrlVerifyResult)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the add-in trust policy module used to perform this function.

`CLHandle` (input/optional)

The handle that describes the add-in certificate library module that can be used to manipulate the CRL as it is applied to the data store and to manipulate the certificates effected by the CRL, if required. If no certificate library module is specified, the TP module uses an assumed CL module, if required.

`CSPHandle` (input/optional)

The handle referencing a Cryptographic Service Provider to be used to verify signatures on the CRL determining whether to trust the CRL and apply it to the data store. The TP module is responsible for creating the cryptographic context structures required to perform the verification operation. If no CSP is specified, the TP module uses an assumed CSP to perform these operations. If optional, the caller will set this value to 0.

`CrlToBeApplied` (*input*)

A pointer to a structure containing the encoded certificate revocation list to be applied to the data store. The CRL type and encoding are included in this structure.

`SignerCertGroup` (*input*)

A pointer to the CSSM_CERTGROUP structure containing one or more related certificates that partially or fully represent the signer of the certificate revocation list. The first certificate in the group is the target certificate representing the CRL signer. Use of subsequent certificates is specific to the trust domain. For example, in a hierarchical trust model, subsequent members are intermediate certificates of a certificate chain.

`ApplyCrlVerifyContext` (input/optional)

A structure containing credentials, policy information, and contextual information to be used in the verification process. All of the input values in the context are optional. The service provider can define default values or can attempt to operate without input for all the other fields of this input structure. The operation can fail if a necessary input value is omitted and the service module can not define an appropriate default value.

`ApplyCrlVerifyResult` (output/optional)

A pointer to a structure containing information generated during the verification process. The information can include:

`Evidence`	(output/optional)
`NumberOfEvidences`	(output/optional)

DESCRIPTION

This function updates persistent storage to reflect entries in the certificate revocation list. The TP module determines whether the memory-resident CRL is trusted, and if it should be applied to one or more of the persistent databases. Side effects of this function can include saving a persistent copy of the CRL in a data store, or removing certificate records from a data store.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_CL_HANDLE
CSSMERR_TP_INVALID_CSP_HANDLE
CSSMERR_TP_INVALID_CRL_TYPE
CSSMERR_TP_INVALID_CRL_ENCODING
CSSMERR_TP_INVALID_CRL_POINTER
CSSMERR_TP_INVALID_CRL
CSSMERR_TP_INVALID_CERTGROUP_POINTER
CSSMERR_TP_INVALID_CERTGROUP
CSSMERR_TP_INVALID_CERTIFICATE
CSSMERR_TP_INVALID_ACTION
CSSMERR_TP_INVALID_ACTION_DATA
```

```
CSSMERR_TP_VERIFY_ACTION_FAILED
CSSMERR_TP_INVALID_CRLGROUP_POINTER
CSSMERR_TP_INVALID_CRLGROUP
CSSMERR_TP_INVALID_CRL_AUTHORITY
CSSMERR_TP_INVALID_CALLERAUTH_CONTEXT_POINTER
CSSMERR_TP_INVALID_POLICY_IDENTIFIERS
CSSMERR_TP_INVALID_TIMESTRING
CSSMERR_TP_INVALID_STOP_ON_POLICY
CSSMERR_TP_INVALID_CALLBACK
CSSMERR_TP_INVALID_ANCHOR_CERT
CSSMERR_TP_CERTGROUP_INCOMPLETE
CSSMERR_TP_INVALID_DL_HANDLE
CSSMERR_TP_INVALID_DB_HANDLE
CSSMERR_TP_INVALID_DB_LIST_POINTER
CSSMERR_TP_INVALID_DB_LIST
CSSMERR_TP_AUTHENTICATION_FAILED
CSSMERR_TP_INSUFFICIENT_CREDENTIALS
CSSMERR_TP_NOT_TRUSTED
CSSMERR_TP_CERT_REVOKED
CSSMERR_TP_CERT_SUSPENDED
CSSMERR_TP_CERT_EXPIRED
CSSMERR_TP_CERT_NOT_VALID_YET
CSSMERR_TP_INVALID_CERT_AUTHORITY
CSSMERR_TP_INVALID_SIGNATURE
CSSMERR_TP_INVALID_NAME
CSSMERR_TP_CERTIFICATE_CANT_OPERATE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlGetFirstItem, *CSSM_CL_CrlGetNextItem*, *CSSM_DL_CertRevoke*

Functions for the TP SPI:

CL_CrlGetFirstItem, *CL_CrlGetNextItem*, *DL_CertRevoke*

TP_CertCreateTemplate

NAME

TP_CertCreateTemplate: CSSM_TP_CertCreateTemplate – Allocate and initialize template memory (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_CertCreateTemplate
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
uint32 NumberOfFields,
const CSSM_FIELD *CertFields,
CSSM_DATA_PTR CertTemplate)
SPI:
CSSM_RETURN CSSMTPI TP_CertCreateTemplate
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
uint32 NumberOfFields,
const CSSM_FIELD *CertFields,
CSSM_DATA_PTR CertTemplate)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the add-in trust policy module used to perform this function.

`CLHandle` (*input*)

The handle that describes the certificate library module used to perform this function.

`NumberOfFields` (*input*)

The number of certificate field values specified in the `CertFields`.

`CertFields` (*input*)

A pointer to an array of OID/value pairs that identifies the field values to initialize a new certificate.

`CertTemplate` (*output*)

A pointer to a CSSM_DATA structure that will contain the unsigned certificate template as a result of this function.

DESCRIPTION

This function allocates and initializes memory for an encoded certificate template output in `CertTemplate->Data`. The template values are specified by the input OID/value pairs contained in `CertFields`. The initialization process includes encoding all certificate field values according to the certificate type and certificate template encoding supported by the trust policy library module. The `CertTemplate` output is an unsigned certificate template in the format supported by the TP.

The memory for `CertTemplate->Data` is allocated by the service provider using the calling application's memory management routines. The application must deallocate the memory.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_CL_HANDLE
CSSMERR_TP_INVALID_FIELD_POINTER
CSSMERR_TP_UNKNOWN_TAG
CSSMERR_TP_INVALID_NUMBER_OF_FIELDS
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_CertGetAllTemplateFields, CSSM_TP_CertSign

Functions for the TP SPI:

TP_CertGetAllTemplateFields, TP_CertSign

TP_CertGetAllTemplateFields

NAME

TP_CertGetAllTemplateFields: CSSM_TP_CertGetAllTemplateFields – Get CertTemplate field values (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_CertGetAllTemplateFields
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *CertTemplate,
uint32 *NumberOfFields,
CSSM_FIELD_PTR *CertFields)
SPI:
CSSM_RETURN CSSMTPI TP_CertGetAllTemplateFields
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
const CSSM_DATA *CertTemplate,
uint32 *NumberOfFields,
CSSM_FIELD_PTR *CertFields)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the add-in trust policy module used to perform this function.

`CLHandle` (*input*)

The handle that describes the certificate library module used to perform this function.

`CertTemplate` (*input*)

A pointer to the CSSM_DATA structure containing the packed, encoded certificate template.

`NumberOfFields` (*output*)

The length of the output array of fields.

`CertFields` (*output*)

A pointer to an array of CSSM_FIELD structures which contains the OIDs and values of the fields of the input certificate template.

DESCRIPTION

This function extracts and returns all field values from `CertTemplate`. The `CertTemplate` parameter is an unsigned certificate template in the format supported by the TP. Fields are returned as a set of OID-value pairs. The OID identifies the TP certificate template field and the data format of the value extracted from that field. The Trust Policy module defines and uses a preferred data format for returning field values from this function. Memory for the `CertFields` output is allocated by the service provider using the calling application's memory management routines. The application must deallocate the memory, by calling `CSSM_CL_FreeFields()` (CSSM API), or `CL_FreeFields()` (TP SPI).

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_TP_INVALID_CL_HANDLE`
`CSSMERR_TP_INVALID_FIELD_POINTER`
`CSSMERR_TP_UNKNOWN_FORMAT`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_CertCreateTemplate, CSSM_TP_CertSign

Functions for the TP SPI:

TP_CertCreateTemplate, TP_CertSign

TP_CertGroupConstruct

NAME

TP_CertGroupConstruct: CSSM_TP_CertGroupConstruct – Construct credential (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_TP_CertGroupConstruct
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_DL_DB_LIST *DBList,
const void *ConstructParams,
const CSSM_CERTGROUP *CertGroupFrag,
CSSM_CERTGROUP_PTR *CertGroup)
SPI:
CSSM_RETURN CSSMTPI TP_CertGroupConstruct
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_DL_DB_LIST *DBList,
const void *ConstructParams,
const CSSM_CERTGROUP *CertGroupFrag,
CSSM_CERTGROUP_PTR *CertGroup)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle to the trust policy module to perform this operation.

`CLHandle` (input/optional)

The handle to the certificate library module that can be used to manipulate and parse values in stored in the certgroup certificates. If no certificate library module is specified, the TP module uses an assumed CL module.

`CSPHandle` (input./optional)

A handle specifying the Cryptographic Service Provider to be used to verify certificates as the certificate group is constructed. If the a CSP handle is not specified, the trust policy module can assume a default CSP. If the module cannot assume a default, or the default CSP is not available on the local system, an error occurs.

`DBList` (*input*)

A list of handle pairs specifying a data storage library module and a data store, identifying certificate databases containing certificates (and possibly other security objects) that are managed by that module. certificates (and possibly other security objects). The data stores should be searched to complete construction of a semantically-related certificate group.

`ConstructParams` (input/optional)

A pointer to data that can be used by the add-in trust policy module in constructing the `CertGroup`. The semantics of this parameter are defined by the trust policy and the credential model supported by that policy. The input parameter can consist of a set of values, each guiding some aspect of the construction process. Parameter values can:

- Limit the certificates that are added to the constructed set.
- Identify other sources of certificates for inclusion in the constructed set.

`CertGroupFrag` (*input*)

A list of certificates that form a possibly incomplete set of certificates. The first certificate in the group represents the target certificate for which a group of semantically related certificates will be assembled. Subsequent intermediate certificates can be supplied by the caller. They need not be in any particular order.

`CertGroup` (*output*)

A pointer to a complete certificate group based on the original subset of certificates and the certificate data stores. The CSSM_CERTGROUP and its sub-structure is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function builds a collection of certificates that together make up a meaningful credential for a given trust domain. For example, in a hierarchical trust domain, a certificate group is a chain of certificates from an end entity to a top level certification authority. The constructed certificate group format (such as ordering) is implementation specific. However, the subject or end-entity is always the first certificate in the group.

A partially constructed certificate group is specified in `CertGroupFrag`. The first certificate is interpreted to be the subject or end-entity certificate. Subsequent certificates in the `CertGroupFrag` structure may be used during the construction of a certificate group in conjunction with certificates found in the data stores specified in `DBList`. The trust policy defines the certificates that will be included in the resulting set.

The output set is a sequence of certificates ordered by the relationship among them. The result set can be augmented by adding semantically-related certificates obtained by searching the certificate data stores specified in `DBList`. The data stores are searched in order of appearance in `DBList`. If the TP supports a hierarchical model of certificates, the function output is an uninterrupted, ordered chain of certificates based on the first certificate as the leaf of the certificate chain. If the certificate is multiply-signed, then the ordered chain will follow the first signing certificate. The function should also detect cross-certificate pairs and should include both certificates without duplicating either certificate.

Extraneous certificates in the `CertGroupFrag` fragment or contained in the `DBList` data stores are ignored. The certificate group returned by this function can be used as input to the function `CSSM_TP_CertGroupVerify()` (CSSM API), or `TP_CertGroupVerify()` (TP SPI).

The constructed certificate group can be consistent locally or globally. Consistency can be limited to the local system if locally-defined points of trust are inserted into the group.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_CL_HANDLE
CSSMERR_TP_INVALID_CSP_HANDLE
CSSMERR_TP_INVALID_DL_HANDLE
CSSMERR_TP_INVALID_DB_HANDLE
CSSMERR_TP_INVALID_DB_LIST_POINTER
CSSMERR_TP_INVALID_DB_LIST
CSSMERR_TP_INVALID_CERTGROUP_POINTER
CSSMERR_TP_INVALID_CERTGROUP
CSSMERR_TP_INVALID_CERTIFICATE
CSSMERR_TP_CERTGROUP_INCOMPLETE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_CertGroupPrune, *CSSM_TP_CertGroupVerify*

Functions for the TP SPI:

TP_CertGroupPrune, *TP_CertGroupVerify*

TP_CertGroupPrune

NAME

TP_CertGroupPrune: CSSM_TP_CertGroupPrune – Remove locally issued anchor certificates (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_CertGroupPrune
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
const CSSM_DL_DB_LIST *DBList,
const CSSM_CERTGROUP *OrderedCertGroup,
CSSM_CERTGROUP_PTR *PrunedCertGroup)
SPI:
CSSM_RETURN CSSMTPI TP_CertGroupPrune
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
const CSSM_DL_DB_LIST *DBList,
const CSSM_CERTGROUP *OrderedCertGroup,
CSSM_CERTGROUP_PTR *PrunedCertGroup)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle to the trust policy module to perform this operation.

`CLHandle` (input/optional)

The handle to the certificate library module that can be used to manipulate and parse the certgroup certificates and the certificates in the specified data stores. If no certificate library module is specified, the TP module uses an assumed CL module.

`DBList` (*input*)

A list of handle pairs specifying a data storage library module and a data store, identifying certificate databases containing certificates (and possibly other security objects) that are managed by that module. The data stores are searched for anchor certificates restricted to have local scope. These certificates are candidates for removal from the subject certificate group.

`OrderedCertGroup` (*input*)

The initial complete set of semantically-related certificates - for example, the result of a `CSSM_TP_CertGroupConstruct()` (CSSM API), or `TP_CertGroupConstruct()` (TP SPI), call - from which certificates will be selectively removed.

`PrunedCertGroup` (*output*)

A pointer to a certificate group containing those certificates which are verifiable credentials outside of the local system. The CSSM_CERTGROUP and its substructure is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function removes any locally issued anchor certificates from a constructed certificate group. The prune operation can remove those certificates that have been signed by any local certificate authority, as it is possible that these certificates will not be meaningful on other systems.

This operation can also remove additional certificates that can be added to the certificate group again using the `CSSM_TP_CertGroupConstruct()` (CSSM API), or `TP_CertGroupConstruct()` (TP SPI), operation. The pruned certificate group should be suitable for export to external hosts/entities, which can in turn reconstruct and verify the certificate group.

The `DBList` parameter specifies a set of data stores containing certificates that should be pruned from the group.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_CL_HANDLE
CSSMERR_TP_INVALID_DL_HANDLE
CSSMERR_TP_INVALID_DB_HANDLE
CSSMERR_TP_INVALID_DB_LIST_POINTER
CSSMERR_TP_INVALID_DB_LIST
CSSMERR_TP_INVALID_CERTGROUP_POINTER
CSSMERR_TP_INVALID_CERTGROUP
CSSMERR_TP_INVALID_CERTIFICATE
CSSMERR_TP_CERTGROUP_INCOMPLETE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_CertGroupConstruct, *CSSM_TP_CertGroupVerify*

Functions for the TP SPI:

TP_CertGroupConstruct, *TP_CertGroupVerify*

TP_CertGroupToTupleGroup

NAME

TP_CertGroupToTupleGroup: CSSM_TP_CertGroupToTupleGroup – Create a set of authorization tuples (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_CertGroupToTupleGroup
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE  CLHandle,
const CSSM_CERTGROUP *CertGroup,
CSSM_TUPLEGROUP_PTR *TupleGroup)
SPI:
CSSM_RETURN CSSMTPI TP_CertGroupToTupleGroup
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE  CLHandle,
const CSSM_CERTGROUP *CertGroup,
CSSM_TUPLEGROUP_PTR *TupleGroup)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the trust policy service module used to perform this function.

`CLHandle` (input/optional)

The handle that describes the certificate library module that can be used to scan the certificate fields for values. If no certificate library module is specified, the TP module uses an assumed CL module.

`CertGroup` (*input*)

A group of certificates in the native certificate format supported by the Trust Policy module. The certificates carry authorizations for one or more certificate subjects.

`TupleGroup` (*output*)

A pointer to a structure containing references to one or more tuples resulting from the translation process. Storage for structure and the tuples is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function creates a set of authorization tuples based on a set of input certificates. The certificates must be of the type managed by the Trust Policy module. The trust policy module may require that the input certificates be successfully verified before being translated to tuples. It is assumed that the certificates carry authorizations. The trust policy service provider interprets the certificate authorization fields and generates

one or more tuples corresponding to those authorizations. The certificates of the type managed by the Trust Policy module. The resulting tuples can be input to an authorization evaluation function, such as `CSSM_AC_AuthCompute()` (CSSM API), or `AC_AuthCompute()` (AC SPI), which determines whether a particular action is authorized under a basic set of authorization assumptions.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_TP_INVALID_CL_HANDLE`
`CSSMERR_TP_INVALID_CERTGROUP_POINTER`
`CSSMERR_TP_INVALID_CERTGROUP`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_TupleGroupToCertGroup, *CSSM_AC_AuthCompute*

Functions for the TP SPI:

TP_TupleGroupToCertGroup, *AC_AuthCompute*

TP_CertGroupVerify

NAME

TP_CertGroupVerify: CSSM_TP_CertGroupVerify – Determine if a certificate is trusted (CDSA)

