z/OS

Open Cryptographic Services Facility Application Programming

Version 2 Release 1

SC14-7513-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 295.

This edition applies to Version 2 Release 1 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

Preface

The Open Cryptographic Services Facility (OCSF) is a derivative of the IBM Keyworks technology which is an implementation of the Common Data Security Architecture (CDSA) for applications running in the UNIX Services environment.

Recently cryptography has come into widespread use in meeting multiple security needs, such as confidentiality, integrity, authentication, and non-repudiation. In order to address these requirements in the emerging Internet, Intranet, and Extranet application domains, the CDSA was developed. The OCSF is a comprehensive set of layered security services. The OCSF focuses on security in peer-to-peer, store-and-forward, and archival applications. It is designed to be compliant with industry standards such as OpenGroup, and is applicable to a broad range of hardware and operating system platforms. OCSF is intended to include full life cycle key management and portable credentials. The definition of such a set of layered security services and an open architecture protects the investment made in implementation of security applications by facilitating the reuse of core components of the architecture for different products.

The security services available in the OCSF are defined by the categories of service provider modules that the architecture accommodates. These service providers are:
- Cryptographic Services[1]
- Trust Policy Libraries
- Certificate Libraries
- Data Storage Libraries.

OCSF Architecture

The OCSF Architecture consists of a set of layered security services and associated programming interfaces designed to furnish an integrated set of information and communication security capabilities. Each layer builds on the more fundamental services of the layer directly below it.

These layers start with fundamental components such as cryptographic algorithms, random numbers, and unique identification information in the lower layers, and build up to digital certificates, key management and recovery mechanisms, and secure transaction protocols in higher layers. The OCSF Architecture is intended to be the multiplatform security architecture that is both horizontally broad and vertically robust.

Figure 1 on page xiv shows a simplified view of the layered architecture of an OCSF-based system. There are four major layers in the OCSF Architecture: Application Domains, System Security Services, OCSF Framework, and Service Providers.

The Application Domains layer implements the application domain services, such as Secure Electronic Transaction (SET) and E-Wallet, E-mail services, or file archival services. The System Security Services layer is between the Application Domains layer and the OCSF Framework layer. It implements security protocols that are

1. If you want to provide a Cryptographic Service Provider, you need to contact IBM. For more information, see the *z/OS Open Cryptographic Services Facility Service Provider Module Developer's Guide and Reference*.

used by the Application Domains layer. Software at this layer may implement cryptographic system security services such as Secure Sockets Layer (SSL), Internet Protocol Security (IPSEC), Secure/Multipurpose Internet Mail Extensions (S/MIME) and Electronic Data Interchange (EDI). The System Security Services layer also includes tools and utilities for installing, configuring, and maintaining the OCSF Framework and service provider modules.

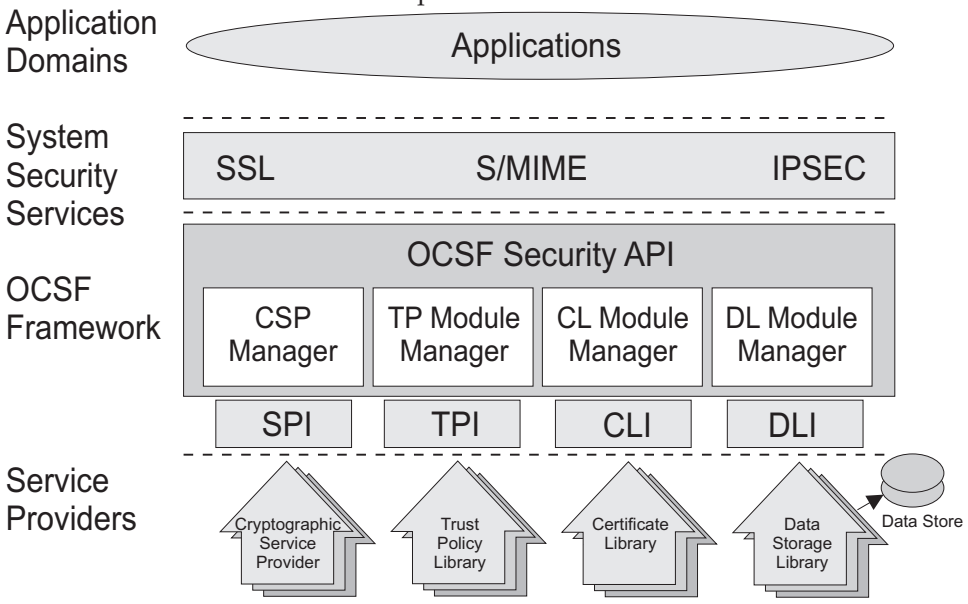

Figure 1. Open Cryptographic Services Facility Architecture.

The OCSF Framework is the central component of this extensible architecture that provides mechanisms to dynamically manage service provider modules. The OCSF Framework defines a common security application programming interface (API) that must be used to access services of service provider modules. Applications request security services through the OCSF security API or through system security services implemented over the OCSF API. The service provider modules actually perform the requested security services. IBM provides a number of service provider modules. Additional service provider modules may be available from other Independent Software Vendors (ISVs) and hardware vendors. Applications may direct their requests to modules from specific vendors or to any module that performs the required services.

Who should use this information

This information provides an overview of the OCSF for ISVs who develop their own operating systems or other security products either as complete applications or as service providers to extensible platforms. This information is intended for use by:

- Security application programmers
- Security provider module developers that need to use the services of other service providers
- Experienced software designers
- Security architects who work in high end cryptography
- Sophisticated integrators familiar with numerous forms of network computing
- Vendors of customizable service providers for cryptographic, trust, and database services.

This audience understands the requirements for a ubiquitous security infrastructure upon which they can build security-aware application products.

Requirements

The software required to develop applications using the OCSF include the z/OS C/C++ Compiler and runtime library. You need to have the z/OS SecureWay Security Server's RACF (or equivalent security product). See Chapter 1, “Configuring and Getting Started,” on page 1 for the RACF settings that you need.

Conventions used in this information

This information uses these typographic conventions:

Bold **Bold** words or characters represent system elements that you must enter into the system literally, such as commands.

Italic *Italicized* words or characters represent values for variables that you must supply.

`Example Font`
Examples and information displayed by the system are printed using an example font that is a `constant width typeface`.

Where to Find More Information

Where necessary, this information references information in other books. For complete titles and order numbers for all elements of z/OS, see the *z/OS Information Roadmap*.

This information provides an overview of the OCSF. It explains how to integrate OCSF into applications and contains a sample OCSF application. It also defines the interfaces that application developers employ to access security services provided by the OCSF framework and service provider modules. Specific information about the individual service providers is also provided.

The *z/OS Open Cryptographic Services Facility Service Provider Module Developer's Guide and Reference* describes the features common to all OCSF service provider modules. It defines the interfaces for certificate, trust, and data library service providers. Service provider developers must conform to these interfaces in order for the individual service provider modules to be accessible through the OCSF framework.

Internet Sources

The softcopy z/OS publications are also available for web-browsing and for viewing or printing PDFs using the following URL:

`http://www.ibm.com/systems/z/os/zos/bkserv/`

How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html).
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRCFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   US
4. Fax the comments to us, as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:

- Your name and address.
- Your email address.
- Your telephone or fax number.
- The publication title and order number:
  z/OS OCSF Application Programming
  SC14-7513-00
- The topic and page number that is related to your comment.
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:

- Contact your IBM service representative.
- Call IBM technical support.
- Visit the IBM Support Portal at z/OS support page (http://www.ibm.com/systems/z/support/).

z/OS Version 2 Release 1 summary of changes

See the following publications for all enhancements to z/OS Version 2 Release 1 (V2R1):

- *z/OS Migration*
- *z/OS Planning for Installation*
- *z/OS Summary of Message and Interface Changes*
- *z/OS Introduction and Release Guide*

Chapter 1. Configuring and Getting Started

Note: You must reinstall and run the configuration scripts with every new release of z/OS.

Chapter 1, "Configuring and Getting Started" describes the procedures that you perform after you have completed code installation. The three additional steps include:

1. **Setting up the necessary security authorizations**

   This step provides the information needed to set up the RACF® Facility Class profiles needed by the Open Cryptographic Services Facility (OCSF). These classes must be set up and z/OS user identities defined to that class before applications using CDSA can be run by those z/OS users. See "Setting Up the Necessary Security Authorizations."
2. **Running the installation script**

   This step installs the individual service providers to OCSF. See "Running the Installation Script" on page 5.
3. **Running the Installation Verification Procedure**

   After you have completed the previous steps, run the Installation Verification Procedure to verify that you have installed and configured your system correctly. See "Running the Installation Verification Procedure" on page 5.

Setting Up the Necessary Security Authorizations

The OCSF implementation of the Common Data Security Architecture (CDSA) for z/OS uses the z/OS SecureWay Security Server's RACF (or an equivalent security product) to authorize the use of its services. The OCSF services are intended to be used by z/OS UNIX System Services based application servers or daemons.

Security Administration

In order to use OCSF services this administration must be done:

- OCSF-related RACF facility class profiles need to be defined and the FACILITY class made active if it is not already active.
- All of the programs, modules and DLLs loaded into the OCSF daemon application address space must be defined as program controlled. Programs or modules loaded from the traditional z/OS search order (that is, STEPLIB, LINKLIST, and so forth) need to reside in program-controlled libraries. Programs loaded from the UNIX file system must have the program-controlled extended attribute.
- OCSF application (daemon) user IDs must be defined to RACF and permitted to the OCSF facility class profiles. Depending on whether your system is operating with z/OS UNIX security or UNIX security, these user IDs will also need to be permitted to the BPX.SERVER facility class profile (when z/OS UNIX security is in effect), or the OCSF daemon application must run with an effective UID of 0 (when UNIX security is in effect). Refer to *z/OS UNIX System Services Planning* for definitions of z/OS UNIX security and UNIX security.