SYNOPSIS

```
# include <cssm.h>

API:
CSSM_RETURN CSSMAPI CSSM_TP_CertGroupVerify
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_CERTGROUP *CertGroupToBeVerified,
const CSSM_TP_VERIFY_CONTEXT *VerifyContext,
CSSM_TP_VERIFY_CONTEXT_RESULT_PTR VerifyContextResult)
SPI:
CSSM_RETURN CSSMTPI TP_CertGroupVerify
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_CERTGROUP *CertGroupToBeVerified,
const CSSM_TP_VERIFY_CONTEXT *VerifyContext,
CSSM_TP_VERIFY_CONTEXT_RESULT_PTR VerifyContextResult)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the add-in trust policy module used to perform this function.

`CLHandle` (input/optional)

The handle that describes the add-in certificate library module that can be used to manipulate the subject certificate and anchor certificates. If no certificate library module is specified, the TP module uses an assumed CL module, if required.

`CSPHandle` (input/optional)

The handle that describes the add-in Cryptographic Service Provider module that can be used to perform the cryptographic operations required to carry out the verification. If no CSP handle is specified, the TP module allocates a suitable CSP.

`CertGroupToBeVerified` (*input*)

A group of one or more certificates to be verified. The first certificate in the group is the primary target certificate for verification. Use of the subsequent certificates during the verification process is specific to the trust domain.

`VerifyContext` (input/optional)

A structure containing credentials, policy information, and contextual information to be used in the verification process. All of the input values in the context are optional except `Action`. The service provider can define default values or can attempt to operate without input for all the other fields of this input structure. The operation can fail if a necessary input value is omitted and the service module can not define an appropriate default value.

`VerifyContextResult` (output/optional)

A pointer to a structure containing information generated during the verification process. The information can include:

`Evidence`	(output/optional)
`NumberOfEvidences`	(output/optional)

DESCRIPTION

This function determines whether the certificate is trusted. The actions performed by this function differ based on the trust policy domain. The factors include practices, procedures and policies defined by the certificate issuer.

Typically certificate verification involves the verification of multiple certificates. The first certificate in the group is the target of the verification process. The other certificates in the group are used in the verification process to connect the target certificate with one or more anchors of trust. The supporting certificates can be contained in the provided certificate group or can be stored in the data stores specified in the `VerifyContext` DBList. This allows the trust policy module to construct a certificate group and perform verification in one operation. The data stores specified by DBList can also contain certificate revocation lists used in the verification process. It is also possible to provide a data store of anchor certificates. Typically the points of Trust are few in number and are embedded in the caller or in the TPM during software manufacturing or at runtime

The caller can select to be notified incrementally as each certificate is verified. The `CallbackWithVerifiedCert` parameter (in the `VerifyContext`) can specify a caller function to be invoked at the end of each certificate verification, returning the verified certificate for use by the caller.

Anchor certificates are a list of implicitly trusted certificates. These include root certificates, cross certified certificates, and locally defined sources of trust. These certificates form the basis to determine trust in the subject certificate.

A policy identifier can specify an additional set of conditions that must be satisfied by the subject certificate in order to meet the trust criteria. The name space for policy identifiers is defined by the application domains to which the policy applies. This is outside of CSSM. A list of policy identifiers can be specified and the stopping condition for evaluating that set of conditions.

The evaluation and verification process can produce a list of evidence. The evidence can be selected values from the certificates examined in the verification process, entire certificates from the process or other pertinent information that forms an audit trail of the verification process. This evidence is returned to the caller after all steps in the verification process have been completed.

If verification succeeds, the trust policy module may carry out the action on the specified data or may return approval for the action requiring the caller to perform the action. The caller must consult TP module documentation outside of this specification to determine all module-specific side effects of this operation.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_TP_INVALID_CL_HANDLE
CSSMERR_TP_INVALID_CSP_HANDLE
CSSMERR_TP_INVALID_CERTGROUP_POINTER
CSSMERR_TP_INVALID_CERTGROUP
CSSMERR_TP_INVALID_CERTIFICATE
CSSMERR_TP_INVALID_ACTION
CSSMERR_TP_INVALID_ACTION_DATA
CSSMERR_TP_VERIFY_ACTION_FAILED
CSSMERR_TP_INVALID_CRLGROUP_POINTER
CSSMERR_TP_INVALID_CRLGROUP
CSSMERR_TP_INVALID_CRL_AUTHORITY
CSSMERR_TP_INVALID_CALLERAUTH_CONTEXT_POINTER
CSSMERR_TP_INVALID_POLICY_IDENTIFIERS
CSSMERR_TP_INVALID_TIMESTRING
CSSMERR_TP_INVALID_STOP_ON_POLICY
CSSMERR_TP_INVALID_CALLBACK
CSSMERR_TP_INVALID_ANCHOR_CERT
CSSMERR_TP_CERTGROUP_INCOMPLETE
CSSMERR_TP_INVALID_DL_HANDLE
CSSMERR_TP_INVALID_DB_HANDLE
CSSMERR_TP_INVALID_DB_LIST_POINTER
CSSMERR_TP_INVALID_DB_LIST
CSSMERR_TP_AUTHENTICATION_FAILED
CSSMERR_TP_INSUFFICIENT_CREDENTIALS
CSSMERR_TP_NOT_TRUSTED
CSSMERR_TP_CERT_REVOKED
CSSMERR_TP_CERT_SUSPENDED
CSSMERR_TP_CERT_EXPIRED
CSSMERR_TP_CERT_NOT_VALID_YET
CSSMERR_TP_INVALID_CERT_AUTHORITY
CSSMERR_TP_INVALID_SIGNATURE
CSSMERR_TP_INVALID_NAME

SEE ALSO

Books

Intel CDSA Application Developer's Guide

TP_CertReclaimAbort

NAME

TP_CertReclaimAbort: CSSM_TP_CertReclaimAbort – Terminate the process of reclaiming certificates (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_CertReclaimAbort
(CSSM_TP_HANDLE TPHandle,
CSSM_LONG_HANDLE KeyCacheHandle)
SPI:
CSSM_RETURN CSSMTPI TP_CertReclaimAbort
(CSSM_TP_HANDLE TPHandle,
CSSM_LONG_HANDLE KeyCacheHandle)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the service provider module used to perform this function.

`KeyCacheHandle` (*input*)

An opaque handle that identifies the cache of protected private keys reclaimed from a certificate authority for potentially recovery on the local system.

DESCRIPTION

This function terminates the iterative process of reclaiming certificates and recovering their associated private keys from a protected key cache. This function must be called even if all private keys are recovered from the cache. This function destroys all intermediate state and secret information used during the reclamation process. At completion of this function, the cache handle is invalid.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_TP_INVALID_KEYCACHE_HANDLE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_CertReclaimKey

Functions for the TP SPI:

TP_CertReclaimKey

TP_CertReclaimKey

NAME

TP_CertReclaimKey: CSSM_TP_CertReclaimKey – Get private key associated with a certificate (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_CertReclaimKey
(CSSM_TP_HANDLE TPHandle,
const CSSM_CERTGROUP *CertGroup,
uint32 CertIndex,
CSSM_LONG_HANDLE KeyCacheHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry)
SPI:
CSSM_RETURN CSSMTPI TP_CertReclaimKey
(CSSM_TP_HANDLE TPHandle,
const CSSM_CERTGROUP *CertGroup,
uint32 CertIndex,
CSSM_LONG_HANDLE KeyCacheHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the service provider module used to perform this operation.

`CertGroup` (*input*)

A pointer to a structure containing a reference to a group of certificates and the number of certificates contained in that group. The certificate group contains all certificates that are candidates for reclamation.

`CertIndex` (*input*)

An index value that identifies the certificate whose associated private key is to be recovered and stored in the local CSP. This index value I references the I-th certificate in `CertGroup`.

`KeyCacheHandle` (*input*)

A reference handle that uniquely identifies the cache of protected private keys associated with the reclaimed certificates contained in `CertGroup`. The structure of the cache is opaque to the caller.

`CSPHandle` (*input*)

The handle that describes the CSP module where the private key is to be stored. Optionally, the CA service provider can use this CSP to perform additional cryptographic operations or may use another default CSP for that purpose.

`CredAndAclEntry` (input/optional)

A structure containing one or more credentials authorized for creating a key and the prototype ACL entry that will control future use of the newly created key. The credentials and ACL entry prototype can be presented as immediate values or callback functions can be provided for use by the CSP to acquire the credentials and/or the ACL entry interactively. If the CSP provides public access for creating a key, then the credentials can be NULL. If the CSP defines a default initial ACL entry for the new key, then the ACL entry prototype can be an empty list.

DESCRIPTION

This function recovers the private key associated with a certificate and securely stores that key in the specified Cryptographic Service Provider. The key and its associated certificate are among a set of certificates and private keys reclaimed from a certificate authority.

The particular private key to be recovered to the local system is identified by its associated certificate. The certificate is identified by its `CertIndex` position within the `CertGroup`.

The reclamation process associates the private key with the public key contained in the certificate, and securely stores the private key in the specified Cryptographic Service Provider. The CSP can require that the caller provide access credentials authorizing inserting a new key into the CSP through an `UnwrapKey` operation. The caller should also provide an initial Access Control List (ACL) entry for the newly inserted key. The ACL entry is used to control future use of the recovered private key. These inputs are provided in `CredAndAclEntry`.

When all required private keys have been reclaimed, the key cache can be discarded using the function `CSSM_TP_CertReclaimAbort()` (CSSM API), or `TP_CertReclaimAbort()` (TP SPI). The caller must free the `CertGroup` when it is no longer needed.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_CERTGROUP_POINTER
CSSMERR_TP_INVALID_CERTGROUP
CSSMERR_TP_INVALID_CERTIFICATE
CSSMERR_TP_INVALID_INDEX
CSSMERR_TP_INVALID_KEYCACHE_HANDLE
CSSMERR_TP_INVALID_CSP_HANDLE
CSSMERR_TP_AUTHENTICATION_FAILED
CSSMERR_TP_INSUFFICIENT_CREDENTIALS
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_RetrieveCredResult, *CSSM_TP_Cert_ReclaimAbort*

Functions for the TP SPI:

TP_RetrieveCredResult, *TP_Cert_ReclaimAbort*

TP_CertRemoveFromCrlTemplate

NAME

TP_CertRemoveFromCrlTemplate: CSSM_TP_CertRemoveFromCrlTemplate – Determine if the revoking certificate group can remove the subject certificate group from the CRL template (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_CertRemoveFromCrlTemplate
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_DATA *OldCrlTemplate,
const CSSM_CERTGROUP *CertGroupToBeRemoved,
const CSSM_CERTGROUP *RevokerCertGroup,
const CSSM_TP_VERIFY_CONTEXT *RevokerVerifyContext,
CSSM_TP_VERIFY_CONTEXT_RESULT_PTR RevokerVerifyResult,
CSSM_DATA_PTR NewCrlTemplate)
SPI:
CSSM_RETURN CSSMTPI TP_CertRemoveFromCrlTemplate
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_DATA *OldCrlTemplate,
const CSSM_CERTGROUP *CertGroupToBeRemoved,
const CSSM_CERTGROUP *RevokerCertGroup,
const CSSM_TP_VERIFY_CONTEXT *RevokerVerifyContext,
CSSM_TP_VERIFY_CONTEXT_RESULT_PTR RevokerVerifyResult,
CSSM_DATA_PTR NewCrlTemplate)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the add-in trust policy module used to perform this function.

`CLHandle` (input/optional)

The handle that describes the add-in certificate library module used to perform this function.

`CSPHandle` (input/optional)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function.

`OldCrlTemplate` (input/optional)

A pointer to the CSSM_DATA structure containing an existing certificate revocation list. If this input is NULL, a new list is created or the operation fails.

`CertGroupToBeRemoved` (*input*)

A group of one or more certificates to be removed from the the CRL template.

`RevokerCertGroup` (*input*)

A group of one or more certificates that partially or fully represent the revoking entity for this operation. The first certificate in the group is the target certificate representing the revoker. The use of subsequent certificates is specific to the trust domain.

`RevokerVerifyContext` (*input*)

A structure containing policy elements useful in verifying certificates and their use with respect to a security policy. Optional elements in the verify context left unspecified will cause the internal default values to be used. Default values are specified in the TP module vendor release documents. This context is used to verify the revoker certificate group.

`RevokerVerifyResult` (output/optional)

A pointer to a structure containing information generated during the verification process. The information can include:

`Evidence`	(output/optional)
`NumberOfEvidences`	(output/optional)

`NewCrlTemplate` (*output*)

A pointer to the CSSM_DATA structure containing the updated certificate revocation list. If the pointer is NULL, an error has occurred.

DESCRIPTION

The TP module determines whether the revoking certificate group can remove the subject certificate group from the CRL template. The revoker certificate group is first authenticated and its applicability to perform this operation is determined. Once the trust is established, the TP removes the certificates from the CRL template.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_CL_HANDLE
CSSMERR_TP_INVALID_CSP_HANDLE
CSSMERR_TP_INVALID_CRL_POINTER
CSSMERR_TP_INVALID_CRL
CSSMERR_TP_UNKNOWN_FORMAT
CSSMERR_TP_CRL_ALREADY_SIGNED
CSSMERR_TP_INVALID_CERTGROUP_POINTER
CSSMERR_TP_INVALID_CERTGROUP
```

```
CSSMERR_TP_INVALID_CERTIFICATE
CSSMERR_TP_INVALID_ACTION
CSSMERR_TP_INVALID_ACTION_DATA
CSSMERR_TP_VERIFY_ACTION_FAILED
CSSMERR_TP_INVALID_CRLGROUP_POINTER
CSSMERR_TP_INVALID_CRLGROUP
CSSMERR_TP_INVALID_CRL_AUTHORITY
CSSMERR_TP_INVALID_CALLERAUTH_CONTEXT_POINTER
CSSMERR_TP_INVALID_POLICY_IDENTIFIERS
CSSMERR_TP_INVALID_TIMESTRING
CSSMERR_TP_INVALID_STOP_ON_POLICY
CSSMERR_TP_INVALID_CALLBACK
CSSMERR_TP_INVALID_ANCHOR_CERT
CSSMERR_TP_CERTGROUP_INCOMPLETE
CSSMERR_TP_INVALID_DL_HANDLE
CSSMERR_TP_INVALID_DB_HANDLE
CSSMERR_TP_INVALID_DB_LIST_POINTER
CSSMERR_TP_INVALID_DB_LIST
CSSMERR_TP_AUTHENTICATION_FAILED
CSSMERR_TP_INSUFFICIENT_CREDENTIALS
CSSMERR_TP_NOT_TRUSTED
CSSMERR_TP_CERT_REVOKED
CSSMERR_TP_CERT_SUSPENDED
CSSMERR_TP_CERT_EXPIRED
CSSMERR_TP_CERT_NOT_VALID_YET
CSSMERR_TP_INVALID_CERT_AUTHORITY
CSSMERR_TP_INVALID_SIGNATURE
CSSMERR_TP_INVALID_NAME
CSSMERR_TP_CERTIFICATE_CANT_OPERATE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlAddCert

Functions for the TP SPI:

CL_CrlAddCert

TP_CertRevoke

NAME

TP_CertRevoke: CSSM_TP_CertRevoke – Determine if the revoking certificate group can revoke the subject certificate group (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_CertRevoke
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_DATA *OldCrlTemplate,
const CSSM_CERTGROUP *CertGroupToBeRevoked,
const CSSM_CERTGROUP *RevokerCertGroup,
const CSSM_TP_VERIFY_CONTEXT *RevokerVerifyContext,
CSSM_TP_VERIFY_CONTEXT_RESULT_PTR RevokerVerifyResult,
CSSM_TP_CERTCHANGE_REASON Reason,
CSSM_DATA_PTR NewCrlTemplate)
SPI:
CSSM_RETURN CSSMTPI TP_CertRevoke
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_DATA *OldCrlTemplate,
const CSSM_CERTGROUP *CertGroupToBeRevoked,
const CSSM_CERTGROUP *RevokerCertGroup,
const CSSM_TP_VERIFY_CONTEXT *RevokerVerifyContext,
CSSM_TP_VERIFY_CONTEXT_RESULT_PTR RevokerVerifyResult,
CSSM_TP_CERTCHANGE_REASON Reason,
CSSM_DATA_PTR NewCrlTemplate)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the add-in trust policy module used to perform this function.

`CLHandle` (input/optional)

The handle that describes the add-in certificate library module used to perform this function.

`CSPHandle` (input/optional)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function.

`OldCrlTemplate` (input/optional)

A pointer to the CSSM_DATA structure containing an existing certificate revocation list. If this input is NULL, a new list is created or the operation fails.

`CertGroupToBeRevoked` (*input*)

A group of one or more certificates that partially or fully represent the certificate to be revoked by this operation. The first certificate in the group is the target certificate. The use of subsequent certificates is specific to the trust domain. For example, in a hierarchical trust model subsequent members are intermediate certificates of a certificate chain.

`RevokerCertGroup` (*input*)

A group of one or more certificates that partially or fully represent the revoking entity for this operation. The first certificate in the group is the target certificate representing the revoker. The use of subsequent certificates is specific to the trust domain.

`RevokerVerifyContext` (*input*)

A structure containing policy elements useful in verifying certificates and their use with respect to a security policy. Optional elements in the verify context left unspecified will cause the internal default values to be used. Default values are specified in the TP module vendor release documents. This context is used to verify the revoker certificate group.

`RevokerVerifyResult` (output/optional)

A pointer to a structure containing information generated during the verification process. The information can include:

`Evidence`	(output/optional)
`NumberOfEvidences`	(output/optional)

`Reason` (input/optional)

The reason for revoking the subject certificate.

`NewCrlTemplate` (output/optional)

A pointer to the CSSM_DATA structure containing the updated certificate revocation list. If the pointer is NULL, an error has occurred.

DESCRIPTION

The TP module determines whether the revoking certificate group can revoke the subject certificate group. The revoker certificate group is first authenticated and its applicability to perform this operation is determined. Once the trust is established, the TP revokes the subject certificate by adding it to the certificate revocation list.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_CL_HANDLE
CSSMERR_TP_INVALID_CSP_HANDLE
CSSMERR_TP_INVALID_CRL_POINTER
CSSMERR_TP_INVALID_CRL
CSSMERR_TP_UNKNOWN_FORMAT
CSSMERR_TP_CRL_ALREADY_SIGNED
CSSMERR_TP_INVALID_CERTGROUP_POINTER
CSSMERR_TP_INVALID_CERTGROUP
CSSMERR_TP_INVALID_CERTIFICATE
CSSMERR_TP_INVALID_ACTION
CSSMERR_TP_INVALID_ACTION_DATA
CSSMERR_TP_VERIFY_ACTION_FAILED
CSSMERR_TP_INVALID_CRLGROUP_POINTER
CSSMERR_TP_INVALID_CRLGROUP
CSSMERR_TP_INVALID_CRL_AUTHORITY
CSSMERR_TP_INVALID_CALLERAUTH_CONTEXT_POINTER
CSSMERR_TP_INVALID_POLICY_IDENTIFIERS
CSSMERR_TP_INVALID_TIMESTRING
CSSMERR_TP_INVALID_STOP_ON_POLICY
CSSMERR_TP_INVALID_CALLBACK
CSSMERR_TP_INVALID_ANCHOR_CERT
CSSMERR_TP_CERTGROUP_INCOMPLETE
CSSMERR_TP_INVALID_DL_HANDLE
CSSMERR_TP_INVALID_DB_HANDLE
CSSMERR_TP_INVALID_DB_LIST_POINTER
CSSMERR_TP_INVALID_DB_LIST
CSSMERR_TP_AUTHENTICATION_FAILED
CSSMERR_TP_INSUFFICIENT_CREDENTIALS
CSSMERR_TP_NOT_TRUSTED
CSSMERR_TP_CERT_REVOKED
CSSMERR_TP_CERT_SUSPENDED
CSSMERR_TP_CERT_EXPIRED
CSSMERR_TP_CERT_NOT_VALID_YET
CSSMERR_TP_INVALID_CERT_AUTHORITY
CSSMERR_TP_INVALID_SIGNATURE
CSSMERR_TP_INVALID_NAME
CSSMERR_TP_CERTIFICATE_CANT_OPERATE
CSSMERR_TP_INVALID_REASON
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlAddCert

Functions for the TP SPI:

CL_CrlAddCert

TP_CertSign

NAME

TP_CertSign: CSSM_TP_CertSign – Determine if signer certificate is trusted (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_CertSign
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CertTemplateToBeSigned,
const CSSM_CERTGROUP *SignerCertGroup,
const CSSM_TP_VERIFY_CONTEXT *SignerVerifyContext,
CSSM_TP_VERIFY_CONTEXT_RESULT_PTR SignerVerifyResult,
CSSM_DATA_PTR SignedCert)
SPI:
CSSM_RETURN CSSMTPI TP_CertSign
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *CertTemplateToBeSigned,
const CSSM_CERTGROUP *SignerCertGroup,
const CSSM_TP_VERIFY_CONTEXT *SignerVerifyContext,
CSSM_TP_VERIFY_CONTEXT_RESULT_PTR SignerVerifyResult,
CSSM_DATA_PTR SignedCert)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the add-in trust policy module used to perform this function.

`CLHandle` (input/optional)

The handle that describes the add-in certificate library module used to perform this function.

`CCHandle` (input/optional)

The handle that describes the cryptographic context for signing the certificate. This context also identifies the Cryptographic Service Provider to be used to perform the signing operation. If this handle is not provided by the caller, the trust policy module can assume a default signing algorithm and a default CSP. If the trust policy module does not assume defaults or the default CSP is not available on the local system, an error occurs.

`CertTemplateToBeSigned` (*input*)

A pointer to a structure containing a certificte template to be signed. The CRL type and encoded are included in this structure.

`SignerCertGroup` (*input*)

A group of one or more certificates that partially or fully represent the signer for this operation. The first certificate in the group is the target certificate representing the signer. Use of subsequent certificates is specific to the trust domain. For example, in a hierarchical trust model subsequent members are intermediate certificates of a certificate chain.

`SignerVerifyContext` (input/optional)

A structure containing credentials, policy information, and contextual information to be used in the verification process. All of the input values in the context are optional. The service provider can define default values or can attempt to operate without input for all the other fields of this input structure. The operation can fail if a necessary input value is omitted and the service module can not define an appropriate default value.

`SignerVerifyResult` (output/optional)

A pointer to a structure containing information generated during the verification process. The information can include:

`Evidence`	(output/optional)
`NumberOfEvidences`	(output/optional)

`SignedCert` (*output*)

A pointer to the CSSM_DATA structure containing the signed certificate. The `SignedCert->Data` is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

The TP module decides whether the signer certificate is trusted to sign the `CertTemplateToBeSigned`. The signer certificate group is first authenticated and its applicability to perform this operation is determined. Once the trust is established, this operation signs the entire certificate. The caller must provide a credential that permits the caller to use the private key for this signing operation. The credential can be provided in the cryptographic context `CCHandle`. If `CCHandle` is NULL, the credentials in the `SignerVerifyContext` specify the credential value.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_CL_HANDLE
CSSMERR_TP_INVALID_CONTEXT_HANDLE
CSSMERR_TP_INVALID_CERTGROUP_POINTER
CSSMERR_TP_INVALID_CERTGROUP
CSSMERR_TP_INVALID_CERTIFICATE
CSSMERR_TP_UNKNOWN_FORMAT
```

```
CSSMERR_TP_INVALID_ACTION
CSSMERR_TP_INVALID_ACTION_DATA
CSSMERR_TP_VERIFY_ACTION_FAILED
CSSMERR_TP_INVALID_CRLGROUP_POINTER
CSSMERR_TP_INVALID_CRLGROUP
CSSMERR_TP_INVALID_CRL_AUTHORITY
CSSMERR_TP_INVALID_CALLERAUTH_CONTEXT_POINTER
CSSMERR_TP_INVALID_POLICY_IDENTIFIERS
CSSMERR_TP_INVALID_TIMESTRING
CSSMERR_TP_INVALID_STOP_ON_POLICY
CSSMERR_TP_INVALID_CALLBACK
CSSMERR_TP_INVALID_ANCHOR_CERT
CSSMERR_TP_CERTGROUP_INCOMPLETE
CSSMERR_TP_INVALID_DL_HANDLE
CSSMERR_TP_INVALID_DB_HANDLE
CSSMERR_TP_INVALID_DB_LIST_POINTER
CSSMERR_TP_INVALID_DB_LIST
CSSMERR_TP_AUTHENTICATION_FAILED
CSSMERR_TP_INSUFFICIENT_CREDENTIALS
CSSMERR_TP_NOT_TRUSTED
CSSMERR_TP_CERT_REVOKED
CSSMERR_TP_CERT_SUSPENDED
CSSMERR_TP_CERT_EXPIRED
CSSMERR_TP_CERT_NOT_VALID_YET
CSSMERR_TP_INVALID_CERT_AUTHORITY
CSSMERR_TP_INVALID_SIGNATURE
CSSMERR_TP_INVALID_NAME
CSSMERR_TP_CERTIFICATE_CANT_OPERATE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_CertCreateTemplate, *CSSM_TP_CrlSign*

Functions for the TP SPI:

TP_CertCreateTemplate, *TP_CrlSign*

TP_ConfirmCredResult

NAME

TP_ConfirmCredResult: CSSM_TP_ConfirmCredResult – Confirm credentials (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_ConfirmCredResult
(CSSM_TP_HANDLE TPHandle,
const CSSM_DATA *ReferenceIdentifier,
const CSSM_TP_CALLERAUTH_CONTEXT *CallerAuthCredentials,
const CSSM_TP_CONFIRM_RESPONSE *Responses,
const CSSM_TP_AUTHORITY_ID *PreferredAuthority)
SPI:
CSSM_RETURN CSSMTPI TP_ConfirmCredResult
(CSSM_TP_HANDLE TPHandle,
const CSSM_DATA *ReferenceIdentifier,
const CSSM_TP_CALLERAUTH_CONTEXT *CallerAuthCredentials,
const CSSM_TP_CONFIRM_RESPONSE *Responses,
const CSSM_TP_AUTHORITY_ID *PreferredAuthority)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the certification authority module used to perform this function.

`ReferenceIdentifier` (*input*)

A reference identifier that uniquely identifies execution of the call sequence `CSSM_TP_SubmitCredRequest()` and `CSSM_TP_RetrieveCredResult()` (or the equivalent TP SPI call pair) to submit a set of requests and to retrieve the results of those requests.

`CallerAuthCredentials` (input/optional)

This structure contains a set of caller authentication credentials. The authentication information can be a passphrase, a PIN, a completed registration form, a certificate, or a template of user-specific data. The required set of credentials is defined by the service provider module and recorded in a record in the MDS Primary relation. Multiple credentials can be required. If the local service provider module does not require credentials from a caller, then the `Credentials` field of this verification context structure can be NULL. The structure optionally contains additional credentials that can be used to support the authentication process. Authentication credentials required by the authority should be included in the `RequestInput`. The local TP module can forward information from the `CallerAuthCredentials` to the authority, as appropriate, but is not required to do so.

`Responses` (*input*)

An ordered vector of acknowledges indicating the caller's acceptance or rejection of results. The vector contains one acknowledgement per result returned by `CSSM_TP_RetrieveCredResult()` (CSSM API), or `TP_RetrieveCredResult()` (TP SPI).

`PreferredAuthority` (input/optional)

The identifier which uniquely describes the Authority to receive the acknowledgements. The structure can include:

- An identity certificate for the authority
- The location of the authority

DESCRIPTION

This function submits a vector of acknowledgements to a Certificate Authority for a set of requests and corresponding results identified by `ReferenceIdentifier`. The caller must execute the call sequence `CSSM_TP_SubmitCredRequest()` and `CSSM_TP_RetrieveCredResult()`(or the equivalent TP SPI calls) to submit a set of requests and to retrieve the results of those requests. Some Certificate Authority services accessed through the request and retrieve functions require confirmation. The function `CSSM_TP_RetrieveCredResult()` (CSSM API), or `TP_RetrieveCredResult()` (TP SPI), returns a value indicating whether the caller must invoke `CSSM_TP_ConfirmCredResult()`, (CSSM API), or `TP_ConfirmCredResult()` (TP SPI), to successfully complete the service.

The `Responses` vector accepts or rejects each result independently. If the caller rejects a returned result, the action taken by the authority depends on the requested type of service.

The `ReferenceIdentifier` also identifies the authority process state associated with the function pair `CSSM_TP_SubmitCredRequest()` and `CSSM_TP_RetrieveCredResult()` (or the equivalent TP SPI calls). The `PreferredAuthority` information can be used to further identify the authority to receive the acknowledgement. After successful execution of this function, the value of the `ReferenceIdentifier` is undefined and should not be used in subsequent operations in the current module attach session.

This function fails if `ReferenceIdentifier` is invalid or the Authority process can not be located.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_IDENTIFIER_POINTER
CSSMERR_TP_INVALID_IDENTIFIER
CSSMERR_TP_INVALID_CALLERAUTH_CONTEXT_POINTER
CSSMERR_TP_INVALID_POLICY_IDENTIFIERS
CSSMERR_TP_INVALID_TIMESTRING
CSSMERR_TP_INVALID_STOP_ON_POLICY
CSSMERR_TP_INVALID_CALLBACK
CSSMERR_TP_INVALID_ANCHOR_CERT
CSSMERR_TP_CERTGROUP_INCOMPLETE
CSSMERR_TP_INVALID_DL_HANDLE
CSSMERR_TP_INVALID_DB_HANDLE
CSSMERR_TP_INVALID_DB_LIST_POINTER
```

```
CSSMERR_TP_INVALID_DB_LIST
CSSMERR_TP_AUTHENTICATION_FAILED
CSSMERR_TP_INSUFFICIENT_CREDENTIALS
CSSMERR_TP_NOT_TRUSTED
CSSMERR_TP_CERT_REVOKED
CSSMERR_TP_CERT_SUSPENDED
CSSMERR_TP_CERT_EXPIRED
CSSMERR_TP_CERT_NOT_VALID_YET
CSSMERR_TP_INVALID_CERT_AUTHORITY
CSSMERR_TP_INVALID_SIGNATURE
CSSMERR_TP_INVALID_NAME
CSSMERR_TP_INVALID_RESPONSE_VECTOR
CSSMERR_TP_INVALID_AUTHORITY
CSSMERR_TP_NO_DEFAULT_AUTHORITY
CSSMERR_TP_UNSUPPORTED_ADDR_TYPE
CSSMERR_TP_INVALID_NETWORK_ADDR
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_SubmitCredRequest, *CSSM_TP_RetrieveCredResult*, *CSSM_TP_ReceiveConfirmation*

Functions for the TP SPI:

TP_SubmitCredRequest, *TP_RetrieveCredResult*, *TP_ReceiveConfirmation*

TP_CrlCreateTemplate

NAME

TP_CrlCreateTemplate: CSSM_TP_CrlCreateTemplate – Create an unsigned memory-resident CRL template (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_CrlCreateTemplate
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
uint32 NumberOfFields,
const CSSM_FIELD *CrlFields,
CSSM_DATA_PTR NewCrlTemplate)
SPI:
CSSM_RETURN CSSMTPI TP_CrlCreateTemplate
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
uint32 NumberOfFields,
const CSSM_FIELD *CrlFields,
CSSM_DATA_PTR NewCrlTemplate)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the add-in trust policy module used to perform this function.

`CLHandle` (input/optional)

The handle that describes the add-in certificate library module used to perform this function.

`NumberOfFields` (*input*)

The number of OID/value pairs specified in the `CrlFields` input parameter.

`CrlFields` (*input*)

Any array of field OID/value pairs containing the values to initialize the CRL attribute fields

`NewCrlTemplate` (*output*)

A pointer to the CSSM_DATA structure containing the new CRL. The `NewCrl->Data` is allocated by the service provider and must be deallocated by the application.

DESCRIPTION

This function creates an unsigned, memory-resident CRL template. Fields in the CRL are initialized based on the descriptive data specified by the OID/value input pairs in `CrlFields` and the local domain policy of the TP. The specified OID/value pairs can initialize all or a subset of the general attribute fields in the new CRL, though the module developer may specify a set of fields that must be or cannot be set using this operation. The `NewCrlTemplate` output is an unsigned CRL template in the format supported by the TP.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_TP_INVALID_CL_HANDLE`
`CSSMERR_TP_INVALID_FIELD_POINTER`
`CSSMERR_TP_UNKNOWN_TAG`
`CSSMERR_TP_INVALID_NUMBER_OF_FIELDS`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_CrlSignWithKey, *CSSM_TP_CrlSignWithCert*

Functions for the TP SPI:

TP_CrlSignWithKey, *TP_CrlSignWithCert*

TP_CrlVerify

NAME

TP_CrlVerify: CSSM_TP_CrlVerify – Verify integrity of the certificate revocation list (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_CrlVerify
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_ENCODED_CRL *CrlToBeVerified,
const CSSM_CERTGROUP *SignerCertGroup,
const CSSM_TP_VERIFY_CONTEXT *VerifyContext,
CSSM_TP_VERIFY_CONTEXT_RESULT_PTR RevokerVerifyResult)
SPI:
CSSM_RETURN CSSMTPI TP_CrlVerify
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CSP_HANDLE CSPHandle,
const CSSM_ENCODED_CRL *CrlToBeVerified,
const CSSM_CERTGROUP *SignerCertGroup,
const CSSM_TP_VERIFY_CONTEXT *VerifyContext,
CSSM_TP_VERIFY_CONTEXT_RESULT_PTR RevokerVerifyResult)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the add-in trust policy module used to perform this function.

`CLHandle` (input/optional)

The handle that describes the add-in certificate library module that can be used to manipulate the certificates to be verified. If no certificate library module is specified, the TP module uses an assumed CL module, if required.

`CSPHandle` (input/optional)

The handle referencing a Cryptographic Service Provider to be used to verify signatures on the signer's certificate and on the CRL. The TP module is responsible for creating the cryptographic context structure required to perform the verification operation. If no CSP is specified, the TP module uses an assumed CSP to perform the operations.

`CrlToBeVerified` (*input*)

A pointer to the CSSM_DATA structure containing a signed certificate revocation list to be verified. The CRL type and encoding are included in this structure.

`SignerCertGroup` (*input*)

A pointer to the CSSM_CERTGROUP structure containing one or more related certificates that paretially or fully represent the signer of the certificate revocation list. The first certificate in the group is the target certificate representing the CRL signer. Use of subsequent certificates is specific to the trust domain. For example, in a hierarchical trust model subsequent members are intermediate certificates of a certificate chain - the caller can specify additional points of trust represented by anchor certificates in the `VerifyContext`. The trust policy module can use these additional points of trust in the verification process.

`VerifyContext` (input/optional)

A structure containing credentials, policy information, and contextual information to be used in the verification process. All of the input values in the context are optional. The service provider can define default values or can attempt to operate without input for all the other fields of this input structure. The operation can fail if a necessary input value is omitted and the service module can not define an appropriate default value.

`RevokerVerifyResult` (output/optional)

A pointer to a structure containing information generation during the verification process. The information can include:

`Evidence`	(output/optional)
`NumberOfEvidences`	(output/optional)

DESCRIPTION

This function verifies the integrity of the certificate revocation list and determines whether it is trusted. The conditions for trust are part of the trust policy module. It can include conditions such as validity of the signer's certificate, verification of the signature on the CRL, the identity of the signer, the identity of the sender of the CRL, date the CRL was issued, the effective dates on the CRL, and so on.

The caller can specify additional points of trust represented by anchor certificates in the `VerifyContext`. The trust policy module can use these additional points of trust in the verification process.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_CL_HANDLE
CSSMERR_TP_INVALID_CSP_HANDLE
CSSMERR_TP_INVALID_CRL_TYPE
CSSMERR_TP_INVALID_CRL_ENCODING
CSSMERR_TP_INVALID_CRL_POINTER
CSSMERR_TP_INVALID_CRL
CSSMERR_TP_INVALID_CERTGROUP_POINTER
CSSMERR_TP_INVALID_CERTGROUP
CSSMERR_TP_INVALID_CERTIFICATE
```

```
CSSMERR_TP_INVALID_ACTION
CSSMERR_TP_INVALID_ACTION_DATA
CSSMERR_TP_VERIFY_ACTION_FAILED
CSSMERR_TP_INVALID_CRLGROUP_POINTER
CSSMERR_TP_INVALID_CRLGROUP
CSSMERR_TP_INVALID_CRL_AUTHORITY
CSSMERR_TP_INVALID_CALLERAUTH_CONTEXT_POINTER
CSSMERR_TP_INVALID_POLICY_IDENTIFIERS
CSSMERR_TP_INVALID_TIMESTRING
CSSMERR_TP_INVALID_STOP_ON_POLICY
CSSMERR_TP_INVALID_CALLBACK
CSSMERR_TP_INVALID_ANCHOR_CERT
CSSMERR_TP_CERTGROUP_INCOMPLETE
CSSMERR_TP_INVALID_DL_HANDLE
CSSMERR_TP_INVALID_DB_HANDLE
CSSMERR_TP_INVALID_DB_LIST_POINTER
CSSMERR_TP_INVALID_DB_LIST
CSSMERR_TP_AUTHENTICATION_FAILED
CSSMERR_TP_INSUFFICIENT_CREDENTIALS
CSSMERR_TP_NOT_TRUSTED
CSSMERR_TP_CERT_REVOKED
CSSMERR_TP_CERT_SUSPENDED
CSSMERR_TP_CERT_EXPIRED
CSSMERR_TP_CERT_NOT_VALID_YET
CSSMERR_TP_INVALID_CERT_AUTHORITY
CSSMERR_TP_INVALID_SIGNATURE
CSSMERR_TP_INVALID_NAME
CSSMERR_TP_CERTIFICATE_CANT_OPERATE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_CL_CrlVerify

Functions for the TP SPI:

CL_CrlVerify

TP_FormRequest

NAME

TP_FormRequest: CSSM_TP_FormRequest – Get form from authority (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_FormRequest
(CSSM_TP_HANDLE TPHandle,
const CSSM_TP_AUTHORITY_ID *PreferredAuthority,
CSSM_TP_FORM_TYPE FormType,
CSSM_DATA_PTR BlankForm)
SPI:
CSSM_RETURN CSSMTPI TP_FormRequest
(CSSM_TP_HANDLE TPHandle,
const CSSM_TP_AUTHORITY_ID *PreferredAuthority,
CSSM_TP_FORM_TYPE FormType,
CSSM_DATA_PTR BlankForm)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the certification authority module used to perform this function.

`PreferredAuthority` (input/optional)

A CSSM_TP_AUTHORITY_ID structure containing either a certificate that identifies the Authority process, or a network address directly or indirectly identifying the location of the authority. If the input is NULL, the module can assume a default authority location. If a default authority can not be assumed, the request can not be initiated and the operation fails.

`FormType` (*input*)

Indicates the type of form being requested.

`BlankForm` (*output*)

A CSSM_DATA structure containing the requested form. The caller must have knowledge of the structure of the form based on `FormType`.

DESCRIPTION

This function returns a blank form of type `FormType` from an Authority. If the `PreferredAuthority` list is NULL, the CA module can use a default authority name and location based on `FormType`. The CA module must incorporate knowledge of the interface protocol for communicating with the authority.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_AUTHORITY
CSSMERR_TP_NO_DEFAULT_AUTHORITY
CSSMERR_TP_UNSUPPORTED_ADDR_TYPE
CSSMERR_TP_INVALID_NETWORK_ADDR
CSSMERR_TP_INVALID_FORM_TYPE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_FormSubmit

Functions for the TP SPI:

TP_FormSubmit

TP_FormSubmit

NAME

TP_FormSubmit: CSSM_TP_FormSubmit – Submit form to ClearanceAuthority (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_FormSubmit
(CSSM_TP_HANDLE TPHandle,
CSSM_TP_FORM_TYPE FormType,
const CSSM_DATA *Form,
const CSSM_TP_AUTHORITY_ID *ClearanceAuthority,
const CSSM_TP_AUTHORITY_ID *RepresentedAuthority,
CSSM_ACCESS_CREDENTIALS_PTR Credentials)
SPI:
CSSM_RETURN CSSMTPI TP_FormSubmit
(CSSM_TP_HANDLE TPHandle,
CSSM_TP_FORM_TYPE FormType,
const CSSM_DATA *Form,
const CSSM_TP_AUTHORITY_ID *ClearanceAuthority,
const CSSM_TP_AUTHORITY_ID *RepresentedAuthority,
CSSM_ACCESS_CREDENTIALS_PTR Credentials)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

A handle for the service provider module that will perform the operation.

`FormType` (*input*)

Indicates the type of form being submitted.

`Form` (*input*)

A pointer to the CSSM_DATA structure containing the completed form to be submitted to the `ClearanceAuthority`.

`ClearanceAuthority` (input/optional)

A CSSM_TP_AUTHORITY_ID structure containing either a certificate that identifies the clearance authority process, or a network address directly or indirectly identifying the location of the authority. If the input is NULL, the service provider module can assume a default authority based on the `FormType` and contents of `Form`. If a default authority can not be assumed, the request can not be initiated and the operation fails.

`RepresentedAuthority` (input/optional)

A CSSM_TP_AUTHORITY_ID structure containing either a certificate that identifies the authority represented by the `ClearanceAuthority`, or a network address directly or indirectly identifying the location of the authority. If the input is NULL, the service provider module can assume a default authority based on the `FormType` and contents of `Form`. If a default authority can not be assumed, the request can not be initiated and the operation fails.

`Credentials` (output/optional)

A pointer to a structure containing one or more credentials issued in response to the contents of the `Form`. If the output is NULL, either no credentials were returned or an error occurred.

DESCRIPTION

The completed Form is submitted to a `ClearanceAuthority`, who is acting on behalf of a `RepresentedAuthority`. Typically the submitted form is requesting an authorization credential required as input to future service requests to the `RepresentedAuthority`.

If the form is honored by the `ClearanceAuthority`, then a set of one or more `Credentials` is returned to the requester. These credential can be used as the input credential in future service requests submitted to the `RepresentedAuthority`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_FORM_TYPE
CSSMERR_TP_INVALID_AUTHORITY
CSSMERR_TP_NO_DEFAULT_AUTHORITY
CSSMERR_TP_UNSUPPORTED_ADDR_TYPE
CSSMERR_TP_INVALID_NETWORK_ADDR
CSSMERR_TP_AUTHENTICATION_FAILED
CSSMERR_TP_INSUFFICIENT_CREDENTIALS
CSSMERR_TP_REJECTED_FORM
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_FormRequest

Functions for the TP SPI:

TP_FormRequest

TP_PassThrough

NAME

TP_PassThrough: CSSM_TP_PassThrough – Extend trust policy functionality

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_PassThrough
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DL_DB_LIST *DBList,
uint32 PassThroughId,
const void *InputParams,
void **OutputParams)
SPI:
CSSM_RETURN CSSMTPI TP_PassThrough
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DL_DB_LIST *DBList,
uint32 PassThroughId,
const void *InputParams,
void **OutputParams)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the add-in trust policy module used to perform this function.

`CLHandle` (input/optional)

The handle that describes the add-in certificate library module that can be used to manipulate the subject certificate and anchor certificates. If no certificate library module is specified, the TP module uses an assumed CL module, if required.

`CCHandle` (input/optional)

The handle that describes the context of the cryptographic operation. If the module-specific operation does not perform any cryptographic operations, a cryptographic context is not required

`DBList` (input/optional)

A list of handle pairs specifying a data storage library module and a data store, identifying certificate databases containing certificates (and possibly other security objects) that may be used by the pass-through function. If no DL and DB handle pairs are specified, the TP module can use an assumed DL module and an assumed data store for this operation.

`PassThroughId(`*input*`)`

An identifier assigned by a TP module to indicate the exported function to be performed.

`InputParams` (input/optional)

A pointer to a module, implementation-specific structure containing parameters to be interpreted in a function-specific manner by the requested TP module.

`OutputParams` (output/optional)

A pointer to a module, implementation-specific structure containing the output data. The service provider allocates the memory for substructures. The application must free the memory for the substructures.

DESCRIPTION

This function allows applications to call trust policy module-specific operations that have been exported. Such operations may include queries or services specific to the domain represented by the TP module.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_CL_HANDLE
CSSMERR_TP_INVALID_CONTEXT_HANDLE
CSSMERR_TP_INVALID_DL_HANDLE
CSSMERR_TP_INVALID_DB_HANDLE
CSSMERR_TP_INVALID_DB_LIST_POINTER
CSSMERR_TP_INVALID_DB_LIST
CSSMERR_TP_INVALID_PASSTHROUGH_ID
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

TP_ReceiveConfirmation

NAME

TP_ReceiveConfirmation: CSSM_TP_ReceiveConfirmation – Poll for confirmation (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_ReceiveConfirmation
(CSSM_TP_HANDLE TPHandle,
const CSSM_DATA *ReferenceIdentifier,
CSSM_TP_CONFIRM_RESPONSE_PTR *Responses,
sint32 *ElapsedTime)
SPI:
CSSM_RETURN CSSMTPI TP_ReceiveConfirmation
(CSSM_TP_HANDLE TPHandle,
const CSSM_DATA *ReferenceIdentifier,
CSSM_TP_CONFIRM_RESPONSE_PTR *Responses,
sint32 *ElapsedTime)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the certification authority module used to perform this function.

`ReferenceIdentifier` (*input*)

A reference identifier that uniquely identifies a set of service requests and the results created in response to those requests.

`Responses` (*output*)

An ordered vector of acknowledges indicating the caller's acceptance or rejection of results. The vector contains one acknowledgement per result created by the certificate authority.

`ElapsedTime` (*output*)

If the confirmation has not been received, this output value is the number of seconds elapsed since the certificate authority created the results or CSSM_ELAPSED_TIME_UNKNOWN. If the confirmation has been received, this output value is CSSM_ELAPSED_TIME_COMPLETE.

DESCRIPTION

A certificate authority uses this function to poll for confirmation from a requester who has been served by the authority. A requester sends a confirmation to the authority by successfully invoking the function `CSSM_TP_ConfirmCredResult()` (CSSM API), or `TP_ConfirmCredResult()` (TP SPI).

The `ReferenceIdentifier` uniquely identifies the service request and corresponding results for which confirmation is expected. This reference identifier need not be identical to the reference identifier used by the requester, but a one-to-one mapping between the two name spaces must be well-defined.