See "RACF FACILITY Class Profiles Required by OCSF" on page 2 for details on performing these administrative steps.

RACF FACILITY Class Profiles Required by OCSF

The use of OCSF services is controlled by the RACF facility class profiles:

- **CDS.CSSM** - Authorizes the daemon to call OCSF services **RDEFINE FACILITY CDS.CSSM UACC (NONE)**
- **CDS.CSSM.CRYPTO** - Authorizes the daemon to call a Cryptographic Service Provider (CSP) **RDEFINE FACILITY CDS.CSSM.CRYPTO UACC (NONE)**
- **CDS.CSSM.DATALIB** - Authorizes the daemon to call a Data Library (DL) Service Provider **RDEFINE FACILITY CDS.CSSM.DATALIB UACC (NONE)**

You need to define these profiles, using the **RDEFINE** command as shown, before running any OCSF application, or before running the OCSF installation script described in "Running the Installation Script" on page 5. If these facility class profiles are not defined, OCSF services are unavailable.

An OCSF application, and the OCSF installation script described in "Running the Installation Script" on page 5, must execute under the security context of a user identity that has been granted READ access to the OCSF facility class profiles.

Program Control

Program control is the concept of having "trusted" applications. Installations can define libraries to RACF where these trusted applications will reside. When program control is active on a system, processes will be marked "dirty" (by means of the **SETROPTS WHEN(PROGRAM)** command) if they attempt to load programs from libraries that are not trusted. z/OS UNIX System Services also has the concept of trusted applications. In the UNIX file system, executable files may be tagged with the program-controlled extended attribute. If a user issues a shell command, or runs a program that does not have the program-controlled extended attribute, the process becomes dirty. In either case the process is never "cleaned". The dirty bit remains on, causing certain services to fail as a result. Refer to the *z/OS Security Server RACF Security Administrator's Guide* for more information on Program Control.

Program Control in RACF

The purpose of protecting load modules is to provide installations with the ability to control who can execute what programs and to treat those programs as assets. You protect individual load modules (programs) by creating a profile for the program in the PROGRAM general resource class. A program protected by a profile in the PROGRAM class is called a controlled program. OCSF services utilize other elements of z/OS. If RACF Program Control is activated, these program libraries must also be program controlled:

- C/C++ Runtime Libraries
- Language Environment libraries
- ICSF libraries (if ICSF is used)
- System SSL.

For example, if you have a load library called `MYLOADLIB` residing in `SYS1.XYZ` you would have to issue this RACF command to make it program controlled:

```
REDEFINE PROGRAM MYLOADLIB ADDMEM('SYS1.XYZ')
```

If a discrete profile for the dataset already exists but program control is not enabled in this profile then this command should be issued:

```
ralt program * addmem('dataset.name') uacc(read)
```

Then you can activate that profile by issuing this RACF command:

```
SETROPTS WHEN(PROGRAM) REFRESH
```

Refer to the *z/OS Security Server RACF Security Administrator's Guide*, SC28-1915, for more information on Program Control.

HFS Program Control

You can mark programs and dynamically-loaded libraries (DLLs) in the UNIX file system as controlled (trusted) by turning on the program-controlled extended attribute for the HFS file containing the program or DLL. To turn this extended attribute on, issue:

```
extattr +p filename
```

You can check if a file has the program-controlled extended attribute by using the UNIX shell **ls** command with the **-E** option. This example shows using **ls -E** to verify that the program-controlled attribute is set for one of the OCSF DLLs:

```
$ cd /usr/lpp/ocsf/lib
$ ls -E cssm32.dll
-rwxr-xr-x aps 2 ROOT   SYS1    737280 Nov 3 22:07 cssm32.dll
```

The "p" flag in the second column of the **ls** command output indicates that this file does have the program-controlled extended attribute.

APF Authorization

The SMP/E installation of OCSF now turns on the APF-authorized extended attribute for the OCSF libraries in the **/usr/lpp/ocsf/lib** and **/usr/lpp/ocsf/addins** directories. You can verify this by issuing the UNIX shell **ls** command with the **-E** option as shown in the example:

```
$ cd /usr/lpp/ocsf/lib
$ ls -E cssm32.dll
-rwxr-xr-x aps 2 ROOT   SYS1    737280 Nov 3 22:07 cssm32.dll
```

The "a" flag in the second column of the **ls** command output indicates that this file does have the APF-authorized extended attribute.

Note: OCSF can only be accessed from program state (key 8).

Refer to *z/OS UNIX System Services Planning* for more details.

OSCF User Identities and Permissions

In order to use the services offered by OCSF for z/OS, an OCSF application, as well as the OCSF installation script described on "Running the Installation Script" on page 5, must execute under a z/OS user identity that has been granted READ access to the OCSF CDS.* facility class profiles described (see "RACF FACILITY Class Profiles Required by OCSF" on page 2). These RACF profiles control which user IDs are authorized to use OCSF services.

In addition, OCSF applications, as well as the OCSF installation script on "Running the Installation Script" on page 5, require an additional permission or authority, the nature of which depends on whether your system is operating with z/OS UNIX security, or UNIX security:

- If either the BPX.SERVER or the BPX.DAEMON facility class profile has been defined, then your system is operating with z/OS UNIX security. In this case, the user identity associated with an OCSF application must be granted READ

access to the BPX.SERVER facility class profile. This profile controls the use of the z/OS services used by OCSF to determine access authority. If this profile has not been previously defined on your system, use the **RDEFINE** command to define it.

- If neither the BPX.SERVER nor the BPX.DAEMON facility class profiles have been defined, then your system is operating with UNIX security. In this case, an OCSF application must be run with an effective UID of 0 (super user).

Refer to *z/OS UNIX System Services Planning* for more information comparing z/OS UNIX security and UNIX security.

It is recommended that unique z/OS and UNIX identities (UIDs) be assigned to daemon applications that are authorized to use OCSF services to maintain individual accountability of which applications are cryptographic services on z/OS.

For example, assume that a daemon application needs to use OCSF services on z/OS. The daemon application is assigned the unique UID of 25, and has been associated with the daemon process with a RACF identity of G123456. This daemon's home directory is **/u/apps/g123456**. This daemon runs under the z/OS shell, and the application is started by the daemon's .profile.

Create a RACF user profile, with an OMVS segment using the **RACF ADDUSER** command:

```
ADDUSER G123456 OMVS(UID(25) HOME('/u/apps/g123456') program('/bin/sh'))
```

Refer to the and to the *z/OS Security Server RACF Security Administrator's Guide* for more information.

Granting Permission to Use OCSF Service

These authorizations need to be made for the daemon to use CDSA services.

- Authorize the daemon to the required class profiles in the RACF FACILITY CLASS by issuing the **RACF PERMIT** commands:

  ```
  PERMIT CDS.CSSM CLASS(FACILITY) ID(G123456) ACC(READ)
  PERMIT CDS.CSSM.CRYPTO CLASS(FACILITY) ID(G123456) ACC(READ)
  PERMIT CDS.CSSM.DATALIB CLASS(FACILITY) ID(G123456) ACC(READ)
  ```

- Assuming that the system operates with z/OS UNIX security, authorize the daemon to the class profile BPX.SERVER in the RACF FACILITY CLASS by issuing the **RACF PERMIT** commands:

  ```
  PERMIT BPX.SERVER CLASS(FACILITY) ID(G123456) ACC(READ)
  ```

You may need to authorize the daemon user ID to other profiles depending on the other requirements of the application.

Using Groups

It is recommended, for ease of administration, that the user IDs used by the daemons be connected to a group and that group be given the appropriate permissions to the RACF profiles. Refer to the *z/OS Security Server RACF Security Administrator's Guide* for more information on group profiles.

Refreshing z/OS Security Server Data

After all z/OS SecureWay Security Server RACF definitions have been made, the FACILITY class must be refreshed if it is RACLISTED. Issue this command to perform this action:

```
SETROPTS RACLIST(FACILITY) REFRESH
```

If the FACILITY class is not active you may activate it with this command:

```
SETROPTS CLASSACT(FACILITY)
```

If members were added to PROGRAM Class profiles, program control for those members will not be in effect until this command is issued:

```
SETROPTS WHEN(PROGRAM) REFRESH
```

For more information, refer to the *z/OS Security Server RACF Security Administrator's Guide*.

Running the Installation Script

The installation script is run from a z/OS shell session. IBM recommends that the script be run from a user ID with a UID of 0 (super user). In addition:

- The user ID running the script must be given authorization to use OCSF services by being granted READ access to the CDS.* OCSF facility class profiles.
- If your system is operating with z/OS UNIX security in effect, the user ID running the script must be permitted to the BPX.SERVER facility class profile. (This requirement applies even if you are running the script from a UID 0 user ID.)

Perform these steps:

1. Go to the correct directory, for example:

   ```
   cd /usr/lpp/ocsf/bin
   ```

2. Run this script: **ocsf_install_crypto**

   You receive this output:

   ```
   Installing CSSM...
   CSSM Framework successfully installed
   Installing IBMTP...
   Addin successfully installed.
   Installing IBMTP2...
   Addin successfully installed.
   Installing IBMCL...
   Addin successfully installed.
   Installing IBMCL2...
   Addin successfully installed.
   Installing IBMDL2...
   Addin successfully installed.
   Installing LDAPDL.
   Addin successfully installed.
   Installing IBMWKCSP...
   Addin successfully installed.
   Installing IBMCCA...
   Addin successfully installed
   Installing IBMSWCSP
   Addin successfully installed
   ```

3. When this runs correctly, go to "Running the Installation Verification Procedure."

Running the Installation Verification Procedure

Once you have completed the previous steps, run the Install Verification Procedure (IVP). This verifies that you have installed and configured correctly. To correctly test your configuration, it is suggested that you run the IVP under a few different z/OS user identities that have been authorized to issue OCSF applications. This tests out the configuration done in "Setting Up the Necessary Security Authorizations" on page 1.