`Responses` is an ordered vector of acknowledgements indicating, for each returned result, whether the result was accepted or rejected by the original requester for whom the service was performed.

If a result is rejected by the receiver, then the authority process must undo the service if a reverse operation is possible and available.

If a fatal error occurs, this function returns an error code, indicating that the function call can never be completed. If confirmation has not been received, the function return value is CSSM_OK and the `ElapsedTime` is returned to the caller of this function. The time represents elapsed seconds since the service results were produced by the authority process. Note that there can be a difference between the time the authority process produces the results and the time the results are actually received by the requester. Elapsed time is relative and should increase with consecutive calls using the same `ReferenceIdentifier`. If the TP module has no knowledge of the elapsed time, the value CSSM_ELAPSED_TIME_UNKNOWN must be returned. If the service requester has confirmed receipt of the service results, this function returns CSSM_OK and `ElapsedTime` is CSSM_ELAPSED_TIME_COMPLETE.

This function can be invoked repeatedly until the confirmation is received or until the caller decides the acknowledgement may be lost and chooses to undo the results of the original service request.

This function fails if the `ReferenceIdentifier` is invalid or does not match any requested service for which confirmation is expected.

RETURN VALUE

A CSSM return value combined with elapsed time to indicate one of three results:

Complete Function	**Function Return**	**RetrieveOutput**	**EstimatedTime**
Result	Value		
Confirmation Received	CSSM_OK	CSSM_ELAPSED_TIME_ COMPLETE	
Confirmation not received, but expected in the future	CSSM_OK	CSSM_ELAPSED_TIME_ UNKNOWN or <elapsed seconds>	
Fatal Error, Confirmation is not expected	(!CSSM_OK)	NA	

For a return value of (!CSSM_OK), the return value is an error code.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_TP_INVALID_IDENTIFIER_POINTER`
`CSSMERR_TP_INVALID_IDENTIFIER`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_ConfirmCredResult

Functions for the TP SPI:

CSSM_TP_ConfirmCredResult

TP_SubmitCredRequest

NAME

TP_SubmitCredRequest: CSSM_TP_SubmitCredRequest – Submit credential request (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_SubmitCredRequest
(CSSM_TP_HANDLE TPHandle,
const CSSM_TP_AUTHORITY_ID *PreferredAuthority,
CSSM_TP_AUTHORITY_REQUEST_TYPE RequestType,
const CSSM_TP_REQUEST_SET *RequestInput,
const CSSM_TP_CALLERAUTH_CONTEXT *CallerAuthContext,
sint32 *EstimatedTime,
CSSM_DATA_PTR ReferenceIdentifier)
SPI:
CSSM_RETURN CSSMTPI TP_SubmitCredRequest
(CSSM_TP_HANDLE TPHandle,
const CSSM_TP_AUTHORITY_ID *PreferredAuthority,
CSSM_TP_AUTHORITY_REQUEST_TYPE RequestType,
const CSSM_TP_REQUEST_SET *RequestInput,
const CSSM_TP_CALLERAUTH_CONTEXT *CallerAuthContext,
sint32 *EstimatedTime,
CSSM_DATA_PTR ReferenceIdentifier)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`TPHandle` (*input*)

The handle that describes the certification authority module used to perform this function.

`PreferredAuthority` (input/optional)

The identifier which uniquely describes the Certificate Service Authority to submit the request to.

`RequestType` (*input*)

The identifier of the type of request to submit.

`RequestInput` (*input*)

A pointer to the input parameters to be submitted to the authority who will perform the requested service.

`CallerAuthContext` (input/optional)

This structure contains a set of caller authentication credentials. The authentication information can be a passphrase, a PIN, a completed registration form, a certificate, or a template of user-specific data. The required set of credentials is defined by the service

provider module and recorded in the MDS Primary relation. Multiple credentials can be required. If the local service provider module does not require credentials from a caller, then the `CallerCredentials` field of this verification context structure can be NULL. The structure optionally contains additional credentials that can be used to support the authentication process. Authentication credentials required by the authority should be included in the `RequestInput`. The local service provider module can forward this credential information to the authority, as appropriate, but is not required to do so.

`EstimatedTime` (*output*)

The number of estimated seconds before the service results are ready to be retrieved. A (default) value of zero indicates that the results can be retrieved immediately via the corresponding `CSSM_TP_RetrieveCredResult()` (CSSM API), or `TP_RetrieveCredResult()` (TP SPI), function call. When the local service provider module or the authority cannot estimate the time required to perform the requested service, the output value for estimated time is CSSM_ESTIMATED_TIME_UNKNOWN.

`ReferenceIdentifier` (*output*)

A reference identifier, which uniquely identifies this specific request. The handle persists across application executions and becomes undefined when all local processing of the request has completed. Local processing is completed in one of two ways:

- For certificate services that do not require explicit confirmation by the requester, the reference identifier is invalidated when the corresponding `CSSM_TP_RetrieveCredResult()` (CSSM API), or `TP_RetrieveCredResult()` (TP SPI), function completes (by returning valid results or by failure, which blocks returned results)
- For certificate services that require explicit confirmation by the requester, the reference identifier is invalidated by successfully invoking the function `CSSM_TP_ConfirmCredResu()` (CSSM API), or `CSSM_TP_ConfirmCredResult()` (TP SPI).

DESCRIPTION

If the caller is successfully authenticated, then this function submits a request to the Authority identified by `PreferredAuthority`. The authority service can be local or remote. If the Authority is not specified, then the TP module can assume a default authority based on the `RequestType` and the `CallerAuthContext`. `RequestType` indicates the type of Authority request and `RequestInput` specifies the input parameters needed by the authority to perform the request.

The request is submitted to the authority only if the TP module can successfully authenticate the caller. The `CallerAuthContext` presents the caller's credentials and a list of one or more policies under which the caller should be authenticated. Caller credentials can be presented in several forms:

- Memory-resident credential values, directly referenced by the structure
- Data bases containing credentials
- Callback functions that can be invoked to obtain credentials from an active entity

The local service provider must select and forward the credentials required by the Authority. The caller must provide all necessary credentials through the `CallerAuthContext` parameter.

If the caller can not be authenticated by the local service provider, the function fails and the request is not submitted to the selected authority.

This function returns a `ReferenceIdentifier` and an `EstimatedTime` (specified in seconds). `ReferenceIdentifier` is an ID for the submitted request. `EstimatedTime` defines the expected time to process the request. This time may be substantial when the request requires offline authentication procedures by the Authority process. In contrast, the estimated time can be zero, meaning the result can be obtained immediately using `CSSM_TP_RetrieveCredResult()` (CSSM API), or `TP_RetrieveCredResult()` (TP SPI). After the specified time has elapsed, the caller must use the function `CSSM_TP_RetrieveCredResult()` (CSSMAPI), or `TP_RetrieveCredResult()` (TP SPI), with the reference identifier, to obtain the result of the request.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_TP_INVALID_AUTHORITY
CSSMERR_TP_NO_DEFAULT_AUTHORITY
CSSMERR_TP_UNSUPPORTED_ADDR_TYPE
CSSMERR_TP_INVALID_NETWORK_ADDR
CSSMERR_TP_UNSUPPORTED_SERVICE
CSSMERR_TP_INVALID_REQUEST_INPUTS
CSSMERR_TP_INVALID_CALLERAUTH_CONTEXT_POINTER
CSSMERR_TP_INVALID_POLICY_IDENTIFIERS
CSSMERR_TP_INVALID_TIMESTRING
CSSMERR_TP_INVALID_STOP_ON_POLICY
CSSMERR_TP_INVALID_CALLBACK
CSSMERR_TP_INVALID_ANCHOR_CERT
CSSMERR_TP_CERTGROUP_INCOMPLETE
CSSMERR_TP_INVALID_DL_HANDLE
CSSMERR_TP_INVALID_DB_HANDLE
CSSMERR_TP_INVALID_DB_LIST_POINTER
CSSMERR_TP_INVALID_DB_LIST
CSSMERR_TP_AUTHENTICATION_FAILED
CSSMERR_TP_INSUFFICIENT_CREDENTIALS
CSSMERR_TP_NOT_TRUSTED
CSSMERR_TP_CERT_REVOKED
CSSMERR_TP_CERT_SUSPENDED
CSSMERR_TP_CERT_EXPIRED
CSSMERR_TP_CERT_NOT_VALID_YET
CSSMERR_TP_INVALID_CERT_AUTHORITY
CSSMERR_TP_INVALID_SIGNATURE
CSSMERR_TP_INVALID_NAME
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_TP_RetrieveCredResult

Functions for the TP SPI:

TP_RetrieveCredResult

TP_TupleGroupToCertGroup

NAME

TP_TupleGroupToCertGroup: CSSM_TP_TupleGroupToCertGroup – Create a set of certificate templates (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_TP_TupleGroupToCertGroup
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE  CLHandle,
const CSSM_TUPLEGROUP *TupleGroup,
CSSM_CERTGROUP_PTR *CertTemplates)
SPI:
CSSM_RETURN CSSMTPI TP_TupleGroupToCertGroup
(CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE  CLHandle,
const CSSM_TUPLEGROUP *TupleGroup,
CSSM_CERTGROUP_PTR *CertTemplates)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

DESCRIPTION

This function creates a set of certificate templates based on a set of input tuples. The tuples describe a set of authorizations for one or more subjects. The trust policy service provider maps these authorizations to appropriate template values for one or more certificates of the type managed by the Trust Policy module. The resulting certificate templates can be input to a certificate creation function, such as `CSSM_CL_CertSign()`, (CSSM API), or `CL_CertSign()`, (TP SPI). The signed certificates created by these functions should carry the authorizations described in the input tuples.

PARAMETERS

`TPHandle` (*input*)

The handle that describes the trust policy service module used to perform this function.

`CLHandle` (input/optional)

The handle that describes the certificate library module that can be used to assist in the creation of field values. If no certificate library module is specified, the TP module uses an assumed CL module, if required.

`TupleGroup` (*input*)

A pointer to a group of CSSM_TUPLE describing authorizations for one or more subjects.

`CertTemplates` (*output*)

A pointer to a structure containing references to one or more certificate templates resulting from the translation process. Storage for the structure and certificate templates is allocated by the service provider and must be deallocated by the application.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_TP_INVALID_CL_HANDLE
CSSMERR_TP_INVALID_TUPLEGROUP_POINTER
CSSMERR_TP_INVALID_TUPLEGROUP
CSSMERR_TP_INVALID_TUPLE

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

For the CSSM API:

CSSM_TP_CertGroupToTupleGroup, *CSSM_AC_AuthCompute*

For the TP SPI:

TP_CertGroupToTupleGroup, *AC_AuthCompute*

Terminate

NAME

Terminate – Clean up module-manager-specific activities (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI Terminate
(void)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

None.

DESCRIPTION

This function performs any module-manager-specific cleanup activities in preparation for unloading of the elective module manager.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSSM_EMM_AUTHENTICATE_FAILED`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *Initialize*

UnwrapKey

NAME

UnwrapKey: CSSM_UnwrapKey, CSP_UnwrapKey – Unwrap the wrapped key (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_UnwrapKey
(CSSM_CC_HANDLE CCHandle,
const CSSM_KEY *PublicKey,
const CSSM_WRAP_KEY *WrappedKey,
uint32 KeyUsage,
uint32 KeyAttr,
const CSSM_DATA *KeyLabel,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry,
CSSM_KEY_PTR UnwrappedKey,
CSSM_DATA_PTR DescriptiveData)
SPI:
CSSM_RETURN CSSMCSPI CSP_UnwrapKey
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
const CSSM_KEY *PublicKey,
const CSSM_WRAP_KEY *WrappedKey,
uint32 KeyUsage,
uint32 KeyAttr,
const CSSM_DATA *KeyLabel,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry,
CSSM_KEY_PTR UnwrappedKey,
CSSM_DATA_PTR DescriptiveData,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation.

`PublicKey` (input/optional)

The public key corresponding to the private key being unwrapped. If a symmetric key is being unwrapped, then this parameter must be NULL.

`WrappedKey` (*input*)

A pointer to the wrapped key. The wrapped key may be a symmetric key or the private key of a public/private key pair. The unwrapping method is specified as meta data within the wrapped key and is not specified outside of the wrapped key.

`KeyUsage` (*input*)

A bit mask indicating all permitted uses for the unwrapped key. If no value is specified, the CSP defines the usage mask for the unwrapped key.

`KeyAttr` (*input*)

A bit mask defining other attribute values to be associated with the unwrapped key.

`KeyLabel` (input/optional)

Pointer to a byte string that will be used as the label for the unwrapped key.

`CredAndAclEntry` (input/optional)

A structure containing one or more credentials authorized for creating a key and the prototype ACL entry that will control future use of the newly created key. The credentials and ACL entry prototype can be presented as immediate values or callback functions can be provided for use by the CSP to acquire the credentials and/or the ACL entry interactively. If the CSP provides public access for creating a key, then the credentials can be NULL. If the CSP defines a default initial ACL entry for the new key, then the ACL entry prototype can be an empty list.

`UnwrappedKey` (*output*)

A pointer to a CSSM_KEY structure that returns the unwrapped key.

`DescriptiveData` (*output*)

A pointer to a CSSM_DATA structure that returns any additional descriptive data that was associated with the key during the wrapping operation. It is assumed that the caller incorporated knowledge of the structure of this data. If no additional data is associated with the imported key, this output value is NULL.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

`Privilege` (*input*)

The export privilege to be applied during the cryptographic operation. This parameter is forwarded to the CSP after CSSM verifies the caller and service provider privilege set includes the specified PRIVILEGE.

DESCRIPTION

This function unwraps the wrapped key using the context. The wrapped key can be a symmetric key or a private key. If the unwrapping algorithm is a symmetric algorithm, then a symmetric context must be provided. If the unwrapping algorithm is an asymmetric algorithm, then an asymmetric context must be provided. If the key is a private key, then an asymmetric context must be provide describing the unwrapping algorithm. The CSP can require the caller to provide credentials authorizing the caller to store the

unwrapped key within the CSP. The CSP can also require that the caller provide an initial ACL entry to control future access to the newly stored key. These credentials and the initial ACL entry value are provided in `CredAndAclEntry` parameter. If the unwrapping algorithm is CSSM_ALGID_NONE and the wrapped key is actually a raw key (as indicated by its key attributes), then the key is imported into the CSP. Support for a CSSM_ALGID_NONE unwrapping algorithm is at the option of the CSP. The unwrapped key is restored to its original pre-wrap state based on the key attributes recorded by the wrapped key during the wrap operation. These attributes must not be modified by the caller.

Authorization policy can restrict the set of callers who can create a new resource. In this case, the caller must present a set of access credentials for authorization. Upon successfully authenticating the credentials, the template that verified the presented samples identifies the ACL entry that will be used in the authorization computation. If the caller is authorized, the new resource is created.

The caller must provide an initial ACL entry to be associated with the newly created resource. This entry is used to control future access to the new resource and (since the subject is deemed to be the "Owner") exercise control over its associated ACL. The caller can specify the following items for initializing an ACL entry:

- Subject - A CSSM_LIST structure, containing the type of the subject and a template value that can be used to verify samples that are presented in credentials when resource access is requested.
- Delegation flag - A value indicating whether the Subject can delegate the permissions recorded in the `AuthorizationTag`. (This item only applies to public key subjects).
- Authorization tag - The set of permissions that are granted to the Subject.
- Validity period - The start time and the stop time for which the ACL entry is valid.
- ACL entry tag - A user-defined string value associated with the ACL entry.

  The service provider can modify the caller-provided initial ACL entry to conform to any innate resource-access policy that the service provider may be required to enforce. If the initial ACL entry provided by the caller contains values or permissions that are not supported by the service provider, then the service provider can modify the initial ACL appropriately or can fail the request to create the new resource. Service providers list their supported `AuthorizationTag` values in their Module Directory Services primary record.

NOTES

The `KeyData` field of the CSSM_KEY structure is allocated by the CSP. The application is required to free this memory using the `CSSM_FreeKey()` (CSSM API), or `CSP_FreeKey()` (CSP SPI), function or with the memory functions registered for the `CSPHandle`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSP_KEY_LABEL_ALREADY_EXISTS
CSSMERR_CSP_PUBLIC_KEY_INCONSISTENT
CSSMERR_CSP_PRIVATE_KEY_ALREADY_EXIST
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_WrapKey

Functions for the CSP SPI:

CSP_WrapKey

UnwrapKeyP

NAME

UnwrapKeyP – Unwrap the wrapped keys with privilege (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_UnwrapKeyP
(CSSM_CC_HANDLE CCHandle,
const CSSM_KEY *PublicKey,
const CSSM_WRAP_KEY *WrappedKey,
uint32 KeyUsage,
uint32 KeyAttr,
const CSSM_DATA *KeyLabel,
const CSSM_RESOURCE_CONTROL_CONTEXT *CredAndAclEntry,
CSSM_KEY_PTR UnwrappedKey,
CSSM_DATA_PTR DescriptiveData,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`Privilege` (*input*)

The privilege to be applied during the cryptographic operation.

See *CSSM_UnwrapKey*.

DESCRIPTION

This function is similar to `CSSM_UnwrapKey()`. It also accepts a USEE tag as a privilege request parameter. CSSM checks that either its own privilege set or the Application's privilege set (if the Application is signed) includes the tag. If the tag is found and the service provider privilege set indicates that it is supported, the tag is forwarded to the service provider.

NOTES

The `KeyData` field of the CSSM_KEY structure is allocated by the CSP. The application is required to free this memory using the `CSSM_FreeKey()` function or with the memory functions registered for the CSPHandle.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSP_KEY_LABEL_ALREADY_EXISTS
CSSMERR_CSP_PUBLIC_KEY_INCONSISTENT
CSSMERR_CSP_PRIVATE_KEY_ALREADY_EXIST
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

VerifyData

NAME

VerifyData: CSSM_VerifyData, CSP_VerifyData – Verify input buffer data (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_VerifyData
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount,
CSSM_ALGORITHMS DigestAlgorithm,
const CSSM_DATA *Signature)
SPI:
CSSM_RETURN CSSMCSPI CSP_VerifyData
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
const CSSM_DATA *DataBufs,
uint32 DataBufCount,
CSSM_ALGORITHMS DigestAlgorithm,
const CSSM_DATA *Signature)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`DataBufs` (*input*)

A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

`DataBufCount` (*input*)

The number of `DataBufs` to be verified.

`DigestAlgorithm` (*input*)

If verifying just a digest, specifies the type of digest. In this case, the context should only specify the encryption algorithm. If not verifying just a digest, it must be CSSM_ALGID_NONE. In this case, the context should specify the combination digest/encryption algorithm.

`Signature` (*input*)

A pointer to a CSSM_DATA structure which contains the signature and the size of the signature.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DESCRIPTION

This function verifies all data contained in the set of input buffers based on the input signature.

Verifying can include digesting the data and decrypting the digest (from the signature) or verifying just the digest (already calculated by the application). If digesting the data and decrypting the digest, then the context should specify both digest and decryption algorithms (for example, CSSM_ALGID_MD5WithRSA). In this case, the `DigestAlgorithm` parameter must be set to CSSM_ALGID_NONE. If signing just the digest, then the context should specify just the decryption algorithm and the `DigestAlgorithm` parameter should specify the type of digest (for example, CSSM_ALGID_MD5). Also, `DataBufCount` must be 1.

If the signing algorithm is not reversible or strictly limits the size of the signed data, then the algorithm can specify verification without digesting. In this case, the verify operation is performed on the input data and the size of the input data is restricted by the service provider.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSP_INPUT_LENGTH_ERROR
CSSMERR_CSP_VERIFY_FAILED
CSSMERR_CSP_INVALID_SIGNATURE
CSSMERR_CSP_INVALID_DIGEST_ALGORITHM
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_SignData, *CSSM_VerifyDataInit*, *CSSM_VerifyDataUpdate*, *CSSM_VerifyDataFinal*

Functions for the CSP SPI:

CSP_SignData, *CSP_VerifyDataInit*, *CSP_VerifyDataUpdate*, *CSP_VerifyDataFinal*

VerifyDataFinal

NAME

VerifyDataFinal: CSSM_VerifyDataFinal, CSP_VerifyDataFinal – Finalize the staged verify data (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_VerifyDataFinal
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *Signature)
SPI:
CSSM_BOOL CSSMCSPI CSP_VerifyDataFinal
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *Signature)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`Signature` (*input*)

A pointer to a CSSM_DATA structure which contains the starting address for the signature to verify against and the length of the signature in bytes.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function finalizes the staged verify data function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSP_INPUT_LENGTH_ERROR
CSSMERR_CSP_VERIFY_FAILED
CSSMERR_CSP_INVALID_SIGNATURE
```

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_VerifyData, *CSSM_VerifyDataInit*, *CSSM_VerifyDataUpdate*

Functions for the CSP SPI:

CSP_VerifyData, *CSP_VerifyDataInit*, *CSP_VerifyDataUpdate*

VerifyDataInit

NAME

VerifyDataInit: CSSM_VerifyDataInit, CSP_VerifyDataInit – Initialize the staged verify data (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_VerifyDataInit
(CSSM_CC_HANDLE CCHandle)
SPI:
CSSM_RETURN CSSMCSPI CSP_VerifyDataInit
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform up calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DESCRIPTION

This function initializes the staged verify data function.

For staged operations, a combination operation selecting both a digesting algorithm and a verification algorithm must be specified.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_VerifyDataUpdate, *CSSM_VerifyDataFinal*, *CSSM_VerifyData*

Functions for the CSP SPI:

CSP_VerifyDataUpdate, *CSP_VerifyDataFinal*, *CSP_VerifyData*

VerifyDataUpdate

NAME

VerifyDataUpdate: CSSM_VerifyDataUpdate, CSP_VerifyDataUpdate – Continue the staged verification (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_VerifyDataUpdate
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount)
SPI:
CSSM_RETURN CSSMCSPI CSP_VerifyDataUpdate
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`DataBufs` (*input*)

A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

`DataBufCount` (*input*)

The number of DataBufs to be verified.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function continues the staged verification process over all data contained in the set of input. Verification will be based on the signature presented as input when finalizing the staged verification process.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_VerifyData, *CSSM_VerifyDataInit*, *CSSM_VerifyDataFinal*

Functions for the CSP SPI:

CSP_VerifyData, *CSP_VerifyDataInit*, *CSP_VerifyDataFinal*

VerifyDevice

NAME

VerifyDevice: CSSM_VerifyDevice, CSP_VerifyDevice – Cause the cryptographic module to perform a self verification and integrity check (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_VerifyDevice
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_DATA *DeviceCert)
SPI:
CSSM_RETURN CSSMCSPI CSP_VerifyDevice
(CSSM_CSP_HANDLE CSPHandle,
const CSSM_DATA *DeviceCert)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform this function. If a NULL handle is specified, CSSM returns error.

`DeviceCert` (*input*)

Pointer to CSSM_DATA structure that contains data that identifies the cryptographic device.

DESCRIPTION

This function triggers the cryptographic module to perform self verification and integrity checking.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_DEVICE_VERIFY_FAILED`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

VerifyMac

NAME

VerifyMac: CSSM_VerifyMac, CSP_VerifyMac – Verify the message authentication code (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_VerifyMac
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount,
const CSSM_DATA *Mac)
SPI:
CSSM_RETURN CSSMCSPI CSP_VerifyMac
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
const CSSM_DATA *DataBufs,
uint32 DataBufCount,
const CSSM_DATA *Mac)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`DataBufs` (*input*)

A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

`DataBufCount` (*input*)

The number of DataBufs.

`Mac` (*input*)

A pointer to the CSSM_DATA structure containing the MAC to verify.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DESCRIPTION

This function verifies the message authentication code over all data contained in the set of input buffers based on the input signature.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSP_INPUT_LENGTH_ERROR`
`CSSMERR_CSP_VERIFY_FAILED`
`CSSMERR_CSP_INVALID_SIGNATURE`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_VerifyMacInit, *CSSM_VerifyMacUpdate*, *CSSM_VerifyMacFinal*

Functions for the CSP SPI:

CSP_VerifyMacInit, *CSP_VerifyMacUpdate*, *CSP_VerifyMacFinal*

VerifyMacFinal

NAME

VerifyMacFinal: CSSM_VerifyMacFinal, CSP_VerifyMacFinal – Finalize the staged message authentication code (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_VerifyMacFinal
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *Mac)
SPI:
CSSM_RETURN CSSMCSPI CSP_VerifyMacFinal
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *Mac)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`Mac` (*input*)

A pointer to the CSSM_DATA structure containing the MAC to verify.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function finalizes the staged message authentication code verification function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSP_INPUT_LENGTH_ERROR
CSSMERR_CSP_VERIFY_FAILED
CSSMERR_CSP_INVALID_SIGNATURE
```

COMMENTS FOR SPI

The output is returned to the caller as specified in Buffer Management for Cryptographic Services.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_VerifyMac, *CSSM_VerifyMacInit*, *CSSM_VerifyMacUpdate*

Functions for the CSP SPI:

CSP_VerifyMac, *CSP_VerifyMacInit*, *CSP_VerifyMacUpdate*

VerifyMacInit

NAME

VerifyMacInit: CSSM_VerifyMacInit, CSP_VerifyMacInit – Initialize the staged message authentication code (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_VerifyMacInit
(CSSM_CC_HANDLE CCHandle)
SPI:
CSSM_RETURN CSSMCSPI CSP_VerifyMacInit
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

DESCRIPTION

This function initializes the staged message authentication code verification function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_VerifyMac, *CSSM_VerifyMacUpdate*, *CSSM_VerifyMacFinal*

Functions for the CSP SPI:

CSP_VerifyMac, *CSP_VerifyMacUpdate*, *CSP_VerifyMacFinal*

VerifyMacUpdate

NAME

VerifyMacUpdate: CSSM_VerifyMacUpdate, CSP_VerifyMacUpdate – Continue the staged process of verifying the message authentication code (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_VerifyMacUpdate
(CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount)
SPI:
CSSM_RETURN CSSMCSPI CSP_VerifyMacUpdate
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA *DataBufs,
uint32 DataBufCount)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

`DataBufs` (*input*)

A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

`DataBufCount` (*input*)

The number of DataBufs.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform calls to CSSM for the memory functions managed by CSSM.

DESCRIPTION

This function continues the staged process of verifying the message authentication code over all data in the set of input buffers. Verification will be based on the authentication code presented as input when finalizing the staged verification process.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_VerifyMac, *CSSM_VerifyMacInit*, *CSSM_VerifyMacFinal*

Functions for the CSP SPI:

CSP_VerifyMac, *CSP_VerifyMacInit*, *CSP_VerifyMacFinal*

WrapKey

NAME

WrapKey: CSSM_WrapKey, CSP_WrapKey – Wrap a key using the context (CDSA)

SYNOPSIS

include <cssm.h>

```
API:
CSSM_RETURN CSSMAPI CSSM_WrapKey
(CSSM_CC_HANDLE CCHandle,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_KEY *Key,
const CSSM_DATA *DescriptiveData,
CSSM_WRAP_KEY_PTR WrappedKey)
SPI:
CSSM_RETURN CSSMCSPI CSP_WrapKey
(CSSM_CSP_HANDLE CSPHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_CONTEXT *Context,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_KEY *Key,
const CSSM_DATA *DescriptiveData,
CSSM_WRAP_KEY_PTR WrappedKey,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

API PARAMETERS

`CCHandle` (*input*)

The handle to the context that describes this cryptographic operation.

`AccessCred` (*input*)

A pointer to the set of one or more credentials required to access the private or secret key to be exported from the CSP. The credentials structure can contain an immediate value for the credential, such as a passphrase, or the caller can specify a callback function the CSP can use to obtain one or more credentials.

`Key` (*input*)

A pointer to the key to be wrapped.

`DescriptiveData` (input/optional)

A pointer to a CSSM_DATA structure containing additional descriptive data to be associated and included with the key during the wrapping operation. The caller and the wrapping algorithm incorporate knowledge of the structure of the descriptive data. If the wrapping algorithm does not accept additional descriptive data, then this parameter must be NULL. If the wrapping algorithm accepts descriptive data, the corresponding unwrapping algorithm can be used to extract the descriptive data and the key.

`WrappedKey` (*output*)

A pointer to a CSSM_WRAP_KEY structure that returns the wrapped key.

SPI PARAMETERS

`CSPHandle` (*input*)

The handle that describes the add-in Cryptographic Service Provider module used to perform up-calls to CSSM for the memory functions managed by CSSM.

`Context` (*input*)

Pointer to CSSM_CONTEXT structure that describes the attributes with this context.

`Privilege` (*input*)

The export privilege to be applied during the cryptographic operation. This parameter is forwarded to the CSP after CSSM verifies the caller and service provider privilege set includes the specified PRIVILEGE.

DESCRIPTION

This function wraps the supplied key using the context. It allows a key to be exported from a CSP. Four types of wrapping exist:

1. Wrap a symmetric key with a symmetric key.
2. Wrap a symmetric key with an asymmetric public key.
3. Wrap an asymmetric private key with a symmetric key.
4. Wrap an asymmetric private key with an asymmetric public key.

For types 1 and 3, a symmetric context should be provided. For types 2 and 4, an asymmetric context is provided. If there is a CSSM_ATTRIBUTE_WRAPPED_KEY_FORMAT argument in the context represented by the `CCHandle`, the value of the attribute specifies the format of the wrapped key. If this argument is not present, the symmetric key is wrapped according to CMS for types 1 and 3, and according to PKCS8 for types 2 and 4. If the wrapping algorithm in the context is CSSM_ALGID_NONE, then the key is returned in raw format, if permitted and supported by the CSP (in this case the CSSM_ATTRIBUTE_WRAPPED_KEY_FORMAT attribute is ignored). All significant key attributes are incorporated into the `KeyHeader` of the returned `WrappedKey`, such that the state of the key can be fully restored by the `unwrap` process.

The CSP can require that the cryptographic context includes access credentials for authentication and authorization checks when using the secret or private key.

NOTES

The `KeyData` field of the CSSM_KEY structure is allocated by the CSP. The application is required to free this memory using the `CSSM_FreeKey()` (CSSM API), or `CSP_FreeKey()` (CSP SPI) function, or with the memory functions registered for the `CSPHandle`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

None specific to this call.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions for the CSSM API:

CSSM_UnwrapKey

Functions for the CSP SPI:

CSP_UnwrapKey

WrapKeyP

NAME

WrapKeyP – Wrap a key with privilege (CDSA)

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI CSSM_WrapKeyP
(CSSM_CC_HANDLE CCHandle,
const CSSM_ACCESS_CREDENTIALS *AccessCred,
const CSSM_KEY *Key,
const CSSM_DATA *DescriptiveData,
CSSM_WRAP_KEY_PTR WrappedKey,
CSSM_PRIVILEGE Privilege)
```

LIBRARY

Common Security Services Manager library (`cdsa$incssm300_shr.exe`)

PARAMETERS

`Privilege` (*input*)

The privilege to be applied during the cryptographic operation.

See *CSSM_WrapKey*.

DESCRIPTION

This function is similar to `CSSM_WrapKey()`. It also accepts a USEE tag as a privilege request parameter. CSSM checks that either its own privilege set or the application's privilege set (if the application is signed) includes the tag. If the tag is found, and the service provider privilege set indicates that it is supported, the tag is forwarded to the service provider.

NOTES

The `KeyData` field of the CSSM_KEY structure is allocated by the CSP. The application is required to free this memory using the `CSSM_FreeKey()` functon, or with the memory functions registered for the `CSPHandle`.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Elective Module Manager (EMM) API Functions

These functions are implemented by an Elective Module Manager and are made available to CSSM in a function table.

For a module attach of a service provider managed by an EMM, CSSM invokes ModuleManagerAuthenticate(). Upon successful completion of this function, the elective module manager returns its function table to CSSM with the following modules:

- DeregisterDispatchTable
- EventNotifyManager
- Initialize
- ModuleManagerAuthenticate
- RefreshFunctionTable
- RegisterDispatchTable
- Terminate

DeregisterDispatchTable

NAME

DeregisterDispatchTable – Invalidate CSSM pointers to EMM

SYNOPSIS

include <cssm.h>

```
(void)
```

PARAMETERS

None.

DESCRIPTION

This EMM-defined function is invoked by CSSM once for each `CSSM_ModuleDetach()` operation issued against a service provider of the type managed by the EMM. CSSM uses this function to inform the EMM that the set of CSSM function pointers provided to the EMM when the session was attached are no longer valid.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *RegisterDispatchTable*

EventNotifyManager

NAME

EventNotifyManager – Receive an event notification

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI EventNotifyManager
(const CSSM_MANAGER_EVENT_NOTIFICATION *EventDescription)
```

PARAMETERS

`EventDescription`

A structure containing the following fields:

`DestinationModuleManagerType` (*input*)

The unique service mask identifying the destination module manager.

`SourceModuleManagerType` (*input*)

The unique service mask identifying the source module manager.

`Event` (*input*)

An identifier indicating the event that has or will take place.

`EventId` (input/optional)

A unique identifier associated with this event notification. It must be used in any reply notification that results from this event notification.

`EventData` (input/optional)

Arbitrary data (required or informational) for this event.

DESCRIPTION

This function receives an event notification from another module manager. The source manager is identified by its service mask. The specified event type is interpreted by the received and the appropriate actions must be taken in response. `EventId` and `EventData` are optional. The `EventId` is specified by the source module manager when a reply is expected. The destination module manager must use this identifier when replying to the event notification. The `EventData` is additional data or descriptive information provided to the destination manager.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSSM_MODULE_MANAGER_NOT_FOUND`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Initialize

NAME

Initialize – Verify module version

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI Initialize
(uint32 VerMajor,
uint32 VerMinor)
```

PARAMETERS

`VerMajor` (*input*)

The major version number of the CSSM that is invoking this module manager.

`VerMinor` (*input*)

The minor version number of the CSSM that is invoking this module manager.

DESCRIPTION

This function checks whether the current version of the module is compatible with the CSSM version specified as input and performs any module-manager-specific setup activities.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSSM_MODULE_MANAGER_INITIALIZE_FAIL`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *Terminate*

ModuleManagerAuthenticate

NAME

ModuleManagerAuthenticate – Module manager authentication

SYNOPSIS

include <cdsa/mds.h>

```
CSSM_RETURN CSSMAPI ModuleManagerAuthenticate
(CSSM_KEY_HIERARCHY KeyHierarchy,
const CSSM_GUID *CssmGuid,
const CSSM_GUID *AppGuid,
CSSM_MANAGER_REGISTRATION_INFO_PTR FunctionTable)
```

PARAMETERS

`KeyHierarchy` (*input*)

The CSSM_KEY_HIERARCHY flag indicating which embedded key(s) CSSM should use when verifying the integrity of the module manager.

`CssmGuid` (*input*)

A CSSM_GUID value identifying the calling CSSM. The elective module manager can use this value to locate the signed manifest credentials for CSSM.

`AppGuid` (input/optional)

A CSSM_GUID value identifying the application who invoked the calling CSSM. The elective module manager can use this value to locate the signed manifest credentials for the application.

`FunctionTable` (*output*)

A set of function pointers for EMM-defined functions used by CSSM to communicate state changes related to module attach and module detach operations.

DESCRIPTION

This function should perform the elective module manager's half of the bilateral authentication procedure with CSSM. The `CssmGuid` is used to locate the CSSM's credentials to be verified. The credentials are a zipped, signed manifest.

The `KeyHierarchy` indicates which public key should be used as the root when checking the integrity of the module manager. The `AppGuid` is used to locate the application's signed manifest credentials. The elective module manager must check the application's credentials to verify the application's authorization. If no privileges are requested, then the application is not required to provide a GUID nor a set of signed manifest credentials.

Upon successful completion, the elective module manager returns its function table to the calling CSSM. The EMM function table contains the set of EMM entry points that CSSM uses to notify the module manager of significant events such as module attach and module detach requests issued by an application, and event notifications issued by other module managers.

This function symbol must be exported by the elective module manager, so CSSM can invoke this function upon completion of the loading process.

This function is the first module manager interface invoked by CSSM after loading and invoking the main entry point. In particular, the elective module manager's initialize function is invoked by CSSM after this function has successfully completed execution.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

RefreshFunctionTable

NAME

RefreshFunctionTable – Gets EMM-defined API function

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI RefreshFunctionTable
(CSSM_FUNC_NAME_ADDR_PTR FuncNameAddrPtr,
uint32 NumOfFuncNameAddr)
```

PARAMETERS

`FuncNameAddrPtr` (input/output)

A pointer to a table mapping function names to EMM-defined APIs.

`NumOfFuncNameAddr` (*input*)

The number of entries in the table referenced by `FuncNameAddrPtr`.

DESCRIPTION

CSSM invokes this function to obtain the EMM-defined API function. The table is returned to CSSM in `FuncNameAddrPtr` and CSSM returns the table to the application. The application uses this table to invoke the security services defined by the EMM's service category. CSSM must obtain and forward the API table to the application on behalf of the EMM because the application is not aware of the optional nature of the EMM. Applications use CSSM to obtain the API function table for basic module managers and elective module managers, providing a uniform application programming model.

If the Elective Module Manager needs the service provider's SPI function table in order to initialize the API function table, the EMM can obtain the SPI function table by invoking the CSSM-provided service `cssm_GetAttachFunctions()`. The service module may implement only a subset of the defined functions and the EMM may wish to manage these functions in a particular manner through the API mapping. The elective module manager uses the SPI function table to dispatch application calls for service to attached modules.

Multiple applications and multiple instances of a service module can be concurrently active. The single elective module manager is responsible for managing all of these concurrent sessions. After completing initialization of the API function table, the EMM returns the refreshed API table to CSSM.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

RegisterDispatchTable

NAME

RegisterDispatchTable – Provide the EMM with CSSM function pointers

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI RegisterDispatchTable
(CSSM_STATE_FUNCS_PTR CssmStateCallTable)
```

PARAMETERS

`CssmStateCallTable` (*input*)

A table of function pointers for the set of CSSM-defined functions the elective module manager can use to query and control the state of an attach-session between an application and a service provider managed by the module manager.

DESCRIPTION

This EMM-defined function is invoked by CSSM once for each `CSSM_ModuleAttach()`, operation requesting a service provider of the type managed by the EMM. CSSM uses this function to provide the EMM with a set of CSSM function pointers. The EMM invokes these functions at anytime during the life cycle of the attach-session to obtain information about the current state and to modify the current state of the attach session.

When the attach-session is terminated, CSSM informs the module manager by invoking the EMM function `DeregisterDispatchTable()`. The corresponding set of CSSM state functions become invalid at that time.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *DeregisterDispatchTable*

Terminate

NAME

Terminate – Clean up module-manager-specific activities

SYNOPSIS

include <cssm.h>

```
CSSM_RETURN CSSMAPI Terminate
(void)
```

PARAMETERS

None.

DESCRIPTION

This function performs any module-manager-specific cleanup activities in preparation for unloading of the elective module manager.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSSM_EMM_AUTHENTICATE_FAILED.`

SEE ALSO

Books

Intel CDSA Application Developer's Guide

Online Help

Functions: *Initialize*

Human Recognition Service (HRS) API Functions

CDSA/HRS (Common Data Security Architecture/Human Recognition Service) is a CSSM (Common Security Services Manager) EMM (Elective Module Manager). It is intended to provide a high-level generic authentication model, suited to use for any form of human authentication. Particular emphasis has been made in the design on its suitability for authentication using biometric technology.

It covers the basic functions of Enrollment, Verification, and Identification, and includes a database interface to allow a biometric service provider (BSP) to manage the identification population for optimum performance.

It also provides primitives which allow the application to manage the capture of samples on a client, and the Enrollment, Verification, and Identification on a server.

The HRS is designed for use by both application developers and biometric technology developers. To make the integration of the technology as straightforward and simple as possible (thus enhancing its commercial viability), the approach taken is to hide or encapsulate to the extent possible the complexities of the biometric technology. This approach also serves to extend the generality of the interface to address a larger set of potential biometric technologies and applications.

HRS_CancelGUICallbacks

NAME

CSSM_HRS_CancelGUICallbacks, HRS_CancelGUICallbacks – Cancels GUICallbacks

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_CancelGUICallbacks
        (CSSM_HRS_HANDLE ModuleHandle);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_CancelGUICallbacks
        (CSSM_HRS_HANDLE ModuleHandle);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input) The handle of the attached HRS service provider.

DESCRIPTION

This function cancels GUICallbacks if they have been set. A GUICallback should be canceled before the service provider is detached.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
```

HRS_CancelStreamCallbacks

NAME

CSSM_HRS_CancelStreamCallbacks, HRS_CancelStreamCallbacks – Cancels the StreamCallback

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_CancelStreamCallbacks
        (CSSM_HRS_HANDLE ModuleHandle);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_CancelStreamCallbacks
        (CSSM_HRS_HANDLE ModuleHandle);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input) The handle of the attached HRS service provider.