- Perform these steps:
  1. Go to the correct directory, for example:

     ```
     cd /us/Lapp/itself/ivp
     ```

2. Read the README.ivp and follow the instructions for running the Installation Verification Procedure.
3. Run this script: **ocsf_baseivp**

   You will receive this output:

```
Starting OCSF base addins ivp
Initializing CSSM
CSSM Initialized
Attaching ibmwkcsp
Attach successful, detaching ibmwkcsp
Detach of ibmwkcsp successful
Attaching ibmswcsp
Attach successful, detaching ibmswcsp
Detach of ibmswcsp successful
Attaching ibmcca
Attach successful, detaching ibmcca
Detach of ibmcca successful
Attaching ibmcl
Attach successful, detaching ibmcl
Detach of ibmcl successful
Attaching ibmcl2
Attach successful, detaching ibmcl2
Detach of ibmcl2 successful
Attaching ibmdl2
Attach successful, detaching ibmdl2
Detach of ibmdl2 successful
Attaching ldapdl
Attach successful, detaching ldapdl
Detach of ldapdl successful
Attaching ibmtp
Attach successful, detaching ibmtp
Detach of ibmtp successful
Attaching ibmtp2
Attach successful, detaching ibmtp2
Detach of ibmtp2 successful
Completed OCSF base addins ivp
```

4. When this runs correctly, your installation is complete.

   When you have the Security Level 3 Feature installed, you should perform the additional step of verifying that the correct policy table files are being used. The files **/usr/lpp/ocsf/lib/cssmmanp.dll** and **/usr/lpp/ocsf/lib/cssmusep.dll** are actually links. When only the OCSF base is installed, these links should point to **cssmmanp_sl2.dll** and **cssmusep_sl2.dll**. When the Security Level 3 Feature is installed, they should point to **cssmmanp_sl3.dll** and **cssmusep_sl3.dll**.

Common Problems

The most common problems that may occur now during installation, or in the future when running applications that use Open Cryptographic Services Facility, are unauthorized code or unauthorized users. These kinds of problems can result in return code 9 errors when running the OCSF installation script, or can result in the application returning an error condition that the user or code is unauthorized.

If you encounter these types of errors, here are some things to check:

- Verify that the user ID running the OCSF application or OCSF install script has been permitted to the CDS.* facility class profiles.
- If your system is operating with z/OS UNIX security in effect (either BPX.SERVER or BPX.DAEMON are defined) verify that the user ID running the OCSF application or OCSF install script has been permitted to the BPX.SERVER facility class profile.

  If your system is operating with UNIX security in effect (neither BPX.SERVER nor BPX.DAEMON are defined), verify that the user ID running the OCSF application or OCSF install script has an effective UID of 0.
- Verify that all of the programs, modules, and dynamically loaded libraries (DLLs) being used by the OCSF application have been defined as program

controlled. This includes the modules supplied by OCSF itself, the C/C++ run time library modules, the modules associated with the OCSF application and any other libraries used by the application.

You can use the UNIX shell **ls** command with the **-E** option to verify the program-controlled extended attribute of programs and DLLs resident in the UNIX file system. Refer to *z/OS UNIX System Services Planning* for procedures to follow for verifying that modules loaded from load libraries are defined as program controlled.

If you have problems in determining which program is not program controlled, go to the z/OS Operator's Console and look for message number BPXP015I. This should tell you the name of the program that needs to be program controlled.

- If the problems occur while running in the UNIX shell, they may be due to a dirty address space caused by utilities or applications that were run earlier in the shell session for which _BPX_SHAREAS is YES. Try setting the _BPX_SHAREAS environment variable to NO before running the OCSF installation script or any OCSF application. Setting this environment variable to NO forces new commands or processes to be run in a new address space rather than sharing the current (possibly dirty) one.
- If the problems occur during running of the OCSF installation script, verify that the user ID running the script has write access to the **/var/ocsf** directory.
- Verify that all dates are prior to the year 2038. OCSF does not support the concept of time past the year 2038.

Chapter 2. Open Cryptographic Services Facility Framework

The OCSF Framework layer is the central component in the OCSF architecture; it integrates and manages all the security services. OCSF enables tight integration of individual services, while allowing those services to be provided by interoperable service provider modules. The OCSF Framework has a rich application programming interface (API) to support the development of secure applications and system services, and a service provider interface (SPI) that supports service provider modules that implement building blocks for secure operations.

The primary function of the OCSF Framework layer is to maintain a state regarding the connections between the application layer code and the service providers underneath. Additionally, the OCSF mediates all interactions between applications and the service provider modules and implements and enforces the applicable cryptographic policy. Finally, the OCSF Framework allows the seamless integration of other security functions provided by independent service provider modules.

The OCSF Framework does not prescribe or implement any security services. Application-specific security services are defined and implemented by service provider modules and layered services. The OCSF Framework defines a common API for accessing the services provided by service provider modules. OCSF redirects application API calls to the selected service provider module that will perform the request.

The OCSF API calls can be categorized as service operations or core services. Service operations are functions that invoke a service provider module security operation, such as encrypting data, adding a certificate to a Certificate Revocation List (CRL), or verifying that a certificate is trusted/authorized to perform some action. OCSF module managers are responsible for carrying out service operations. Core services include functions that perform:

- Module management
- Memory management
- Security context management
- Integrity verification.

Chapter 2, “Open Cryptographic Services Facility Framework” discusses the OCSF Framework core services. The individual OCSF module managers are discussed in Chapter 3, “OCSF Policy Modules,” on page 17 through Chapter 7, “Data Storage Library Module Manager,” on page 27. See Chapter 8, “Service Provider Modules,” on page 29, for information on the IBM service provider modules and the functions supported by the individual service providers.

Module Management

The OCSF Framework defines a set of API calls that allow application developers to access and use service provider modules. These module management functions support the installation of service provider modules, the dynamic selection and loading of modules, and the querying of module features and status. System administration utilities use install and uninstall functions to maintain service provider modules on a local system.

Installing and Uninstalling Service Provider Modules

OCSF manages a registry that records the logical name of each service provider module that is installed on the system, the information required to locate and dynamically initiate the service provider, and some minimal meta-data describing the algorithms implemented by the service provider.

A service provider must be installed to the OCSF by recording its services with the OCSF Framework using CSSM_ModuleInstall before an application or another service provider module can use its services.

When a service provider is loaded at run-time it registers a set of OCSF callback functions with the OCSF Framework. There is one callback function for each OCSF-defined SPI call. The service provider may or may not implement all SPI calls defined by OCSF. Unimplemented functions must be registered as null. The service provider may implement additional functions outside of the OCSF-defined SPI calls. The service provider may register a single callback function, and instruct application and module developers (through documentation) to activate these functions through the message-based, OCSF passthrough function. There is one passthrough function defined in each SPI. For example, the passthrough function defined for the cryptographic SPI is CSP_PassThrough.

Service provider modules may also be uninstalled from the OCSF by using the CSSM_ModuleUninstall function. This function removes the service provider name and its associated attributes from the OCSF Framework's service provider registry. Uninstalled must be performed before a new version of the same service provider module is installed in the OCSF Framework registry. It is the responsibility of the service provider to provide the install and uninstall functions.

Listing Service Provider Modules and Services

Before attaching a service module, an application can query the OCSF Framework registry using the CSSM_ListModules function to obtain information on the:

- Modules installed on the system
- Capabilities (and functions) implemented by those modules
- Globally Unique ID (GUID) associated with a given module.

Applications use this information to dynamically select a module for use. A multiservice module has multiple capability descriptions associated with it, at least one per functional area supported by the module. Some areas (such as Cryptographic Service Provider (CSP) and Trust Policy (TP)) may have multiple independent capability descriptions for a single functional area. There is one OCSF Framework registry entry for a multiservice module, which records all service types for the module. OCSF returns all information about a module's capabilities when queried by the application. Each set of capabilities includes a type identifier to distinguish CSPinfo from CLinfo, etc.

Applications can query about the OCSF Framework itself. One function, CSSM_GetInfo, returns version information about the running OCSF Framework. Another function, CSSM_Init, verifies whether the OCSF Framework version the application expects is compatible with the currently running OCSF Framework version. The general function to query service provider module information also returns the module's version information.

Attaching and Detaching Service Provider Modules

Applications select the particular security services they will use by selectively attaching service provider modules. Each module has an assigned GUID and a set of descriptive attributes to assist applications in selecting appropriate modules for their use. A module can implement a range of services across the OCSF APIs (e.g., cryptographic functions, data storage functions) or a module can restrict its services to a single OCSF category of service (e.g., Certificate Library (CL) services only). Modules that span service categories are called multiservice modules.

Applications use a module's GUID to specify the module to be attached. The attach function, CSSM_ModuleAttach, returns a handle representing a unique pairing between the caller and the attached module. This handle must be provided as an input parameter when requesting services from the attached module. OCSF uses

the handle to match the caller with the appropriate service module. The calling application uses the handle to obtain all types of services implemented by the attached module. Figure 2 shows how the handle for an attached Dual Provider service provider is used to perform cryptographic operations and persistent storage of certificates. The single handle value can be used as the CSPHandle in cryptographic operations and as the DLHandle in data storage operations.

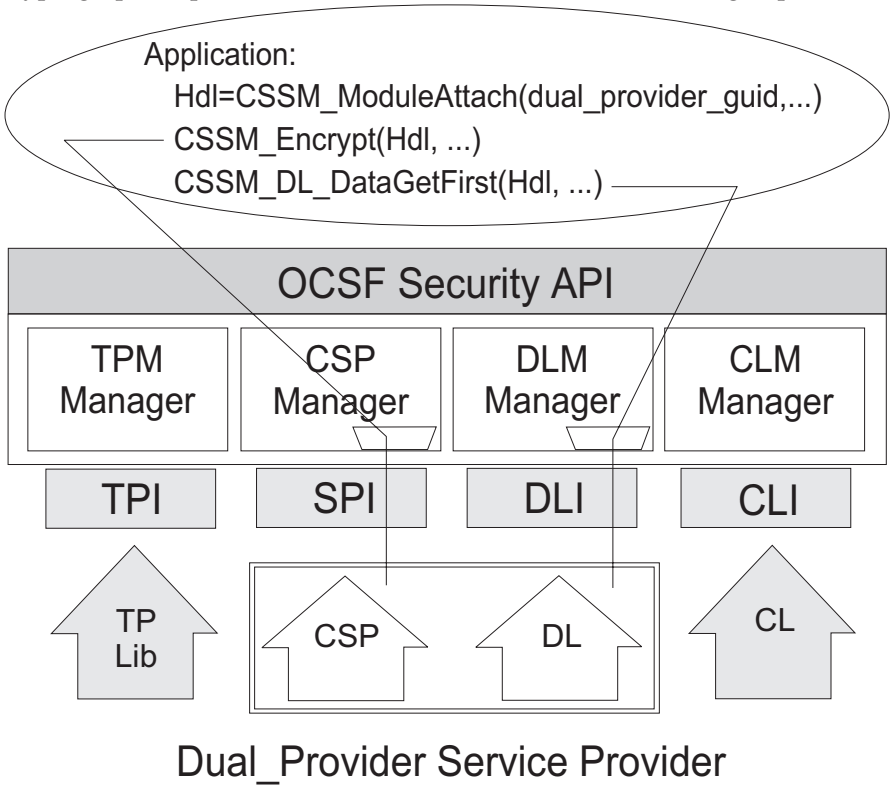

Figure 2. Dual_Provider Cryptographic Services and Persistent Storage Services

Multiple calls to attach are viewed as independent requests. Each attach request returns separate, independent handles that do not share execution state. Service provider modules may be detached using the CSSM_ModuleDetach function. However, an application should not invoke this operation unless all requests to the target service provider have been completed.

Managing Calls Between Service Provider Modules

Applications directly or indirectly select the modules that will be used to provide security services to the application. Service provider modules may (and often will) invoke other service provider modules to perform necessary operations. OCSF forwards all calls uniformly regardless of their origin. Figure 3 illustrates the process by which the OCSF Framework manages calls between modules.

In Figure 3 on page 13, the application invokes func1 in the cryptographic module identified by the handle CSP1. OCSF forwards the function call to func1 in the CSP1 module. The application also invokes func7 in the TP module identified by the handle TP2. Again, OCSF forwards the function call to func7 in the TP2 module. The implementation of func7 in the TP2 module uses functions implemented by a CL module. The TP2 module must invoke the CL functions through the OCSF Framework. To accomplish this, the TP2 module attaches the CL

module, obtaining the handle CL1, and invokes func13 in the CL identified by the handle CL1. OCSF forwards the function call to func13 in the CL1 module. Modules must be attached before they can receive function calls from the OCSF Framework. An error condition occurs if the selected module does not implement the invoked function.

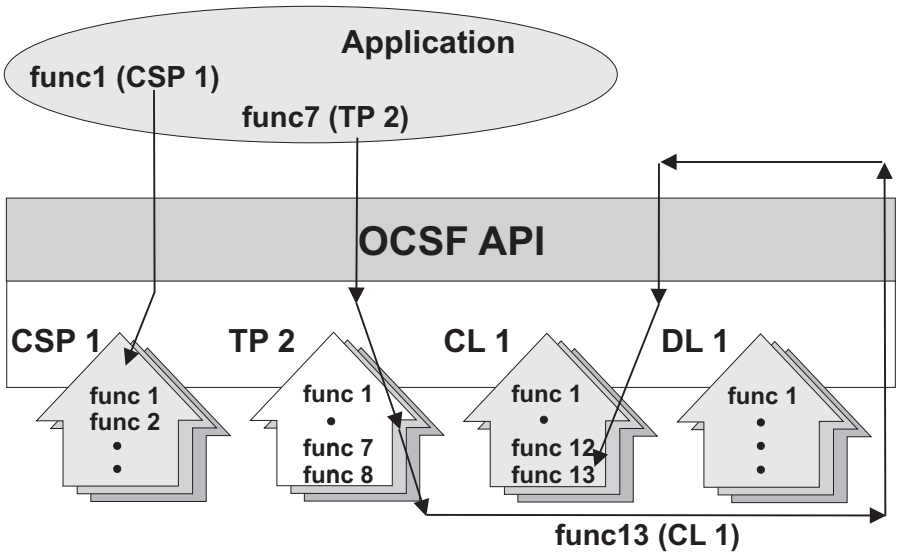

Figure 3. OCSF Framework Directs Calls to Selected Service Provider Modules

Memory Management

The OCSF memory management functions are a class of routines for reclaiming memory allocated by OCSF on behalf of an application from the OCSF memory heap. When OCSF allocates objects from its own heap and returns them to an application, the application must inform OCSF when it no longer requires the use of that object. Applications use specific APIs to free OCSF-allocated memory. When an application invokes an API free function, OCSF can choose to retain or free the indicated object depending on other conditions known only to OCSF. In this way, OCSF and applications work together to manage these objects in the OCSF memory heap.

Security Context Management

Security context management provides secured run-time caching of user-specific state information and secrets. Multistep cryptographic operations, such as staged hashing, require multiple calls to a CSP and the intermediate operation states must be managed. These intermediate states are stored in run-time data structures known as security contexts. The OCSF API provides a number of context functions that applications can use to create, initialize, and cache security contexts.

Security contexts provide mechanisms that:

- Allow an application to use multiple CSPs concurrently.
- Allow an application to concurrently use different parameters for a single CSP algorithm.
- Support layered implementations in their transparent use of multiple CSPs or different algorithm parameters for the same CSP.
- Enable development of reentrant CSPs, layered services, and applications.

Applications retain handles to each security context used during execution. The context handle is a required input parameter to many security service functions. Most applications instantiate and use multiple security contexts. Only one context may be passed to a function, but the application is free to switch among contexts at will, or as required (even per function call).

An application may create multiple contexts directly or indirectly. Indirect creation may occur when invoking layered services, system utilities, TP modules, CL modules, or DL modules that create and use their own appropriate security context as part of the service they provide to the invoking application. Figure 4 shows an example of a hidden security context. An application creates a context specifying the use of sec_context1. The application invokes func1 in the CL using sec_context1 as a parameter. The CL performs two calls to the CSP. For the call to func5, the hidden security context is used. For the call to func6, the application's security context, sec_context2, is passed as a parameter to the CSP.

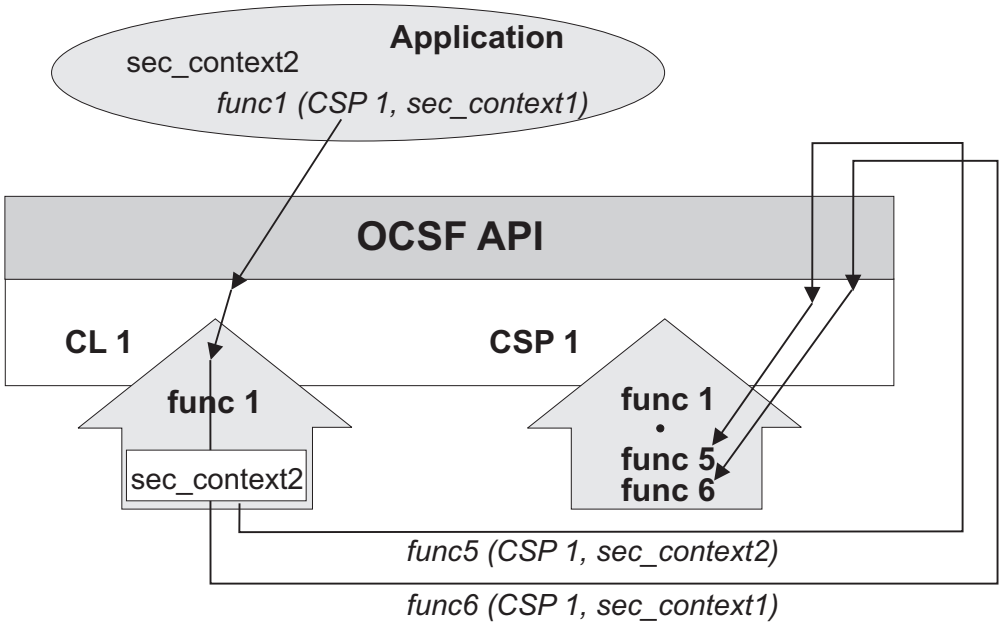

Figure 4. Indirect Creation of a Security Context

These transparent contexts do not concern the application developer, as they are managed entirely by the layered service or service provider module that created them. Each process or thread that creates a security context is responsible for explicitly terminating that context.

OCSF provides a number of API functions to create security contexts. The function used and type of context created depends on the cryptographic operation being performed. For example, the CSSM_CSP_CreateSymmetricContext is used in cryptographic operations involving a symmetric key; the CSSM_CSP_CreateAsymmetricContext is used in operations involving an asymmetric key.

The CSSM_DeleteContext function is paired up with the create context functions. These functions are designed to be used by applications and force notify events to be sent to a service provider module. In contrast, the CSSM_GetContext and CSSM_FreeContext functions are designed to be used by service provider modules since they do not generate events.