DESCRIPTION

This function cancels the StreamCallback if it has been set. A StreamCallback should be canceled before the service provider is detached.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
```

HRS_Capture

NAME

CSSM_HRS_Capture, HRS_Capture – Captures samples

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_Capture
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_BIR_PURPOSE Purpose,
        CSSM_HRS_BIR_HANDLE_PTR CapturedBIR,
        sint32 Timeout,
        CSSM_HRS_BIR_HANDLE_PTR AuditData);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_Capture
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_BIR_PURPOSE Purpose,
        CSSM_HRS_BIR_HANDLE_PTR CapturedBIR,
        sint32 Timeout,
        CSSM_HRS_BIR_HANDLE_PTR AuditData);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`Purpose` (input)	A value indicating the purpose of the biometric data capture.
`CapturedBIR` (output)	A handle to a BIR containing captured data. This data is either an "intermediate" type BIR (which can only be used by either the `Process` or `CreateTemplate` functions, depending on the purpose), or a "processed" BIR (which can be used directly by `VerifyMatch` or `IdentifyMatch`, depending on the purpose).
`Timeout` (input)	An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, and no results. This value can be any positive number. A `?1` value means the service provider's default timeout value will be used.
`AuditData` (output/optional)	A handle to a BIR containing raw biometric data. This data may be used to provide human-identifiable data of the person at the device. If the pointer is NULL on input, no audit data is collected. Not all BSPs support the collection of audit data. A service provider may return a handle value of `CSSM_HRS_UNSUPPORTED_BIR_HANDLE` indicating not supported, or a value of `CSSM_HRS_INVALID_BIR_HANDLE` indicating no audit data is available.

DESCRIPTION

This function captures samples for the purpose specified, and returns either an "intermediate" type BIR (if the `Process` function needs to be called), or a "processed" BIR (if not). The `Purpose` is recorded in the header of the `CapturedBIR`. If `AuditData` is non-NULL, a BIR of type "raw" may be returned. The function returns handles to whatever data is collected, and all local operations can be completed through use of the handles.

If the application needs to acquire the data either to store it in a database or to send it to a server, the application can retrieve the data with the `HRS_GetBIRFromHandle()` function.

The application may request control of the GUI "look-and-feel" by providing a GUI callback pointer in `HRS_SetGUICallbacks()`.

`Capture` serializes use of the device. If two or more applications are racing for the device, the losers will wait until the timeout expires. This serialization takes place in all functions that capture data.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_PURPOSE_NOT_SUPPORTED
CSSMERR_HRS_TIMEOUT_EXPIRED
CSSMERR_HRS_TOO_MANY_HANDLES
CSSMERR_HRS_UNABLE_TO_CAPTURE
```

HRS_CreateTemplate

NAME

CSSM_HRS_CreateTemplate, HRS_CreateTemplate – Creates a new enrollment template from a BIR containing raw biometric data

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_CreateTemplate
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_HRS_INPUT_BIR *CapturedBIR,
        const CSSM_HRS_INPUT_BIR *StoredTemplate,
        CSSM_HRS_BIR_HANDLE_PTR NewTemplate,
        const CSSM_DATA *Payload);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_CreateTemplate
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_HRS_INPUT_BIR *CapturedBIR,
        const CSSM_HRS_INPUT_BIR *StoredTemplate,
        CSSM_HRS_BIR_HANDLE_PTR NewTemplate,
        const CSSM_DATA *Payload);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`CapturedBIR` (input)	The captured BIR or its handle.
`StoredTemplate` (input/optional)	Optionally, the template to be adapted, or its key in a database, or its handle.
`NewTemplate` (output)	A handle to a newly created template that is derived from the `CapturedBIR` and (optionally) the `StoredTemplate`.
`Payload` (input/optional)	A pointer to data that will be wrapped inside the newly created template. This parameter is ignored, if NULL.

DESCRIPTION

This function takes a BIR containing raw biometric data for the purpose of creating a new enrollment template. A new BIR is constructed from the `CapturedBIR`, and (optionally) it may perform an adaptation based on an existing `StoredTemplate`. The old `StoredTemplate` remains unchanged. If the `StoredTemplate` contains a payload, the payload is not copied into the `NewTemplate`. If the `NewTemplate` needs a payload, then that `Payload` must be presented as an argument to the function.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_BIR_SIGNATURE_FAILURE
CSSMERR_HRS_INCONSISTENT_PURPOSE
CSSMERR_HRS_INVALID_BIR
CSSMERR_HRS_PURPOSE_NOT_SUPPORTED
CSSMERR_HRS_RECORD_NOT_FOUND
CSSMERR_HRS_TOO_MANY_HANDLES
CSSMERR_HRS_UNABLE_TO_WRAP_PAYLOAD

HRS_DbClose

NAME

CSSM_HRS_DbClose, HRS_DbClose – Closes an open database

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbClose
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_DB_HANDLE DbHandle);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbClose
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_DB_HANDLE DbHandle);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`DbHandle` (input)	The DB handle for an open database managed by the service provider. This specifies the open database to be closed.

DESCRIPTION

This function closes an open database. All cursors currently set to the database are freed. The default database cannot be closed.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value `CSSM_OK` indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSSM_NOT_INITIALIZED`
`CSSMERR_CSSM_FUNCTION_FAILED`
`CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED`
`CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL`

HRS_DbCreate

NAME

CSSM_HRS_DbCreate, HRS_DbCreate – Creates and opens a new database

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbCreate
        (CSSM_HRS_HANDLE ModuleHandle,
        const uint8 *DbName,
        CSSM_HRS_DB_ACCESS_TYPE AccessRequest,
        CSSM_HRS_DB_HANDLE_PTR DbHandle);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbCreate
        (CSSM_HRS_HANDLE ModuleHandle,
        const uint8 *DbName,
        CSSM_HRS_DB_ACCESS_TYPE AccessRequest,
        CSSM_HRS_DB_HANDLE_PTR DbHandle);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`DbName` (input)	A pointer to the null-terminated string containing the name of the new database.
`AccessRequest` (input)	An indicator of the requested access mode for the database, such as read or write.
`DbHandle` (output)	The handle to the newly created and open data store. The value will be set to `CSSM_HRS_DB_INVALID_HANDLE` if the function fails.

DESCRIPTION

This function creates and opens a new database. The name of the new database is specified by the input parameter `DbName`. The newly created database is opened under the specified access mode.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value `CSSM_OK` indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_DATABASE_ALREADY_EXISTS
CSSMERR_HRS_INVALID_ACCESS_REQUEST
CSSMERR_HRS_INVALID_DATABASE_NAME

HRS_DbDelete

NAME

CSSM_HRS_DbDelete, HRS_DbDelete – Deletes all records from a database

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbDelete
        (CSSM_HRS_HANDLE ModuleHandle,
        const uint8 *DbName);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbDelete
        (CSSM_HRS_HANDLE ModuleHandle,
        const uint8 *DbName);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`DbName` (input)	A pointer to the null-terminated string containing the name of the database to be deleted.

DESCRIPTION

This function deletes all records from the specified database and removes all state information associated with that database.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value `CSSM_OK` indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_CSSM_NOT_INITIALIZED`
`CSSMERR_CSSM_FUNCTION_FAILED`
`CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED`
`CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL`
`CSSMERR_HRS_DATABASE_IS_OPEN`
`CSSMERR_HRS_INVALID_DATABASE_NAME`

HRS_DbDeleteBIR

NAME

CSSM_HRS_DbDeleteBIR, HRS_DbDeleteBIR – Deletes a BIR in an open database

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbDeleteBIR
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_DB_HANDLE DbHandle,
        const CSSM_GUID *KeyValue);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbDeleteBIR
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_DB_HANDLE DbHandle,
        const CSSM_GUID *KeyValue);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`DbHandle` (input)	The handle to the open database.
`KeyValue` (input)	The GUID of the BIR to be deleted.

DESCRIPTION

This function deletes the BIR identified by the `KeyValue` parameter in the open database identified by the `DbHandle` parameter.

If there is a cursor set to the deleted BIR, then the cursor is moved to the next sequential BIR (or set to the start of the database if there are no more records).

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value `CSSM_OK` indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_END_OF_DATABASE
CSSMERR_HRS_RECORD_NOT_FOUND
```

HRS_DbFreeCursor

NAME

CSSM_HRS_DbFreeCursor, HRS_DbFreeCursor – Frees memory and resources associated with a cursor

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbFreeCursor
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_DB_CURSOR_PTR Cursor);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbFreeCursor
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_DB_CURSOR_PTR Cursor);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`Cursor` (input)	The database `Cursor` to be freed.

DESCRIPTION

This function frees memory and resources associated with the specified `Cursor`.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value `CSSM_OK` indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_CURSOR_IS_INVALID
```

HRS_DbGetBIR

NAME

CSSM_HRS_DbGetBIR, HRS_DbGetBIR – Retrieves a BIR from an open database

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbGetBIR
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_DB_HANDLE DbHandle,
        const CSSM_GUID *KeyValue,
        CSSM_HRS_BIR_HANDLE_PTR RetrievedBIR,
        CSSM_HRS_DB_CURSOR_PTR Cursor);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbGetBIR
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_DB_HANDLE DbHandle,
        const CSSM_GUID *KeyValue,
        CSSM_HRS_BIR_HANDLE_PTR RetrievedBIR,
        CSSM_HRS_DB_CURSOR_PTR Cursor);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`DbHandle` (input)	The handle to the open database.
`KeyValue` (input)	The key into the database of the BIR to retrieve.
`RetrievedBIR` (output)	A handle to the retrieved BIR.
`Cursor` (output)	A handle that can be used to iterate through the database from the retrieved record.

DESCRIPTION

This function retrieves the BIR identified by the `Cursor` parameter. The BIR is copied into the service provider's storage, a handle to it is returned, and a pointer to the GUID that uniquely identifies the BIR in the database is returned. The `Cursor` is updated to the next record in the database, or to the first when the end of the database is reached.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_CURSOR_IS_INVALID
CSSMERR_HRS_END_OF_DATABASE

HRS_DbGetNextBIR

NAME

CSSM_HRS_DbGetNextBIR, HRS_DbGetNextBIR – Retrieves the BIR identified by the Cursor parameter

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbGetNextBIR
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_DB_CURSOR_PTR Cursor,
        CSSM_HRS_BIR_HANDLE_PTR RetrievedBIR,
        CSSM_GUID_PTR Guid);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbGetNextBIR
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_DB_CURSOR_PTR Cursor,
        CSSM_HRS_BIR_HANDLE_PTR RetrievedBIR,
        CSSM_GUID_PTR Guid);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`Cursor` (input/output)	A handle indicating which record to retrieve.
`RetrievedBIR` (output)	A handle to the retrieved BIR.
`Guid` (output)	The GUID that uniquely identifies the retrieved BIR in the database.

DESCRIPTION

This function retrieves the BIR identified by the `KeyValue` parameter in the open database identified by the `DbHandle` parameter.

The BIR is copied into the service provider's storage and a handle to it is returned. The `Cursor` is set to point to the next record, or the first record in the database if the retrieved BIR is the last.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value `CSSM_OK` indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_RECORD_NOT_FOUND

HRS_DbOpen

NAME

CSSM_HRS_DbOpen, HRS_DbOpen – Opens the data store

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbOpen
        (CSSM_HRS_HANDLE ModuleHandle,
        const uint8 *DbName,
        CSSM_HRS_DB_ACCESS_TYPE AccessRequest,
        CSSM_HRS_DB_HANDLE_PTR DbHandle,
        CSSM_HRS_DB_CURSOR_PTR Cursor);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbOpen
        (CSSM_HRS_HANDLE ModuleHandle,
        const uint8 *DbName,
        CSSM_HRS_DB_ACCESS_TYPE AccessRequest,
        CSSM_HRS_DB_HANDLE_PTR DbHandle,
        CSSM_HRS_DB_CURSOR_PTR Cursor);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`DbName` (input)	A pointer to the null-terminated string containing the name of the database.
`AccessRequest` (input)	An indicator of the requested access mode for the database, such as read or write.
`DbHandle` (output)	The handle to the opened data store. The value will be set to `CSSM_HRS_DB_INVALID_HANDLE` if the function fails.
`Cursor` (output)	A handle that can be used to iterate through the database.

DESCRIPTION

This function opens the data store with the specified name under the specified access mode. A database `Cursor` is set to point to the first record in the database.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_DATABASE_DOES_NOT_EXIST
CSSMERR_HRS_DATABASE_IS_LOCKED
CSSMERR_HRS_INVALID_ACCESS_REQUEST
CSSMERR_HRS_INVALID_DATABASE_NAME
CSSMERR_HRS_UNABLE_TO_OPEN_DATABASE

HRS_DbQueryBIR

NAME

CSSM_HRS_DbQueryBIR, HRS_DbQueryBIR – Returns a pointer to the GUID of a BIR in an open database

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbQueryBIR
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_DB_HANDLE DbHandle,
        const CSSM_HRS_INPUT_BIR *BIRToQuery,
        CSSM_GUID_PTR Guid);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbQueryBIR
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_DB_HANDLE DbHandle,
        const CSSM_HRS_INPUT_BIR *BIRToQuery,
        CSSM_GUID_PTR Guid);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`DbHandle` (input)	The handle to the open database.
`BIRToQuery` (input)	The BIR to be queried in the open database (either the BIR, its handle, or the key to the BIR in another open database).
`Guid` (output)	The GUID that uniquely identifies the BIR in the database.

DESCRIPTION

This function returns a pointer to the GUID of a BIR identified by the `BIRToQuery` parameter, if the BIR is in the open database identified by the `DbHandle` parameter. Otherwise, `CSSMERR_HRS_RECORD_NOT_FOUND` is returned.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value `CSSM_OK` indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_RECORD_NOT_FOUND

HRS_DbSetCursor

NAME

CSSM_HRS_DbSetCursor, HRS_DbSetCursor – Sets the cursor to point to a specified record in a database

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbSetCursor
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_DB_HANDLE DbHandle,
        const CSSM_GUID *KeyValue,
        CSSM_HRS_DB_CURSOR_PTR Cursor);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbSetCursor
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_DB_HANDLE DbHandle,
        const CSSM_GUID *KeyValue,
        CSSM_HRS_DB_CURSOR_PTR Cursor);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`DbHandle` (input)	A handle to the open database.
`KeyValue` (input)	The key into the database of the BIR to which the `Cursor` is to be set.
`Cursor` (output)	A handle that can be used to iterate through the database from the retrieved record.

DESCRIPTION

This function sets the cursor to point to the record indicated by the `KeyValue` in the database identified by the `DbHandle`. A NULL value sets the cursor to the first record in the database.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value `CSSM_OK` indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_RECORD_NOT_FOUND
```

HRS_DbStoreBIR

NAME

CSSM_HRS_DbStoreBIR, HRS_DbStoreBIR – Stores a BIR in an open database

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_DbStoreBIR
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_HRS_INPUT_BIR *BIRToStore,
        CSSM_HRS_DB_HANDLE DbHandle,
        CSSM_GUID_PTR Guid);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_DbStoreBIR
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_HRS_INPUT_BIR *BIRToStore,
        CSSM_HRS_DB_HANDLE DbHandle,
        CSSM_GUID_PTR Guid);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`BIRToStore` (input)	The BIR to be stored in the open database (either the BIR or its handle, or the index to it in another open database).
`DbHandle` (input)	The handle to the open database.
`Guid` (output)	A GUID that uniquely identifies the new BIR in the database. This GUID cannot be changed. To associate a different BIR with the user, it is necessary to delete the old one, store a new one in the database, and then replace the old GUID with the new one in the application account database.

DESCRIPTION

This function stores the BIR identified by the `BIRToStore` parameter in the open database identified by the `DbHandle` parameter. If the `BIRToStore` is identified by a BIR handle, the input BIR handle is freed. If the `BIRToStore` is identified by a database key value, the BIR is copied to the open database.

A new `GUID` is assigned to the new BIR in the database, and this `GUID` can be used as a key value to access the BIR later.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL

HRS_EnableEvents

NAME

CSSM_HRS_EnableEvents, HRS_EnableEvents – Enables the events from the attached HRS service provider in the current process

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_EnableEvents
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_MODULE_EVENT_MASK *Events);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_EnableEvents
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_MODULE_EVENT_MASK *Events);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`Events` (input)	A pointer to a mask indicating which events to enable.

DESCRIPTION

This function enables the events (indicated by the Events mask) from the attached HRS service provider in the current process. All other events from this HRS service provider are disabled for this process.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
```

HRS_Enroll

NAME

CSSM_HRS_Enroll, HRS_Enroll – Captures biometric data for the purpose of enrollment

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_Enroll
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_BIR_PURPOSE Purpose,
        const CSSM_HRS_INPUT_BIR *StoredTemplate,
        CSSM_HRS_BIR_HANDLE_PTR NewTemplate,
        const CSSM_DATA *Payload,
        sint32 Timeout
        CSSM_HRS_BIR_HANDLE_PTR AuditData);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_Enroll
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_BIR_PURPOSE Purpose,
        const CSSM_HRS_INPUT_BIR *StoredTemplate,
        CSSM_HRS_BIR_HANDLE_PTR NewTemplate,
        const CSSM_DATA *Payload,
        sint32 Timeout
        CSSM_HRS_BIR_HANDLE_PTR AuditData);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`Purpose` (input)	A value indicating the desired purpose of Enrollment.
`StoredTemplate` (input/optional)	Optionally, the BIR to be adapted, or its key in a database, or its handle.
`NewTemplate` (output)	A handle to a newly created template that is derived from the new raw samples and (optionally) the `StoredTemplate`.
`Payload` (input/optional)	A pointer to data that will be wrapped inside the newly created template. This parameter is ignored, if NULL.
`Timeout` (input)	An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, and no results. This value can be any positive number. A `?1` value means the BSP's default timeout value will be used.

AuditData (output/optional)	A handle to a BIR containing biometric audit data. This data may be used to provide human-identifiable data of the person at the device. If the pointer is NULL on input, no audit data is collected. Not all HRS service providers support the collection of audit data. An HRS service provider may return a handle value of CSSM_HRS_UNSUPPORTED_BIR_HANDLE to indicate AuditData is not supported, or a value of CSSM_HRS_INVALID_BIR_HANDLE to indicate that no audit data is available.

DESCRIPTION

This function captures biometric data from the attached device for the purpose of creating a ProcessedBIR for the purpose of enrollment. The Enroll function may be split between client and server if a streaming callback has been set. Either the client or the server can initiate the operation.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_PURPOSE_NOT_SUPPORTED
CSSMERR_HRS_RECORD_NOT_FOUND
CSSMERR_HRS_TIMEOUT_EXPIRED
CSSMERR_HRS_TOO_MANY_HANDLES
CSSMERR_HRS_UNABLE_TO_CAPTURE
CSSMERR_HRS_UNABLE_TO_WRAP_PAYLOAD

HRS_FreeBIRHandle

NAME

CSSM_HRS_FreeBIRHandle, HRS_FreeBIRHandle – Frees memory and resources associated with the BIR handle

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_FreeBIRHandle
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_BIR_HANDLE Handle);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_FreeBIRHandle
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_BIR_HANDLE Handle);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`Handle` (input)	The BIR handle to be freed.

DESCRIPTION

This function frees the memory and resources associated with the specified BIR handle. The associated BIR is no longer referenceable through that handle. If necessary, the application must make the BIR persistent either in an HRS-managed database or an application-managed database before freeing the handle.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
```

HRS_GetBIRFromHandle

NAME

CSSM_HRS_GetBIRFromHandle, HRS_GetBIRFromHandle – Retrieves the BIR associated with a BIR handle

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_GetBIRFromHandle
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_BIR_HANDLE Handle,
        CSSM_HRS_BIR_PTR *BIR);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_GetBIRFromHandle
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_BIR_HANDLE Handle,
        CSSM_HRS_BIR_PTR *BIR);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`Handle` (input)	The handle of the BIR to be retrieved.
`BIR` (output)	The retrieved BIR.

DESCRIPTION

This function retrieves the BIR associated with a BIR handle. The handle is invalidated. The HRS service provider allocates the storage for both the retrieved BIR structure and its data members, using the application's memory allocation callback function.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
```

HRS_GetHeaderFromHandle

NAME

CSSM_HRS_GetHeaderFromHandle, HRS_GetHeaderFromHandle – Retrieves the BIR header identified by handle

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_GetHeaderFromHandle
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_BIR_HANDLE Handle,
        CSSM_HRS_BIR_HEADER_PTR Header);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_GetHeaderFromHandle
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_BIR_HANDLE Handle,
        CSSM_HRS_BIR_HEADER_PTR Header);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`Handle` (input)	The handle of the BIR whose header is to be retrieved.
`Header` (output)	The header of the specified BIR.

DESCRIPTION

This function retrieves the BIR header identified by `handle`. The BIR handle is not freed by the HRS service provider.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
```

HRS_Identify

NAME

CSSM_HRS_Identify, HRS_Identify – Captures biometric data from the attached device and compares it against the Population

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_Identify
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_HRS_FAR *MaxFARRequested,
        const CSSM_HRS_FRR *MaxFRRRequested,
        const CSSM_BOOL *FARPrecedence,
        const CSSM_HRS_IDENTIFY_POPULATION *Population,
        CSSM_BOOL Binning,
        uint32 MaxNumberOfResults,
        uint32 *NumberOfResults,
        CSSM_HRS_CANDIDATE_ARRAY_PTR *Candidates,
        sint32 Timeout,
        CSSM_HRS_BIR_HANDLE_PTR AuditData);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_Identify
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_HRS_FAR *MaxFARRequested,
        const CSSM_HRS_FRR *MaxFRRRequested,
        const CSSM_BOOL *FARPrecedence,
        const CSSM_HRS_IDENTIFY_POPULATION *Population,
        CSSM_BOOL Binning,
        uint32 MaxNumberOfResults,
        uint32 *NumberOfResults,
        CSSM_HRS_CANDIDATE_ARRAY_PTR *Candidates,
        sint32 Timeout,
        CSSM_HRS_BIR_HANDLE_PTR AuditData);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input) — The handle of the attached HRS service provider.

`MaxFARRequested` (input) — The requested FAR criterion for successful verification.

`MaxFRRRequested` (input/optional) — The requested FRR criterion for successful verification. A NULL pointer indicates that this criterion is not provided.

FARPrecedence (input)	If both criteria are provided, this parameter indicates which takes precedence: CSSM_TRUE for FAR, CSSM_FALSE for FRR.
Population (input)	The population of Templates against which the Identify match is performed.
Binning (input)	A Boolean value indicating whether Binning is on or off. Binning is a search optimization technique that the BSP may employ. It is based on searching the population according to the intrinsic characteristics of the biometric data. While it may improve the speed of the Match operation, it may also increase the probability of missing a candidate.
MaxNumberOfResults (input)	Specifies the maximum number of match candidates to be returned as a result of the 1:N match. A value of zero is a request for all candidates.
NumberOfResults (output)	Specifies the number of candidates returned in the Candidates array as a result of the 1:N match.
Candidates (output)	A pointer to an array of CSSM_HRS_CANDIDATE_PTRs corresponding to the BIRs identified as a result of the match process (that is, indices associated with BIRs found to exceed the match threshold). This list is in rank order, with the highest scoring record being first. If no matches are found, this pointer will be set to NULL. If the Population was presented in a database, the IDs are database IDs; if the set was presented in an in-memory array, the IDs are indexes into the array.
Timeout (input)	An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, and no results. This value can be any positive number. A ?1 value means the service provider's default timeout value will be used.
AuditData (output/optional)	A handle to a BIR containing raw biometric data. This data may be used to provide human-identifiable data of the person at the device. If the pointer is NULL on input, no audit data is collected. Not all HRS service providers support the collection of audit data. A BSP may return a handle value of CSSM_HRS_UNSUPPORTED_BIR_HANDLE to indicate AuditData is not supported, or a value of CSSM_HRS_INVALID_BIR_HANDLE to indicate that no audit data is available.

DESCRIPTION

This function captures biometric data from the attached device, and compares it against the Population. The application must request a maximum FAR value criterion for a successful match, and may also (optionally) request a maximum FRR criterion for a successful match. If a maximum FRR value is provided, the application must also indicate via the FARPrecedence parameter, which criteria takes precedence.

The function returns a number of candidates from the population that match according to the specified criteria, and the FARAchieved and, optionally, the FRRAchieved are returned for each result in the Candidate array.

The Identify function may be split between client and server if a streaming callback has been set. Either the client or the server can initiate the operation.

NOTE Not all service providers support 1:N identification. See your service provider's programming manual for more details.

Depending on the service provider and the location and size of the database to be searched, this operation can take a significant amount of time to perform. Check your service provider's manual for recommended `Timeout` values.

The number of match candidates found by the service provider is dependent on the actual FAR used.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value `CSSM_OK` indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_BIR_SIGNATURE_FAILURE
CSSMERR_HRS_FUNCTION_NOT_SUPPORTED
CSSMERR_HRS_INCONSISTENT_PURPOSE
CSSMERR_HRS_NO_INPUT_BIRS
CSSMERR_HRS_RECORD_NOT_FOUND
CSSMERR_HRS_TIMEOUT_EXPIRED
CSSMERR_HRS_TOO_MANY_HANDLES
CSSMERR_HRS_UNABLE_TO_CAPTURE
```

HRS_IdentifyMatch

NAME

CSSM_HRS_IdentifyMatch, HRS_IdentifyMatch – Performs an identification (1-to-many) match between a ProcessedBIR and a set of stored BIRs

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_IdentifyMatch
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_HRS_FAR *MaxFARRequested,
        const CSSM_HRS_FRR *MaxFRRRequested,
        const CSSM_BOOL *FARPrecedence,
        const CSSM_HRS_INPUT_BIR *ProcessedBIR,
        const CSSM_HRS_IDENTIFY_POPULATION *Population,
        CSSM_BOOL Binning,
        uint32 MaxNumberOfResults,
        uint32 *NumberOfResults,
        CSSM_HRS_CANDIDATE_ARRAY_PTR *Candidates,
        sint32 Timeout);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_IdentifyMatch
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_HRS_FAR *MaxFARRequested,
        const CSSM_HRS_FRR *MaxFRRRequested,
        const CSSM_BOOL *FARPrecedence,
        const CSSM_HRS_INPUT_BIR *ProcessedBIR,
        const CSSM_HRS_IDENTIFY_POPULATION *Population,
        CSSM_BOOL Binning,
        uint32 MaxNumberOfResults,
        uint32 *NumberOfResults,
        CSSM_HRS_CANDIDATE_ARRAY_PTR *Candidates,
        sint32 Timeout);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`MaxFARRequested` (input)	The requested FAR criterion for successful verification.
`MaxFRRRequested` (input/optional)	The requested FRR criterion for successful verification. A NULL pointer indicates that this criterion is not provided.

FARPrecedence (input)	If both criteria are provided, this parameter indicates which takes precedence: CSSM_TRUE for FAR, CSSM_FALSE for FRR.
ProcessedBIR (input)	The BIR to be verified.
Population (input)	The population of BIRs against which the Identify match is performed.
Binning (input)	A Boolean indicating whether Binning is on or off. Binning is a search-optimization technique that the service provider may employ. It is based on searching the population according to the intrinsic characteristics of the biometric data. While it may improve the speed of the Match operation, it may also increase the probability of missing a candidate.
MaxNumberOfResults (input)	Specifies the maximum number of match candidates to be returned as a result of the 1:N match. A value of zero is a request for all candidates.
NumberOfResults (output)	Specifies the number of candidates returned in the Candidates array as a result of the 1:N match.
Candidates (output)	A pointer to an array of CSSM_HRS_CANDIDATE_PTRs corresponding to the BIRs identified as a result of the match process (that is, indices associated with BIRs found to exceed the match threshold). This list is in rank order, with the highest scoring record being first. If no matches are found, this pointer will be set to NULL. If the Population was presented in a database, the IDs are database IDs; if the set was presented in an in-memory array, the IDs are indexes into the array.
Timeout (input)	An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, and no candidate list. This value can be any positive number. A ?1 value means the service provider's default timeout value will be used.

DESCRIPTION

This function performs an identification (1-to-many) match between a ProcessedBIR and a set of stored BIRs. The ProcessedBIR is the "processed" BIR captured specifically for this identification. The population that the match takes place against can be presented in one of three ways:

- In a database identified by an open database handle
- Input in an array of BIRs
- In the "default" database of the BSP (possibly stored in the biometric device)

The application must request a maximum FAR value criterion for a successful match, and may also (optionally) request a maximum FRR criterion for a successful match. If a maximum FRR value is provided, the application must also indicate, via the FARPrecedence parameter, which criteria takes precedence. The FARAchieved and, optionally, the FRRAchieved are returned for each result in the Candidate array.

NOTE Not all service providers support 1:N identification.

Depending on the service provider and the location and size of the database to be searched, this operation can take a significant amount of time to perform.

The number of match candidates found by the service provider is dependent on the actual FAR used.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_BIR_NOT_FULLY_PROCESSED
CSSMERR_HRS_BIR_SIGNATURE_FAILURE
CSSMERR_HRS_FUNCTION_NOT_SUPPORTED
CSSMERR_HRS_INCONSISTENT_PURPOSE
CSSMERR_HRS_NO_INPUT_BIRS
CSSMERR_HRS_RECORD_NOT_FOUND
CSSMERR_HRS_TIMEOUT_EXPIRED

HRS_Import

NAME

CSSM_HRS_Import, HRS_Import – Imports non-realtime raw biometric data to construct a BIR

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_Import
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_DATA *InputData,
        CSSM_HRS_BIR_BIOMETRIC_DATA_FORMAT InputFormat,
        CSSM_HRS_BIR_PURPOSE Purpose,
        CSSM_HRS_BIR_HANDLE_PTR ConstructedBIR);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_Import
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_DATA *InputData,
        CSSM_HRS_BIR_BIOMETRIC_DATA_FORMAT InputFormat,
        CSSM_HRS_BIR_PURPOSE Purpose,
        CSSM_HRS_BIR_HANDLE_PTR ConstructedBIR);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`InputData` (input)	A pointer to image/stream data to import into a ProcessedBIR. The image/stream conforms to the standard identified by `InputFormat`.
`InputFormat` (input)	The format of the `InputData`.
`Purpose` (input)	A value indicating the Enroll purpose.
`ConstructedBIR` (output)	A handle to a BIR constructed from the imported biometric data. This BIR may be either an Intermediate or Processed BIR (as indicated in the header).

DESCRIPTION

This function imports non-realtime raw biometric data to construct a BIR for the purpose specified. `InputData` identifies the memory buffer containing the raw biometric data, while `InputFormat` identifies the form of the raw biometric data.

The function returns a handle to the `ConstructedBIR`.

If the application needs to acquire the BIR either to store it in a database or to send it to a server, the application can retrieve the data with the `HRS_GetBIRFromHandle()` function, or store it directly using `HRS_DbStoreBIR()`.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value `CSSM_OK` indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_FUNCTION_NOT_SUPPORTED
CSSMERR_HRS_PURPOSE_NOT_SUPPORTED
CSSMERR_HRS_TOO_MANY_HANDLES
CSSMERR_HRS_UNABLE_TO_IMPORT
CSSMERR_HRS_UNSUPPORTED_FORMAT
```

HRS_Process

NAME

CSSM_HRS_Process, HRS_Process – Processes the intermediate data captured via a call to HRS_Capture for the purpose of either verification or identification

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_Process
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_HRS_INPUT_BIR *CapturedBIR,
        CSSM_HRS_BIR_HANDLE_PTR ProcessedBIR);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_Process
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_HRS_INPUT_BIR *CapturedBIR,
        CSSM_HRS_BIR_HANDLE_PTR ProcessedBIR);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`CapturedBIR` (input)	The captured BIR or its handle.
`ProcessedBIR` (output)	A handle for the newly constructed "processed" BIR, NULL.

DESCRIPTION

This function processes the intermediate data captured via a call to `HRS_Capture()` for the purpose of either verification or identification. If the processing capability is in the attached service provider, a "processed" BIR is returned; otherwise, `ProcessedBIR` is set to NULL.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
```

```
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_BIR_SIGNATURE_FAILURE
CSSMERR_HRS_INCONSISTENT_PURPOSE
CSSMERR_HRS_INVALID_BIR
CSSMERR_HRS_PURPOSE_NOT_SUPPORTED
CSSMERR_HRS_RECORD_NOT_FOUND
CSSMERR_HRS_TOO_MANY_HANDLES
CSSMERR_HRS_UNABLE_TO_WRAP_PAYLOAD
```

HRS_SetGUICallbacks

NAME

CSSM_HRS_SetGUICallbacks, HRS_SetGUICallbacks – Allows the application to establish callbacks so that the application can control the "look-and-feel" of the biometric user interface

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_SetGUICallbacks
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_GUI_STREAMING_CALLBACK GuiStreamingCallback,
        void *GuiStreamingCallbackCtx,
        CSSM_HRS_GUI_STATE_CALLBACK GuiStateCallback,
        void *GuiStateCallbackCtx);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_SetGUICallbacks
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_GUI_STREAMING_CALLBACK GuiStreamingCallback,
        void *GuiStreamingCallbackCtx,
        CSSM_HRS_GUI_STATE_CALLBACK GuiStateCallback,
        void *GuiStateCallbackCtx);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`GuiStreamingCallback` (input)	A pointer to an application callback to deal with the presentation of biometric streaming data.
`GuiStreamingCallbackCtx` (input)	A generic pointer to context information provided by the application that will be presented on the callback.
`GuiStateCallback` (input)	A pointer to an application callback to deal with GUI state changes.
`GuiStateCallbackCtx` (input)	A generic pointer to context information provided by the application that will be presented on the callback.

DESCRIPTION

This function allows the application to establish callbacks so that the application can control the "look-and-feel" of the biometric user interface. Note that not all HRS service providers provide streaming data.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL

HRS_SetPowerMode

NAME

CSSM_HRS_SetPowerMode, HRS_SetPowerMode – Sets the device to the requested power mode

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_SetPowerMode
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_POWER_MODE PowerMode);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_SetPowerMode
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_POWER_MODE PowerMode);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`PowerMode` (input)	A 32-bit value indicting the power mode to which to set the device.

DESCRIPTION

This function sets the device to the requested power mode if the device supports it.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value `CSSM_OK` indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

`CSSMERR_HRS_FUNCTION_NOT_SUPPORTED`

HRS_SetStreamCallback

NAME

CSSM_HRS_SetStreamCallback, HRS_SetStreamCallback – Allows the application to establish a callback for client/server communication

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_SetStreamCallback
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_STREAM_CALLBACK StreamCallback,
        void *StreamCallbackCtx);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_SetStreamCallback
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_HRS_STREAM_CALLBACK StreamCallback,
        void *StreamCallbackCtx);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`StreamCallback` (input)	A pointer to an application callback to deal with the client/servertransmission of protocol data units between HRS service providers.
`StreamCallbackCtx` (input)	A generic pointer to context information provided by the application that will be presented on the callback.

DESCRIPTION

This function allows the application to establish a callback for client/server communication. The callback allows the HRS service provider to send a protocol message to its partner service provider, and to receive a protocol message in exchange.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL

HRS_StreamInputOutput

NAME

CSSM_HRS_StreamInputOutput, HRS_StreamInputOutput – Passes a protocol data unit into the HRS service provider and obtains a response

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_StreamInputOutput
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_DATA_PTR InMessage,
        CSSM_DATA_PTR OutMessage);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_StreamInputOutput
        (CSSM_HRS_HANDLE ModuleHandle,
        CSSM_DATA_PTR InMessage,
        CSSM_DATA_PTR OutMessage);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`InMessage` (input)	This parameter contains a protocol data unit from the partner HRS service provider.
`OutMessage` (output)	This parameter contains a protocol data unit to be sent back to the partner HRS service provider. If the parameter is NULL, there is no message to return.

DESCRIPTION

This function allows the application to pass a protocol data unit into the HRS service provider from the partner HRS service provider, and to obtain a response message to return to the partner.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
```

HRS_Verify

NAME

CSSM_HRS_Verify, HRS_Verify – Captures biometric data from the attached device and compares it against the StoredTemplate

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_Verify
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_HRS_FAR *MaxFARRequested,
        const CSSM_HRS_FRR *MaxFRRRequested,
        const CSSM_BOOL *FARPrecedence,
        const CSSM_HRS_INPUT_BIR *StoredTemplate,
        CSSM_HRS_BIR_HANDLE_PTR AdaptedBIR,
        CSSM_BOOL *Result,
        CSSM_HRS_FAR_PTR FARAchieved,
        CSSM_HRS_FRR_PTR FRRAchieved,
        CSSM_DATA_PTR *Payload,
        sint32 Timeout,
        CSSM_HRS_BIR_HANDLE_PTR AuditData);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_Verify
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_HRS_FAR *MaxFARRequested,
        const CSSM_HRS_FRR *MaxFRRRequested,
        const CSSM_BOOL *FARPrecedence,
        const CSSM_HRS_INPUT_BIR *StoredTemplate,
        CSSM_HRS_BIR_HANDLE_PTR AdaptedBIR,
        CSSM_BOOL *Result,
        CSSM_HRS_FAR_PTR FARAchieved,
        CSSM_HRS_FRR_PTR FRRAchieved,
        CSSM_DATA_PTR *Payload,
        sint32 Timeout,
        CSSM_HRS_BIR_HANDLE_PTR AuditData);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input) The handle of the attached HRS service provider.

`MaxFARRequested` (input) The requested FAR criterion for successful verification.

`MaxFRRRequested` (input/optional) The requested FRR criterion for successful verification. A NULL pointer indicates that this criterion is not provided.

FARPrecedence (input)	If both criteria are provided, this parameter indicates which takes precedence: CSSM_TRUE for FAR, CSSM_FALSE for FRR.
StoredTemplate (input)	The BIR to be verified against, its key in a database, or its handle.
AdaptedBIR (output/optional)	A pointer to the handle of the adapted BIR (Biometric Identification Record). This parameter can be NULL if an adapted BIR is not desired. Not all HRS service providers support the adaptation of BIRs. The function may return a handle value of CSSM_HRS_UNSUPPORTED_BIR_HANDLE to indicate that adaptation is not supported, or a value of CSSM_HRS_INVALID_BIR_HANDLE to indicate that adaptation was not possible.
Result (output)	A pointer to a Boolean value indicating (CSSM_TRUE/CSSM_FALSE) whether the BIRs matched or not, according to the specified criteria.
FARAchieved (output)	A pointer to an FAR value indicating the closeness of the match.
FRRAchieved (output/optional)	A pointer to an FRR value indicating the closeness of the match.
Payload (output/optional)	If the StoredTemplate contains a payload, it is returned in an allocated CSSM_DATA structure if the FARAchieved satisfies the policy of the service provider.
Timeout (input)	An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, and no results. This value can be any positive number. A ?1 value means the service provider's default timeout value will be used.
AuditData (output/optional)	A handle to a BIR containing raw biometric data. This data may be used to provide human-identifiable data of the person at the device. If the pointer is NULL on input, no audit data is collected. Not all service providers support the collection of audit data. An HRS service provider may return a handle value of CSSM_HRS_UNSUPPORTED_BIR_HANDLE to indicate AuditData is not supported, or a value of CSSM_HRS_INVALID_BIR_HANDLE to indicate that no audit data is available.

DESCRIPTION

This function captures biometric data from the attached device, and compares it against the StoredTemplate. The application must request a maximum FAR value criterion for a successful match, and may also (optionally) request a maximum FRR criterion for a successful match. If a maximum FRR value is provided, the application must also indicate via the FARPrecedence parameter, which criteria takes precedence. The Boolean Result indicates whether verification was successful or not, and the FARAchieved is a FAR value indicating how closely the BIRs actually matched.

The service provider may optionally return the corresponding FRR that was achieved through the FRRAchieved return parameter.

If the StoredTemplate contains a payload, the Payload may be returned upon successful verification. Optionally, a new AdaptedBIR may be constructed.

The Verify function may be split between client and server if a streaming callback has been set. Either the client or the server can initiate the operation.

If the match is successful, an attempt may be made to adapt the `StoredTemplate` with information taken from the ProcessedBIR. (Not all service providers perform adaptation.) The resulting `AdaptedBIR` should now be considered an optimal enrollment, and be saved in the enrollment database. It is up to the application whether or not it uses or discards this data. It is important to note that adaptation may not occur in all cases.

In the event of an adaptation, this function stores the handle to the new BIR in the memory pointed to by the `AdaptedBIR` parameter.

RETURN VALUE

A `CSSM_RETURN` value indicating success or specifying a particular error condition. The value `CSSM_OK` indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

```
CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_BIR_SIGNATURE_FAILURE
CSSMERR_HRS_INCONSISTENT_PURPOSE
CSSMERR_HRS_RECORD_NOT_FOUND
CSSMERR_HRS_TIMEOUT_EXPIRED
CSSMERR_HRS_TOO_MANY_HANDLES
CSSMERR_HRS_UNABLE_TO_CAPTURE
```

HRS_VerifyMatch

NAME

CSSM_HRS_VerifyMatch, HRS_VerifyMatch – Performs a verification (1-to-1) match between two BIRs – the ProcessedBIR and the StoredTemplate

SYNOPSIS

```
#include <hrs.h>
```

API

```
CSSM_RETURN CSSMAPI CSSM_HRS_VerifyMatch
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_HRS_FAR *MaxFARRequested,
        const CSSM_HRS_FRR *MaxFRRRequested,
        const CSSM_BOOL *FARPrecedence,
        const CSSM_HRS_INPUT_BIR *ProcessedBIR,
        const CSSM_HRS_INPUT_BIR *StoredTemplate,
        CSSM_HRS_BIR_HANDLE *AdaptedBIR,
        CSSM_BOOL *Result,
        CSSM_HRS_FAR_PTR FARAchieved,
        CSSM_HRS_FRR_PTR FRRAchieved,
        CSSM_DATA_PTR *Payload);
```

SPI