OCSF Security Context Changes

In OS/390® OCSF Version 2 Release 9 (V2R9), a change was made in the way OCSF security contexts are manipulated. Before V2R9, a CSSM_Get... Context call caused a new copy of the OCSF security context, created by a CSSM_CSP_Create...Context call, to be created. That copy had to be freed by a

CSSM_FreeContext call. In V2R9, however, a copy of the security context is created during the CSSM_CSP_Create...Context call. When any CSSM_GetContext call is made, a pointer to the copy is returned. Although CSSM_FreeContext should still be issued for compatibility, the security context copy is freed when CSSM_DeleteContext is called. Developers must be careful that in their applications no CSSM_UpdateContextAttribute calls are made while any thread is using a context, either explicitly or implicitly during a CSSM_CSP... call. Also, applications must complete all uses of the CSSM security context before the CSSM_DeleteContext call is made.

Integrity Verification Services

As a security frame work, OCSF provides each application with checking of the integrity of the OCSF environment in which the application is running. OCSF requires all code including OCSF binaries and the invoking application to be program controlled. Non-program controlled binaries causes the environment to become "dirty" and the result will be failure of attaching OCSF Service Providers. In addition, Cryptographic Service Providers and Policy Modules have additional checks to verify their validity.

Chapter 3. OCSF Policy Modules

Policy modules are provided with OCSF that represent the cryptographic algorithms and their associated strengths that can be used when performing data encryption or decryption. Mandated policies are typically derived from jurisdiction-based regulations on the use of cryptographic products for data confidentiality. The jurisdictions for policies can coincide with the political boundaries of countries in order to enforce the law enforcement needs of these political jurisdictions. Political jurisdictions may define policies based on export, import or use controls. Policies specify the exact cryptographic protocol suites (algorithms, modes, key lengths, etc.) allowed. Chapter 3, "OCSF Policy Modules" describes how policies are used in the OCSF.

Usage of OCSF Policy Modules

There are two policy modules within the OCSF. These modules dictate the policies enforced by the OCSF framework when an application requests symmetric or asymmetric encryption or decryption. One of the policy modules represents the cryptographic algorithms and their strengths allowed by the US government. This is sometimes referred to as the policy module for the country of manufacture. The second policy module represents the cryptographic algorithms and their strengths allowed where OCSF is being used. This is sometimes referred to as the policy module for the country of use. The values in these modules are dependent upon whether only the OCSF base or the OCSF Security Level 3 feature is applied.

OCSF Behavior When Only the OCSF Base is Installed

The use and behavior of policy modules by the OCSF framework when only the OCSF base is installed are as follows:

- For symmetric encryption, a check is made to disallow nested encryptions of a data buffer. If the input buffer to be encrypted is identical to a buffer of cipher text produced in the recent past, the framework considers this an attempt to perform nested encryption of a data buffer and disallows it.
- When a symmetric context is created or updated a check is made to see if the strength of the cryptography requested is stronger than allowed by the policy modules or if the algorithm requested is not defined by the policy modules. If so, the cryptographic context is flagged. An encryption or decryption request made with that context will be denied.
- When an asymmetric context is created or updated a check is made to see if the strength of the cryptography requested is stronger than allowed by the policy modules or if the algorithm requested is not defined by the policy modules. If so, the cryptographic context is flagged. An encryption, decryption, key wrap or key unwrap request made with that context will be denied.

OCSF Behavior When the OCSF Security Level 3 Feature is Installed

When the OCSF Security Level 3 feature is installed there are no restrictions on cryptographic strengths or algorithms used. Policy checking and enforcement is waived.

Implementation of OCSF Policy Modules

The OCSF Policy modules are implemented in two separate DLLs:
- cssmmanp.dll - corresponding to the policies for country of manufacture
- cssmusep.dll - corresponding to the policies for the country of use.

These DLLs are loaded automatically when the application issues the required CSSM_Init API that is used to instantiate the Framework.

Chapter 4. Cryptographic Module Manager

The Cryptographic Module Manager administers the Cryptographic Service Providers (CSPs) modules that may be installed on the local system, and defines a common application programming interface (API) for accessing CSP modules. All cryptography functions are implemented by the CSPs. This localizes all cryptography into exchangeable modules. OCSF administers a queryable registry of local CSPs. The registry lists the locally accessible CSPs and their cryptographic services (and algorithms).

The nature of the cryptographic functions contained in any particular CSP depends on the task the CSP was designed to perform. For example, a VISA smart card would be able to digitally sign credit card transactions on behalf of the card's owner. A digital employee badge would be able to authenticate a user for physical or electronic access.

The Cryptographic Module Manager does not assume any particular form for a CSP. CSPs can be implemented in hardware, software, or both; operationally, the distinction must be transparent. The two visible distinctions between hardware and software implementations are the degree of trust the application receives by using a given CSP, and the cost of developing that CSP. A hardware implementation should be more tamper-resistant than a software implementation. Hence a higher level of trust is achieved by the application. All CSPs that can be loaded by the OCSF must contain a verification check[2].

Multiple CSPs may be loaded and active within the OCSF at any time, and a single application may use multiple CSPs concurrently. Interpreting the resulting level of trust and security is the responsibility of the application or the TP module used by the application. The Cryptographic Module Manager defines a high-level, certificate-based API for cryptographic services to support application development. This API is in Chapter 12, “Cryptographic Services API,” on page 107. A CSP may or may not support multithreaded applications. For information on interface support by cryptographic service providers, refer to the *z/OS Open Cryptographic Services Facility Service Provider Module Developer's Guide and Reference*. For specifics on the cryptographic service proviers available with OCSF, refer to Chapter 8, “Service Provider Modules,” on page 29.

Supporting Legacy CSPs

CSPs existed prior to the definition of the OCSF Cryptographic API. These legacy CSPs have defined their own APIs for cryptographic services. These interfaces are CSP-specific, nonstandard, and (in general) low-level key-based interfaces. They present a considerable development effort to the application developer attempting to secure an application by using those services.

Acknowledging legacy CSPs, the OCSF defines an optional adaptation layer between the Cryptographic Module Manager and a CSP. The adaptation layer allows the CSP vendor to implement a shim to map the OCSF SPI to the CSP's existing API, and to implement any additional management functions that are

2. If you want to provide a Cryptographic Service Provider, you need to contact IBM. For more information, see the *z/OS Open Cryptographic Services Facility Service Provider Module Developer's Guide and Reference*.

required for the CSP to function as a service provider module in the extensible OCSF. New CSPs may support the OCSF SPI directly (without the aid of an adaptation layer).

Cryptography Services API

The security services API defined by the Cryptographic Module Manager are certificate-based. This contrasts with the approach taken by many CSPs, where low-level concepts such as key type, key size, hash functions, and byte ordering are the standard granularity of interface options. The Cryptographic Module Manager hides these behind high-level operations such as:

- SignData
- VerifyData
- DigestData
- EncryptData
- DecryptData
- GenerateKeyPair.

Security-conscious applications use these high-level concepts to provide authentication, data integrity, data and communication privacy, and non-repudiation of messages to the end-users. A CSP may implement any algorithm. For example, CSPs may provide one or more of the algorithms, in one or more modes:

- Bulk encryption algorithm: DES, Triple DES, IDEA, RC2, RC4, RC5, Blowfish, CAST
- Digital signature algorithm: RSA, DSS
- Key negotiation algorithm: Diffie-Hellman
- Cryptographic hash algorithm: MD4, MD5, SHA
- Unique identification number: hardcoded or random generated
- Random number generator: attended and unattended
- Encrypted storage: symmetric-keys, private-keys.

The application's associated security context defines parameter values for the low-level variables that control the details of cryptographic operations. Setting input parameters to cryptographic algorithms is not a policy decision of the OCSF Framework. Applications use CSPs that provide the services and features required by the application. For example, an application issuing a request to *EncryptData* may reference a security context that defines these parameters:

- Algorithm to be used (such as RC5)
- Algorithm-specific parameters (such as key length)
- Cryptographic variables (such as the key).

Most applications will use default OCSF contexts that are available through API function calls such as CSSM_CSP_CreateSignatureContext. Typically, a distinct context will be used for encrypting, hashing, and signing. For a given application, once initialized, these contexts will 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.

Application developers who demand fine-grained control of cryptographic operations can achieve this by directly and repeatedly updating the security context to direct the CSP for each operation, and by using the Cryptographic Module Manager API *passthrough* feature.

Dependencies with the Policy Modules

The Cryptographic Module Manager of the OCSF is responsible for handling the cryptographic functions of OCSF and the enforcement of the cryptographic algorithms and strengths allowed by the policy module. The Cryptographic Module Manager and cryptographic functions in the OCSF framework:

- Invoke policy enforcement functions for cryptographic context create and update operations.
- Set the cryptographic context unusable if the cryptographic strength is too strong or an algorithm requested is not allowed as per the policy modules.
- Check the cryptographic context before allowing encryption/decryption operations to occur.

Whenever a cryptographic context is created or updated using the OCSF API functions, the Cryptographic Module Manager invokes a policy enforcement function; the latter checks the policies to determine whether the cryptographic context defines an operation or strength outside of the allowable bounds as defined by the policy modules. If so, the cryptographic context is set to signal that the context is unusable. If the cryptographic context is updated so that the request is included in the bounds of the policy module, then the context is set to be usable again.

When the encryption/decryption operations of the OCSF are invoked, the Cryptographic Module Manager checks the cryptographic context to determine whether the context is usable for encryption/decryption operations. If the context is flagged as unusable, the encryption/decryption API function returns an error and the encryption/decryption operation will not take place.

Chapter 5. Trust Policy Module Manager

The Trust Policy (TP) Module Manager administers the TP modules that may be installed on the local system and defines a common application programming interface (API) for these libraries. The TP API allows applications to request security services that require *policy review and approval* as the first step in performing the operation. Operations defined in the TP API include verifying trust in:

- A certificate for signing or revoking another certificate
- A user or user-agent to perform an application-specific action
- The issuer of a Certificate Revocation List (CRL).