```
CSSM_RETURN CSSMHRI HRS_VerifyMatch
        (CSSM_HRS_HANDLE ModuleHandle,
        const CSSM_HRS_FAR *MaxFARRequested,
        const CSSM_HRS_FRR *MaxFRRRequested,
        const CSSM_BOOL *FARPrecedence,
        const CSSM_HRS_INPUT_BIR *ProcessedBIR,
        const CSSM_HRS_INPUT_BIR *StoredTemplate,
        CSSM_HRS_BIR_HANDLE *AdaptedBIR,
        CSSM_BOOL *Result,
        CSSM_HRS_FAR_PTR FARAchieved,
        CSSM_HRS_FRR_PTR FRRAchieved,
        CSSM_DATA_PTR *Payload);
```

LIBRARY

HRS Extensible Module Manager (`cdsa$inhrsemm_shr.exe`)

PARAMETERS

The parameter definitions are the same for the API and the SPI.

`ModuleHandle` (input)	The handle of the attached HRS service provider.
`MaxFARRequested` (input)	The requested FAR criterion for successful verification.
`MaxFRRRequested` (input/optional)	The requested FRR criterion for successful verification. A NULL pointer indicates that this criterion is not provided.

FARPrecedence (input)	If both criteria are provided, this parameter indicates which takes precedence: CSSM_TRUE for FAR, CSSM_FALSE for FRR.
ProcessedBIR (input)	The BIR to be verified, or its handle.
StoredTemplate (input)	The BIR to be verified against, its key in a database, or its handle.
AdaptedBIR (output/optional)	A pointer to the handle of the adapted BIR. This parameter can be NULL if an Adapted BIR is not desired. Not all service providers support the adaptation of BIRs. The function may return a handle value of CSSM_HRS_UNSUPPORTED_BIR_HANDLE to indicate that adaptation is not supported, or a value of CSSM_HRS_INVALID_BIR_HANDLE to indicate that adaptation was not possible.
Result (output)	A pointer to a Boolean value indicating (CSSM_TRUE/CSSM_FALSE) whether the BIRs matched or not according to the specified criteria.
FARAchieved (output)	A pointer to an FAR value indicating the closeness of the match.
FRRAchieved (output/optional)	A pointer to an FRR value indicating the closeness of the match.
Payload (output/optional)	If the StoredTemplate contains a payload, it is returned in an allocated CSSM_DATA structure if the FARAchieved satisfies the policy of the service provider.

DESCRIPTION

This function performs a verification (1-to-1) match between two BIRs—the ProcessedBIR and the StoredTemplate. The ProcessedBIR is the "processed" BIR constructed specifically for this verification. The StoredTemplate was created at enrollment. The application must request a maximum FAR value for a successful match, and may also (optionally) request a maximum FRR for a successful match. If a maximum FRR value is provided, the application must also indicate (via the FARPrecedence parameter) which one takes precedence. The Boolean Result indicates whether verification was successful or not, and the FARAchieved is a FAR value indicating how closely the BIRs actually matched.

The service provider may optionally return the corresponding FRR that was achieved, through the FRRAchieved return parameter.

By setting the AdaptedBIR pointer to non-NULL, the application can request that a BIR be constructed by adapting the StoredTemplate using the ProcessedBIR. A new handle is returned to the AdaptedBIR. If the StoredTemplate contains a Payload, the Payload may be returned upon successful verification if the FARAchieved is sufficiently stringent. This is controlled by the policy of the service provider.

If the match is successful, an attempt may be made to adapt the StoredTemplate with information taken from the ProcessedBIR. (Not all service providers perform adaptation.) The resulting AdaptedBIR should now be considered an optimal enrollment template, and be saved in the enrollment database. It is up to the application whether or not it uses or discards this data. It is important to note that adaptation may not occur in all cases.

In the event of an adaptation, this function stores the handle to the new BIR in the memory pointed to by the AdaptedBIR parameter.

RETURN VALUE

A CSSM_RETURN value indicating success or specifying a particular error condition. The value CSSM_OK indicates success. All other values represent an error condition.

ERRORS

Errors are described in the CDSA Technical Standard.

CSSMERR_CSSM_NOT_INITIALIZED
CSSMERR_CSSM_FUNCTION_FAILED
CSSMERR_CSSM_FUNCTION_NOT_IMPLEMENTED
CSSMERR_CSSM_FUNCTION_INTEGRITY_FAIL
CSSMERR_HRS_ADAPTATION_NOT_SUPPORTED
CSSMERR_HRS_BIR_NOT_FULLY_PROCESSED
CSSMERR_HRS_BIR_SIGNATURE_FAILURE
CSSMERR_HRS_INCONSISTENT_PURPOSE

A Open Source Notice

A.1 Intel Open Source License for CDSA/CSSM Implementation (BSD License with Export Notice)

IMPORTANT: READ BEFORE DOWNLOADING, COPYING, INSTALLING OR USING. By downloading, copying, installing or using the software you agree to this license. If you do not agree to this license, do not download, install, copy or use the software.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Intel Corporation nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL INTEL OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

EXPORT LAWS: THIS LICENSE ADDS NO RESTRICTIONS TO THE EXPORT LAWS OF YOUR JURISDICTION. It is licensee's responsibility to comply with any export regulations applicable in licensee's jurisdiction. This software is subject to the U.S. Export Administration Regulations and other U.S. law, and may not be exported or re-exported to certain countries (currently Afghanistan (Taliban-controlled areas), Cuba, Iran, Iraq, Libya, North Korea, Serbia (except Kosovo), Sudan and Syria) or to persons or entities prohibited from receiving U.S. exports (including Denied Parties, Specially Designated Nationals, and entities on the Bureau of Export Administration Entity List or involved with missile technology or nuclear, chemical or biological weapons).

Glossary

AAL *See Application Adaptation Layer (AAL).*

AC Authorization Computation service provider module. Synonymous with Authorization Computation Module (ACM).

Accountability A mechanism whereby the action of a user or a machine can be traced to that user or machine. A user's action may be audited and stored in a data bank called an audit trail. Subsequent searching of the audit trail can match events to the event instigator. In the commercial world, accountability is important to establish accurate billing procedures.

Application Adaptation Layer (AAL) An interface between CDSA and applications designed to use CDSA services.

API Application Programming Interface.

Asymmetric Algorithms Cryptographic algorithms using one key to encrypt and a second key to decrypt. They are often called public-key algorithms. One key is called the public key, and the other is called the private key or secret key.

Attach A process whereby an application obtains a service provider module handle, via an ATTACH call to CSSM. A service provider module can be a dynamic load module added at runtime on demand or a statically resident module.

Authentication A user or machine's identity must be established before establishing a connection to a computer. Authentication is the process of proving identity to the satisfaction of the permission-granting authority.

Authorization Permission for an entity to perform an action upon an object. Authorization is evaluated by a set of access control rules. Evaluation typically includes authentication of the requesting entity. The result of the evaluation should be conveyed to an agent that can enable the requested action upon the target object.

Bilateral authentication A scheme designed for two entities to establish trust in the identity and integrity of each other.

Biometric input The gathering of data from a personal, unique source, such as fingerprints, retina patterns, or human voice, for the purposes of verification or authorization.

BIR Biometric Identification Record.

BSAFE A cryptographic toolkit from RSA Data Security Incorporated.

BSP Biometric Service Provider.

CDSA *See Common Data Security Architecture (CDSA).*

Certificate A combination of an asymmetric public key and other identifying private information, which is digitally signed by a private key so it can be verified. *See also Digital certificate.*

Certificate Authority An entity that guarantees or sponsors a certificate. For example, a credit card company signs a cardholder's certificate to assure that the cardholder is who he or she claims to be. The credit card company is a certificate authority. Certificate authorities issue, verify, and revoke certificates.

Certificate chain The hierarchical chain of all other certificates used to sign the current certificate. This includes the Certificate Authority (CA) who signs the certificate, the CA who signed that CA's certificate, and so on. There is no limit to the depth of the certificate chain.

Certificate signing The Certificate Authority (CA) can sign certificates it issues or cosign certificates issued by another CA. In a general signing model, an object signs an arbitrary set of one or more objects. Hence, any number of signers can attest to an arbitrary set of objects. The arbitrary objects could be, for example, pieces of a document for libraries of executable code.

Certificate validity date A start date and a stop date for the validity of a certificate. If a certificate expires, the Certificate Authority (CA) may issue a new certificate.

Certification authority An entity that guarantees or sponsors a certificate. For example, a credit card company signs a cardholder's certificate to ensure that the cardholder is who he or she claims to be.

The credit card company is a certificate authority. Certificate authorities issue, verify, and revoke certificates.

CL Certificate Library service provider module. Synonymous with Certificate Library Module (CLM).

CRL Certificate Revocation List. An official list of certificates that are no longer valid.

Common Data Security Architecture (CDSA) A set of layered security services that address communications and data security problems in the emerging Internet and Intranet application space. CDSA consists of three basic layers:

- A set of system security services
- The Common Security Services Manager (CSSM)
- Add-in security modules (CSPs, TPs, CLs, DLs, ACs)

Common Security Services Manager (CSSM)

The central layer of the Common Data Security Architecture (CDSA) that defines the following service components:

- Cryptographic Services Manager
- Trust Policy Services Manager
- Certificate Library Services Manager
- Data Storage Library Services Manager
- Authorization Computation Manager
- Elective Module Manager
- Integrity Services Manager
- Security Context Manager

CSSM binds together all the security services required by applications. In particular, it facilitates linking digital certificates to cryptographic actions and trust protocols.

Confidentiality Information is revealed only to those who are authorized to see it. Confidentiality can be provided through an authorization and access control mechanism. It can also be provided through encryption and decryption operations, which limit data access to those who possess the cryptographic keys required to decrypt the information.

Cryptographic algorithm A method or defined mathematical process for implementing a cryptography operation. A Cryptographic algorithm may specify the procedure for encrypting and decrypting a byte stream, digitally signing an object, computing the hash of an object, or generating a random number.

Cryptographic Service Providers (CSPs)

Modules that provide secure key storage and cryptographic functions. The modules may be software only or hardware with software drivers. The cryptographic functions provided may include:

- Bulk encryption and decryption
- Digital signing
- Cryptographic hash
- Random number generation
- Key exchange

Cryptography The art and science of using mathematics to secure information and create a high degree of trust in the electronic media.

Cryptoki The name of the PKCS#11 Version 1.0 standard published by RSA Laboratories. The standard specifies the interface for accessing cryptographic services performed by a removable device. For additional information, refer to http://www.rsasecurity.com.

CSP *See Cryptographic Service Providers (CSPs).*

CSSM *See Common Security Services Manager (CSSM).*

Digital certificate The binding of some identification to a public key in a particular domain, as attested to directly or indirectly by the digital signature of the owner of that domain. A digital certificate is an unforgeable credential in cyberspace. The certificate is issued by a trusted authority and covered by that party's digital signature. The certificate may attest to the certificate holder's identity or may authorize certain

actions by the certificate holder. A certificate may include multiple signatures and may attest to multiple objects or multiple actions.

Digital signature A data block that was created by applying a cryptographic signing algorithm to some other data using a secret key. Digital signatures may be used to:

- Authenticate the source of a message, data, or document.
- Verify that the content of a message has not been modified since it was signed by the sender.
- Verify that a public key belongs to a particular person.

Typical digital signing algorithms include RSA signaturing and DSS, the Digital Signature Standard defined by NIST FIPS Pub 186.

DL Database Library service provider module.

EISL Embedded Integrity Services Library.

EMM Elective module manager: an extensibility mechanism in CDSA supporting the dynamic addition of new categories of service, beyond the basic set of Cryptographic Service Provider (CSP), Trust Policy (TP), Authorization Computation (AC), Certificate Library (CL), and Data Storage Library (DL).

ESW Electronic shrink-wrap. A term used to refer to an aggregate collection of data files identified by a manifest or bill of materials.

FAR False Accept Rate: the probability that biometric data samples are falsely decided by the HRS as matching; that is, they should not match, but do.

FRR False Reject Rate: the probability that biometric data samples are falsely decided by the HRS as not matching; that is, they should match, but do not.

Generic Cryptographic Services (GCS) A set of services and associated APIs designed to provide key-based cryptographic operations to applications. GCS predates CDSA. GCS requirements were based on early hardware-based cryptographic devices where cryptographic keys were retained within the device. Some Internet applications require the secured transmission of cryptographic keys. The CDSA Cryptographic Service APIs accommodate both types of requirements.

Generic Security Services (GSS) A set of services and associated APIs defined by the International Engineering Task Force (IETF). The defined APIs support concurrent applications in authenticating each other, delegating rights and privileges to each other, and using confidentiality and integrity verification services to secure communications between the applications.

GUI Graphical User Interface.

GUID Globally unique identifier.

Hash algorithm A cryptographic algorithm used to compress a variable-size input stream into a unique, fixed-size output value. The function is one-way, meaning the input value cannot be derived from the output value. A cryptographically strong hash algorithm is collision-free, meaning unique input values produce unique output values. Hashing is typically used in digital signing algorithms. Example hash algorithms include MD and MD2 from RSA Data Security. MD5, also from RSA Data Security, hashes a variable-size input stream into a 128-bit output value. SHA, a Secure Hash Algorithm published by the U.S. Government, produces a 160-bit hash value from a variable-size input stream.

HRS Human Recognition Services. HRS is a CSSM Elective Module Manager intended to provide a high-level generic authentication model suited for any form of human authentication. Particular emphasis has been made in the design on its suitability for authentication using biometric technology.

IBIA International Biometric Industry Association.

Integrity Information is said to have integrity if that data has not been modified or altered since the point in time when an authorized agent intended the data to be static. Information integrity is important for all data types including authorization data and authentication credentials.

Key Management Public-private key pairs are items that need to be securely managed. A key may be lost, stolen, or compromised. If this happens, the

key (and in fact, the key pair) must be nulled. Whatever task the key was used for, a new key must be issued and used. In the case of the lost key, a duplicate should be available. If not, the data protected by the lost key may itself be lost. The null public key must be advertised as invalid. It will be listed in a data bank called a revocation list. The new public key must be distributed to those entitled to have it.

Leaf certificate The certificate in a certificate chain that has not been used to sign another certificate in that chain. The leaf certificate is signed directly or transitively by all other certificates in the chain.

Manifest A digital signature of a file, created using certificates. The digital signature takes the form of a separate file called a manifest. The manifest contains the encrypted digest of the target file and the X509 certificates of the signers. This data is sufficient to guarantee the identity of the signer of a file and the authenticity of the file's contents.

MDS *See Module Directory Services (MDS).*

Message Digest The digital fingerprint of an input stream. A cryptographic hash function is applied to an input message of arbitrary length and returns a fixed-size output, which is called the digest value.

Meta-information Descriptive information specified by a service provider module and stored in MDS. This information advertises the module's services. CSSM supports application queries for this information. The information may change at runtime.

Module Directory Services (MDS) A platform-independent registration service for managing executable code modules and their associated signed integrity credentials.

Nonce A nonrepeating value, usually but not necessarily random.

OID Object identifier.

Owned certificate A certificate whose associated private key resides in a local CSP. Digital signature algorithms require the private key when signing data. A system may supply certificates it owns along with signed data to allow others to verify the signature. A system uses certificates that it does not own to verify signatures created by others.

Payload Data wrapped inside biometric data for release to an application on successful verification of authenticity of a user. This can be any data that is useful to an application.

PIN Personal Identification Number.

PKI *See Public Key Infrastructure (PKI).*

Private key The cryptographic key used to decipher or sign messages in public-key cryptography. This key is kept secret by its owner.

Public key The cryptographic key used to encrypt messages in public-key cryptography. The public key is available to multiple users (for example, the public).

Public Key Infrastructure (PKI) The agreed infrastructure, ultimately to be applied worldwide, in which secure electronic business (eCommerce, banking, legal transactions) and secure electronic welfare (medical welfare, state and government provision for pensions, social security, and so forth) can function securely using the private-public key method of cryptography.

PVC Pointer validation checking.

Random number generator A function that generates cryptographically strong random numbers that cannot be easily guessed by an attacker. Random numbers are often used to generate session keys.

Root certificate The prime certificate, such as the official certificate of a corporation or government entity. The root certificate is positioned at the top of the certificate hierarchy in its domain, and it guarantees the other certificates in its certificate chain. The root certificate's public key is the foundation of signature verification in its domain.

RSA RSA Data Security, Incorporated, Bedford, MA. Producers of the BSAFE toolkit.

Secret key A cryptographic key used with symmetric algorithms, usually to provide confidentiality.

Secure Electronic Transaction (SET) A specification designed to utilize technology for authenticating the parties involved in payment card purchases on any type of online network, including the Internet. SET focuses on maintaining confidentiality of information, ensuring message integrity, and authenticating the parties involved in a transaction. More information about SET is available at: http://www.setco.org/. *See also Secure Sockets Layer (SSL).*

Secure Sockets Layer (SSL) Also known as Above Transport Layer Security (TLS). A security protocol that prevents eavesdropping, tampering, or message forgery over the Internet. An SSL service negotiates a secure session between two communicating endpoints. Basic facilities include certificate-based authentication, end-to-end data integrity, and optional data privacy. SSL has been submitted to the IETF as an Internet Draft for Transport Layer Security (TLS).

Security context A control structure that retains state information shared between a cryptographic service provider and the application agent requesting service from the CSP. A security context specifies CSP and application-specific values, such as required key length and desired hash functions.

Security infrastructure An agreed infrastructure for the security of all electronic data transfer. Such an infrastructure would, in theory, lessen the need for organizations to construct trust domains. An international security infrastructure would facilitate the creation of a secure Internet. Presently, global efforts are more focussed on an architecture for Public Key Infrastructure, seen by many as the blueprint for the infrastructure that will facilitate eCommerce.

Security perimeter A conceptual perimeter or boundary of a computer system or local area network inside which the security is at a known level of competence. If data is required to cross this perimeter, it is prudent to pass all such data through a firewall.

Security-relevant event An event where a CSP-provided function is performed, a security service provider module is loaded, or a breach of system security is detected.

Security risk assessment An exercise performed by specialists to assess how vulnerable an enterprise is to various forms of security attack. The ideal outcome from this exercise is a recommended range of security measures, hardware, software, and procedural, which give a level of protection commensurate with the value of the assets that need to be protected.

Session key A cryptographic key used to encrypt and decrypt data. The key is shared by two or more communicating parties, who use the key to ensure privacy of the exchanged data.

SET *See Secure Electronic Transaction (SET).*

Signature *See Digital Signature.*

Signature chain The hierarchical chain of signers, from the root certificate to the leaf certificate, in a certificate chain.

Signing and sealing The electronic equivalent to the handwritten signature and the secure strong room. Precise ways of performing these actions may vary, but signing by digital signature and sealing (for transport or storage) by encryption is evolving towards internationally agreed protocols which will be acceptable to the commercial world, the legal profession, and governments.

Single sign-on A mechanism whereby a single action of user authentication and authorization can permit a user to access all computers and systems where he has access permission, without the need to enter multiple passwords. Single sign-on reduces human error, a major component of systems failure.

SmartCard A card of the same dimensions as the magnetic-stripe credit card, but containing processing ability and memory storage space. Because the card can contain storage credentials and cryptographic keys and perform encryption/decryption operations, its power as a tamper-proof personal token for authentication makes it very attractive to a whole range of computer applications.

SPI Service provider interface.

SPKI Simple public key infrastructure. Information about SPKI can be found at http://www.ietf.org/html.charters/spki-charter.html.

SSL *See Secure Sockets Layer (SSL).*

SSLeay A free implementation of the Secure Sockets Layer. *See also Secure Sockets Layer (SSL).*

Symmetric algorithms Cryptographic algorithms that use a single secret key for encryption and decryption. Both the sender and receiver must know the secret key. Well known symmetric functions include DES (Data Encryption Standard) and IDEA. DES was endorsed by the U.S. Government as a standard in 1977. It's an encryption block cipher that operates on 64-bit blocks with a 56-bit key. It is designed to be implemented in hardware, and works well for bulk encryption. IDEA (International Data Encryption Algorithm) uses a 128-bit key.

Token The logical view of a cryptographic device, as defined by a CSP's interface. A token can be hardware, a physical object, or software. A token contains information about its owner in digital form and about the services it provides for electronic-commerce and other communication applications. A token is a secure device. It may provide a limited or a broad range of cryptographic functions. Examples of hardware tokens are SmartCards and PMCIA cards.

TP Trust Policy service provider module. Synonymous with Trust Policy Module (TPM).

Trust domains A designated virtual area that has a known and accepted level of security, and thus a known and accepted level of trust. A local area network is an example of a domain that is likely to be trusted. Domains may be geographically wide ranging, and may be made up of subdomains. A domain is only as trustworthy as its weakest component.

Verification A process performed to check the integrity of a message, to determine the sender of a message, or both. Different algorithms are used to support different modes of verification.

A typical procedure supporting integrity verification is the combination of a one-way hash function and a reversible digital signaturing algorithm. A one-way hash of the message is computed. The hash value is signed by encrypting it with a private key. The message and the encrypted hash value are sent to a receiver. The recipient recomputes the one-way hash, decrypts the signed hash value, and compares it with the computed hash. If the values match, then the message has not been tampered since it was signed.

The identity of a sender can be verified by a challenge-response protocol. The recipient sends the message sender a random challenge value. The original sender uses its private key to sign the challenge value and returns the result to the receiver. The receiver uses the corresponding public key to verify the signature over the challenge value. If the signature is valid, the sender is the holder of the private key. If the receiver can reliably associate the corresponding public key with the named/known entity, then the identity of the sender is said to have been verified.

Web of trust A trust network among people who know and communicate with each other. Digital certificates are used to represent entities in the web of trust. Any pair of entities can determine the extent of trust between the two, based on their relationship in the web.

X509v3 certificate This standard defines the contents and structure of a digital certificate. The specification is ITU-T Recommendation X.509, Data Networks and Open System Communications Directory: Authentication Framework, 06/97. This certificate format constitutes a widely accepted basis for a public key infrastructure. To support the PKI, certificates of this form are digitally signed and issued by certification authorities (CAs).

D

E

U

V

W