A digital certificate binds an identification in a particular domain to a public key. When a certificate is issued (created and signed) by a Certificate Authority (CA), the binding between key and identity is attested by the digital signature on the certificate. The issuing authority also associates a level of trust with the certificate. The actions of the user, whose identity is bound to the certificate, are constrained by the TP governing the certificate's usage domain. A digital certificate is intended to be an unforgettable credential in cyberspace.

The use of digital certificates is the basis on which the OCSF is designed. The OCSF assumes the concept of digital certificates in its broadest sense; that is, an identity bound to a public key. Certificates are often used for identification, authentication, and authorization. The way in which applications interpret and manipulate the contents of certificates to achieve these ends is defined by the real world trust model the application has chosen as its model for trust and security.

The primary purpose of a TP service provider is to answer the question *"Is this certificate trusted for this action?"* The OCSF TP API defines the generic operations that should be defined for certificate-based trust in every application domain. The specific semantics of each operation is defined by the:

- Application domain
- Trust model
- Policy statement for a domain
- Certificate type.

The trust model is expressed as an executable policy that is used/invoked by all applications that ascribe to that policy and the trust model it represents.

As an infrastructure, OCSF is policy neutral; it does not incorporate any single policy. For example, the verification procedure for a credit card certificate should be defined and implemented by the credit company issuing the certificate. Employee access to a lab housing a critical project should be defined by the company whose intellectual property is at risk. Rather than defining policies, OCSF provides the infrastructure for installing and managing policy-specific modules. This ensures extensibility of certificate-based trust on every platform hosting OCSF.

Different TPs define different actions that may be requested by an application. There are also a few basic actions that should be common to every TP. These actions are operations on the basic objects used by all trust models. The basic objects common to all trust models are certificates and CRLs. The basic operations on these objects are sign, verify, and revoke.

Application developers and trust domain authorities benefit from the ability to define and implement policy-based modules. Application developers are freed from the burden of implementing a policy description and certifying that their implementation conforms. Instead, the application only needs to build in a list of the authorities and certificate issuers it uses.

Domain authorities also benefit from an infrastructure that supports TP modules. Authorities are sure that applications using their modules will adhere to the policies of the domain. In addition, dynamic download of trust modules (possibly from remote systems) ensures timely and accurate propagation of policy changes. Individual functions within the module may combine local and remote processing. This flexibility allows the module developer to implement policies based on the ability to communicate with a remote authority system. This also allows the policy implementation to be decomposed in any convenient distributed manner.

Implementing a TP module may or may not be tightly coupled with one or more CL modules and one or more DL modules. The TP embodies the semantics of the domain. The CL and the DL embody the syntax of a certificate format and operations on that format. A TP can be completely independent of certificate format, or it may be defined to operate with a small number of certificate formats. A TP implementation may invoke a CL module and/or a DL module to manipulate certificates.

Trust Policy API

OCSF provides TP operations on certificates and CRL lists. These operations include:

- TP operations, such as signing, verifying, or revoking, on individual certificates and CRLs.
- TP operations on groups of certificates such as constructing an ordered group, verifying the signatures on a group, and removing certificates from a group.
- Passthrough operations for unique certificate and CRL operations.
- For detailed information on each of these functions, see Chapter 14, “Trust Policy Services API,” on page 191.

Chapter 6. Certificate Library Module Manager

The Certificate Library Module Manager administers the Certificate Libraries (CLs) that may be installed on the local system. It defines a common application programming interface (API) for these libraries.

The API allows applications to manipulate memory-resident certificates and Certificate Revocation Lists (CRLs).

Operations defined in the API include create, sign, verify, and extract field values. The CL modules implement all certificate operations. Application-invoked calls are dispatched to the appropriate library module. Each library incorporates knowledge of certificate data formats and how to manipulate that format. The OCSF Certificate Module Manager administers a queryable registry of local libraries. The registry enumerates the locally accessible libraries and attributes of those libraries, such as the certificate type manipulated by each registered library.

The primary purpose of a CL module is to perform memory-based, syntactic manipulations on the basic objects of trust: certificates and CRLs. The data format of a certificate will influence (if not determine) the data format of CRLs used to track revoked certificates. For this reason, these objects should be manipulated by a single, cohesive library. CL modules incorporate detailed knowledge of data formats. The Certificate Library Module Manager defines API calls to perform security operations (such as signing, verifying, revoking, viewing, etc.) on memory-resident certificates and CRLs. The mechanics of performing these operations is tightly bound to the data format of a given certificate. One or more modules may support the same certificate format, such as X.509 ASN/DER-encoded certificates or Simple Distributed Security Infrastructure (SDSI) certificates.

As new standard formats are defined and accepted by the industry, CL modules will be defined and implemented by industry members and used directly and indirectly by many applications. CL modules encapsulate certificate and CRL data formats from the semantics of TPs, which are implemented in TP modules.

Since CL modules manipulate memory-based objects only, the persistence of certificates and CRLs is an independent property of these objects. It is the responsibility of the application and/or the TP module to use data storage modules to make these objects persistent (if appropriate). It must be possible for the storage mechanism used by a data storage module to be independent of the other modules. It must also be possible to design a CL module that depends on the storage mechanism of a DL module.

Application developers and TP module developers both benefit from the extensibility of CL modules. Applications are free to use multiple certificate types without requiring the application developer to write format-specific code to manipulate certificates and CRLs. Without increased development complexity, multiple certificate formats can be used on one system, within one application domain, or by one application. Certificate Authorities (CAs) who issue certificates also benefit. Dynamically downloading CLs ensures timely and accurate propagation of data-format changes.

Certificate Library Services API

The Certificate Library Services API defines numerous operations on memory-resident certificates and CRLs as required by every certificate type. These operations include:

- Creating new certificates and new CRLs
- Signing existing certificates and existing CRLs
- Viewing certificates
- Verifying certificates and CRLs
- Extracting values (e.g., public keys) from certificates
- Importing and exporting certificates of other data formats
- Revoking certificates
- Reinstating revoked certificates
- Searching CRLs
- Providing passthrough for unique, format-specific certificate and CRL operations.

For detailed information on the Certificate Library API functions, see Chapter 15, "Certificate Library Services API," on page 207.

Chapter 7. Data Storage Library Module Manager

The Data Storage Library Module Manager defines an application programming interface (API) for secure, persistent storage of certificates and Certificate Revocation Lists (CRLs). The API allows applications to search and select certificates and CRLs, and to query meta-data about each data store (such as its name, date of last modification, size of the data store, etc.). Data Storage Library (DL) modules implement data store operations. These modules may be drivers or gateways to traditional, full-featured Database Management Systems (DBMS), to customized services layered over a file system, or provide access to other forms of stable storage. A data storage module may execute and store its data locally or remotely.

The primary purpose of a DL module is to provide secure, persistent storage, retrieval, and recovery of certificates and CRLs. The persistence of these generic trust objects is independent of the memory-based manipulations performed by Certificate Library (CL) modules. DL modules may be invoked by applications, TP modules, or CL modules that make decisions about the persistence of these trust objects.

A single DL module may be tightly tied to a CL module or may be independent of all CL modules. A data DL that is tightly tied to a CL module implements a persistent storage mechanism that is dependent on the data format of the certificate. An independent DL implements a storage mechanism that stores certificates and CRLs without regard for their specific format. A single, physical data store managed by such DL modules may even contain individual certificates of different formats.

Each DL module can manage any number of independent, physical data stores. Each data store must have a logical name used by callers to refer to the persistent data store. Implementation of the DL module may use local file system facilities, commercial database management products, and custom-stable storage devices.

A DL module is responsible for the integrity of the records it stores. If the DL module uses an underlying commercial DBMS, it may choose to further secure the data store by leveraging integrity services provided by the DBMS. DL modules that choose to implement persistence using the local file system or a custom- stable storage device, must decide which (if any) integrity mechanisms to provide.

Data Storage Library Services API

The Data Storage Library Services API defines two categories of operations, which include:

- Data store management functions. The data store management functions operate on a data store as a single unit. These operations include opening and closing data stores, creating and deleting data stores, and importing and exporting data stores. A data store may contain certificates only, CRLs only, or both. It is unusual for a DL module to manage a data store containing both certificates and CRLs, but there is nothing in the OCSF or the DL module API that prevents a DL module from implementing persistence in this manner. Typically, separate physical data stores are used to store certificates and CRLs.

- Persistence operations on certificates and CRLs. The persistence operations on data stores include:
  - Adding new certificates and new CRLs
  - Updating existing certificates
  - Deleting certificates and CRLs
  - Retrieving certificates and CRLs
  - Passthrough for unique, module-specific operations.

For detailed information on the Data Storage Library API functions, see Chapter 16, "Data Storage Library Services API," on page 231.

Chapter 8. Service Provider Modules

All cryptographic and key recovery functions, as well as the Trust Policies (TPs), certificates, and data store functions are performed by service provider modules. The OCSF Framework itself only manages the interactions between service provider modules and applications that use them. The OCSF Architecture supports these types of service providers.

- Cryptographic Service Providers
- Trust Policy Modules
- Certificate Library Modules
- Data Storage Library Modules.

Chapter 8, "Service Provider Modules" presents a brief overview of each type of service provider module. For a detailed discussion of the OCSF interface the service providers must support refer to the *z/OS Open Cryptographic Services Facility Service Provider Module Developer's Guide and Reference*. Independent Software Vendors (ISVs) who develop modules for use with OCSF must support the interface specifications described. The modules may implement all or a subset of these application programming interfaces (APIs). A single module may also provide services in multiple categories of service. These are called multiservice modules. Several service provider modules are provided with the OCSF. These modules are described in "OCSF Service Provider Modules" on page 31.

Cryptographic Service Provider Modules

Cryptographic Service Providers (CSPs) are modules equipped to perform cryptographic operations and to securely store private keys. A CSP may implement one or more of these cryptographic functions:

- Bulk encryption algorithm
- Digital signature algorithm
- Cryptographic hash algorithm
- Unique identification number
- Random number generator
- Secure key storage
- Custom facilities unique to the CSP.

A CSP may be implemented in software, hardware, or both. Typically, CSPs provide encrypted storage for private keys and variables. CSPs must also deliver key management services, including key escrow, if it is supported. As a minimum, CSPs do not reveal key material unless it has been wrapped, but they must support importing, exporting, and generating keys. The key generation module of a CSP should be made tamper-resistant.

CSPs typically provide secured storage of private keys and variables. Applications may query the CSP to retrieve private keys stored within the CSP. The CSP is responsible for controlling access to the private keys it secures. A callback function implemented by the requester is invoked by the CSP (or the CSP's adaptation layer) to obtain the identity and authorization of the user or process requesting the private key. Most CSPs are capable of importing private keys created by other CSPs and providing secured storage for such keys.

Trust Policy Modules

Trust Policy (TP) modules implement policies defined by Certificate Authorities (CAs) and institutions. Policies define the level of trust required before certain actions can be performed. Three basic categories or actions exist for all certificate-based trust domains:

- Actions on certificates
- Actions on Certificate Revocation Lists (CRLs)
- Domain-specific actions (such as issuing a check or writing to a file).

The generic operations defined in the *z/OS Open Cryptographic Services Facility Service Provider Module Developer's Guide and Reference* should be supported by every TP module. Each module may choose to implement the subset of these operations that are required for its policy. When a TP function has determined the trustworthiness of performing an action, the TP function may invoke functions in the Certificate Library (CL) and Data Storage Library (DL) modules to carry out the mechanics of the approved action.

Certificate Library Modules

Certificate Library (CL) modules implement syntactic manipulation of memory-resident certificates and CRLs. The OCSF Certificate API defines the generic operations that should be supported by every CL module. Each module may choose to implement only those operations required to manipulate a specific certificate data format.

The implementation of the CL operations should be free of certificate semantics. Semantic interpretation of certificate values should be implemented in TP modules, layered services, and applications. The OCSF makes manipulation of certificates and CRLs orthogonal to persistence of those objects. Hence, it is not recommended that CL modules invoke the services of DL modules. TP modules, layered security services, and applications should make decisions regarding the persistence of certificates.

Data Storage Library Module

A Data Storage Library (DL) module provides stable storage for certificates and CRLs. Stable storage could be provided by the:

- Commercially available Database Management System (DBMS) product
- Native file system
- Custom hardware-based storage devices.

Each DL module may choose to implement only those operations required to provide persistent storage for certificates and CRLs under its selected model of service.

Semantic interpretation of certificate values and CRL values is usually assumed to be implemented in TP modules. A pass-through function, DL_PassThrough, is defined in the DL API that allows each DL service provider to provide additional functions to store and retrieve certificates and CRLs, such as performance enhancing retrieval functions.

OCSF Service Provider Modules

A number of service provider modules may be provided with the OCSF. These modules can be incorporated into applications to perform cryptographic security operations. The modules include:

- Cryptographic Service Provider Module - There are five cryptographic modules that may be provided with OCSF.
  - IBM Software Cryptographic Service Provider, Version 1.0

    Note: This provider differs in the maximum key strength allowed for various symmetric and asymmetric encryption algorithms.
  - IBM Weak Software Cryptographic Service Provider, Version 1.0

    Note: This provider differs in the maximum key strength allowed for various symmetric and asymmetric encryption algorithms.
  - IBM Software Cryptographic Service Provider 2, Version 1.0

    Note: This provider may be used in place of or in addition to IBM Cryptographic Service Provider, Version 1.0.
  - IBM Weak Software Cryptographic Service Provider 2, Version 1.0

    Note: This provider may be used in place of or in addition to IBM WEAK Software Cryptographic Service Provider, Version 1.0.
  - IBM CCA Cryptographic Module, Version 1.0.
- Trust Policy Module - There are two trust policy modules that may be provided with OCSF.
  - IBM Standard Trust Policy Library, Version 1.0
  - IBM Extended Trust Policy Library, Version 1.0.
- Certificate Library Module - There is one supported certificate library module that is provided with OCSF.
  - IBM Certificate Library, Version 1.0

    Note: There is an additional certificate library module that is internal, it is the IBM Internal Certificate Library, Version 1.0
- Data Store Library Module - There is two data store library modules that may be provided with OCSF.
  - IBM Data Library, Version 1.0
  - IBM LDAP Data Library, Version 1.0

The OCSF API functions supported by each service of the provider modules are outlined in the Data Store Library Module. For detailed information on the behavior of the individual APIs, refer to the *z/OS Open Cryptographic Services Facility Service Provider Module Developer's Guide and Reference*.

IBM Software Cryptographic Service Provider, Version 1.0

Regarding Use of the IBM Software Cryptographic Service Provider

Portions of the IBM Software Cryptographic Service Provider contained in the Open Cryptographic Services Facility base of OS/390 contain software code provided by RSA Data Security, Inc.

Prior to utilizing the OS/390 Open Cryptographic Services APIs of the IBM Software Cryptographic Service Provider functionality contained in the OCSF base for purposes of development and test only, you must provide your company name, company contact name, address and telephone number to RSA Data Security, Inc. (RSA), by sending this information to:

Email: sales@rsa.com <mailto:sales@rsa.com> or

RSA Data Security, Inc.
2955 Campus Drive, Suite 400
San Mateo, CA 94403-2507
Attention: SALES or

FAX: 650-295-7770
Attention: SALES.

Prior to using (except for test or development purposes), marketing, selling, or distributing applications developed by you that directly utilize the Open Cryptographic Services Facility Cryptographic Services APIs of the IBM Software Cryptographic Service Provider functionality (i.e., utilizing the Open Cryptographic Services Facility Cryptographic Services APIs of the IBM Software Cryptographic Service Provider contained in the OCSF base), you must first obtain (if you have not already done so) a license from RSA for that application.

The files required for the IBM Software Cryptographic Service Provider, Version 1.0 are:
- ibmswcsp.so
- ibmswcsp.h

The IBM Software Cryptographic Service Provider module provides cryptographic functionality. Table 1 on page 33 lists the OCSF API functions supported by this module.

All functions that require input/output buffers support only one buffer at a time and not a vector of buffers. If an application provides a buffer to the CSP module, it must also specify the buffer length. On return from an OCSF API function, the length field of an output buffer will be set to the length of returned data. If an output buffer's length is set to zero and its data pointer is set to NULL, the CSP will allocate the needed memory on the application behalf. It is the responsibility of the application to free this memory when done.

For encryption/decryption operations, there are two operative contexts:
- Symmetric or asymmetric
- Key generation.

The effective bits attribute for RC2, or the rounds attribute for RC5, must be set in the symmetric context, not the key generation context. The value of either parameter is passed as the Params input to CSSM_CSP_CreateSymmetricContext or to CSSM_UpdateContextAttributes.

Table 1. IBM Software Cryptographic Service Provider OCSF Functions

Function	Suppported	Comments
CSSM_QuerySize	No	
CSSM_SignData CSSM_SignDataInit CSSM_SignDataUpdate CSSM_SignDataFinal	Yes	Algorithms Supported: CSSM_ALGID_MD2WithRSA CSSM_ALGID_MD5WithRSA CSSM_ALGID_SHA1WithRSA CSSM_ALGID_SHA1WithDSA
CSSM_VerifyData CSSM_VerifyDataInit CSSM_VerifyDataUpdate CSSM_VerifyDataFinal	Yes	Algorithms Supported: CSSM_ALGID_MD2WithRSA CSSM_ALGID_MD5WithRSA CSSM_ALGID_SHA1WithRSA CSSM_ALGID_SHA1WithDSA
CSSM_DigestData CSSM_DigestDataInit CSSM_DigestDataUpdate CSSM_DigestDataFinal	Yes	Algorithms Supported: CSSM_ALGID_MD2 CSSM_ALGID_MD5 CSSM_ALGID_SHA1
CSSM_DigestDataClone	No	
CSSM_GenerateMac	No	
CSSM_GenerateMacInit	No	
CSSM_GenerateMacUpdate	No	
CSSM_GenerateMacFinal	No	
CSSM_VerifyMac	No	
CSSM_VerifyMacInit	No	
CSSM_VerifyMacUpdate	No	
CSSM_VerifyMacFinal	No	
CSSM_EncryptData1 (See Note) CSSM_EncryptDataInit CSSM_EncryptDataUpdate CSSM_EncryptDataFinal	Yes	Algorithms/modes supported: See Table 2 on page 34
CSSM_DecryptData (See Note) CSSM_DecryptDataInit CSSM_DecryptDataUpdate CSSM_DecryptDataFinal	Yes	Algorithms/modes supported: See Table 2 on page 34
CSSM_QueryKeySizeInBits	Yes	
CSSM_GenerateKey	Yes	Algorithms/Modes Supported: CSSM_ALGID_DES CSSM_ALGID_3DES_3KEY CSSM_ALGID_RC2 CSSM_ALGID_RC4 CSSM_ALGID_RC5
CSSM_GenerateKeyPair	Yes	Algorithms Supported: (see Note 1 on page 35) CSSM_ALGID_RSA CSSM_ALGID_DSA CSSM_ALGID_DSA_BSAFE CSSM_ALGID_DH

Table 1. IBM Software Cryptographic Service Provider OCSF Functions (continued)

Function	Suppported	Comments
CSSM_GenerateRandom	Yes	Algorithms Supported: CSSM_ALGID_MD2Random CSSM_ALGID_MD5Random
CSSM_GenerateAlgorithmParams	Yes	Algorithm Supported: (see Note 2 on page 35) CSSM_ALGID_DH
CSSM_WrapKey	No	
CSSM_UnwrapKey	No	
CSSM_DeriveKey	Yes	Algorithm Supported: (see Note 3 on page 35) CSSM_ALGID_DH
CSSM_CSP_PassThrough	No	
CSSM_CSP_Login	No	
CSSM_CSP_Logout	No	
CSSM_CSP_ChangeLoginPassword	No	

Note: The Cryptographic strength allowed is dependent on the policy module that you have the OCSF feature that you have installed.

Table 2. Algorithms/Modes Supported for CSSM_Encrypt and CSM_Decrypt Functions

Algorithm	Mode
CSSM_ALGID_RSA (See Note 1)	----
CSSM_ALGID_RSA_PKCS (See Note 2)	----
CSSM_ALGID_DES	CSSM_ALGMODE_CBCPadIV8 CSSM_ALGMODE_CBC_IV8 CSSM_ALGMODE_CBC
CSSM_ALGID_3DES_3KEY	CSSM_ALGMODE_CBCPadIV8 CSSM_ALGMODE_CBC_IV8 CSSM_ALGMODE_CBC
CSSM_ALGID_RC2	CSSM_ALGMODE_CBCPadIV8 CSSM_ALGMODE_CBC_IV8 CSSM_ALGMODE_CBC
CSSM_ALGID_RC4	CSSM_ALGMODE_NONE
CSSM_ALGID_RC5	CSSM_ALGMODE_CBCPadIV8 CSSM_ALGMODE_CBC_IV8 CSSM_ALGMODE_CBC
Note: 1. The input value must be less than the key size. 2. The total input must be no more than *k*-11 bytes long; where *k* is the key size length in bytes.	

Note:

1. **CSSM_GenerateKeyPair** - For CSSM_ALGID_RSA, the key attribute specified on the CSSM_GenerateKeyPair invocation determines the format of the key. If CSSM_KEYATTR_PERMANENT is specified then the key pair that is generated is in typical IBM software CSP key format. If CSSM_KEYATTR_SENSITIVE is specified then the key pair that is generated is in an ICSF (token) readable format. This format allows RSA key pairs to be generated by the software CSP which can be utilized by the IBM (hardware) CCA module.

   For CSSM_ALGID_DH, the public key contains the public part to be exchanged with the other side. The private key contains a temporary handle that is valid only during the attach session. The private key and the other side's public key will be input to the CSSM_DeriveKey to derive the agreed upon symmetric key.

   Invoke CSSM_CSP_CreateKeyGenContext in the IBM Software Cryptographic Service Provider with values in the KeyHeader set as follows:
   - KeyAttr for both private and public keys set to CSSM_KEYATTR_SENSITIVE
   - KeyUsage and KeySizeInBits set to the appropriate value.

Intended Use of Key	Key Usage	KeySizeInBits
OAEP SET Block Compose OAEP SET Block DeCompose	CSSM_KEYUSE_SIGN	1024
Wrap key	CSSM_KEYUSE_WRAP CSSM_KEYUSE_ANY	256-1024
Unwrap key	CSSM_KEYUSE_UNWRAP CSSM_KEYUSE_ANY	256-1024
Signature Generate	CSSM_KEYUSE_SIGN CSSM_KEYUSE_ANY	256-1024
Signature Verify	CSSM_KEYUSE_VERIFY CSSM_KEYUSE_ANY	256-1024

2. **CSSM_GenerateAlgorithmParams** - This function must be called with a KEYGEN context with the *Params* input of the CSSM_CSP_CreateKeyGenContext set to NULL. The output *Params* of this functions is then passed to another CSSM_CSP_CreateKeyGenContext to generate the Diffie-Hellman key pair.

   Generating a key pair for Diffie-Hellman requires an additional input called key generation parameters. These are usually supplied from an external source, but if they are not, you need to generate them by:

   a. Invoking CSSM_CSP_CreateKeyGenContext with *Params* set to NULL.
   b. Invoking CSSM_GenerateAlgorithParams. The output *Params* from this function contains the key generation parameters.
   c. Deleting the KeyGenContext built in step a; you will need a new KeyGenContext to generate the Diffie-Hellman key pair itself.

   A similar requirement exists for DSA, where the extra parameters are sometimes called **network values**. If you don't already have the key generation parameters, you need to perform the same three steps as for Diffie-Hellman.
3. **CSSM_DeriveKey** - The *BaseKey* parameter should be set to the private key returned from the CSSM_GenerateKeyPair function. *Param* should be set to the public key received from the other side of the key exchange operation.

IBM Weak Software Cryptographic Service Provider, Version 1.0

Regarding Use of the IBM Weak Software Cryptographic Service Provider

Portions of the IBM Weak Software Cryptographic Service Provider contained in the Open Cryptographic Services Facility base of OS/390 contain software code provided by RSA Data Security, Inc.

Prior to utilizing the OS/390 Open Cryptographic Services APIs of the IBM Weak Software Cryptographic Service Provider functionality contained in the OCSF base for purposes of development and test only, you must provide your company name, company contact name, address and telephone number to RSA Data Security, Inc. (RSA), by sending this information to:

Email: sales@rsa.com <mailto:sales@rsa.com> or

RSA Data Security, Inc.
2955 Campus Drive, Suite 400
San Mateo, CA 94403-2507
Attention: SALES or

FAX: 650-295-7770
Attention: SALES.

Prior to using (except for test or development purposes), marketing, selling or distributing applications developed by you that directly utilize the Open Cryptographic Services Facility Cryptographic Services APIs of the IBM Weak Software Cryptographic Service Provider functionality (i.e., utilizing the Open Cryptographic Services Facility Cryptographic Services APIs of the IBM Weak Software Cryptographic Service Provider contained in the OCSF base), you must first obtain (if you have not already done so) a license from RSA for that application.

The files required for the IBM Weak Software Cryptographic Service Provider, Version 1.0 are:
- ibmwkcsp.so
- Ibmwkcsp.h

The Weak Software Cryptographic Provider offers the same OCSF API functions as the Software Cryptographic Service Provider (see Table 1 on page 33), except for DES and 3DES_3KEY.

The maximum cryptographic strengths allowed are 40 bit for RC2, RC4, and RC5. The maximum cryptographic strengths allowed are 512 bits for RSA and DSA requests.

IBM Software Cryptographic Service Provider 2, Version 1.0

The files required for the IBM Software Cryptographic Service Provider 2 , Version 1.0 are:
- ibmswcsp2.so
- ibmswcsp2.h

The IBM Software Cryptographic Service Provider 2 module provides cryptographic functionality. Table 3 on page 37 lists the OCSF API functions supported by this module.

All functions that require input/output buffers support only one buffer at a time and not a vector of buffers. If an application provides a buffer to the CSP module, it must also specify the buffer length. On return from an OCSF API function, the length field of an output buffer will be set to the length of returned data. If an output buffer's length is set to zero and its data pointer is set to NULL, the CSP will allocate the needed memory on the application behalf. It is the responsibility of the application to free this memory when done. Encryption/Decryption in place is not supported. That is, the same buffer may not be supplied as both input and output to the Encryption and Decryption functions.

For encryption/decryption operations, there are two operative contexts:
- Symmetric or asymmetric
- Key generation.

The effective bits attribute for RC2 must be set in the symmetric context, not the key generation context. The value of either parameter is passed as the Params input to CSSM_CSP_CreateSymmetricContext or to CSSM_UpdateContextAttributes.

Table 3. IBM Software Cryptographic Service Provider 2 OCSF Functions

Function	**Suppported**	**Comments**
CSSM_QuerySize	No	
CSSM_SignData CSSM_SignDataInit CSSM_SignDataUpdate CSSM_SignDataFinal	Yes	Algorithms Supported: CSSM_ALGID_MD2WithRSA CSSM_ALGID_MD5WithRSA CSSM_ALGID_SHA1WithRSA CSSM_ALGID_SHA1WithDSA
CSSM_VerifyData CSSM_VerifyDataInit CSSM_VerifyDataUpdate CSSM_VerifyDataFinal	Yes	Algorithms Supported: CSSM_ALGID_MD2WithRSA CSSM_ALGID_MD5WithRSA CSSM_ALGID_SHA1WithRSA CSSM_ALGID_SHA1WithDSA
CSSM_DigestData CSSM_DigestDataInit CSSM_DigestDataUpdate CSSM_DigestDataFinal	Yes	Algorithms Supported: CSSM_ALGID_MD2 CSSM_ALGID_MD5 CSSM_ALGID_SHA1
CSSM_DigestDataClone	No	
CSSM_GenerateMac	No	
CSSM_GenerateMacInit	No	
CSSM_GenerateMacUpdate	No	
CSSM_GenerateMacFinal	No	
CSSM_VerifyMac	No	
CSSM_VerifyMacInit	No	
CSSM_VerifyMacUpdate	No	
CSSM_VerifyMacFinal	No	

Table 3. IBM Software Cryptographic Service Provider 2 OCSF Functions (continued)

Function	Suppported	Comments
CSSM_EncryptData1 (See Note) CSSM_EncryptDataInit CSSM_EncryptDataUpdate CSSM_EncryptDataFinal	Yes	Algorithms/modes supported: See Table 4 on page 39. For algorithm CSSM_ALGID_RSA, repeated calls to CSSM_EncryptDataUpdate accumulate cleartext data, but do not perform any encryption until CSSM_EncryptDataFinal is called.
CSSM_DecryptData (See Note) CSSM_DecryptDataInit CSSM_DecryptDataUpdate CSSM_DecryptDataFinal	Yes	Algorithms/modes supported: See Table 4 on page 39
CSSM_QueryKeySizeInBits	Yes	
CSSM_GenerateKey	Yes	Algorithms/Modes Supported: CSSM_ALGID_DES CSSM_ALGID_3DES_3KEY CSSM_ALGID_RC2 CSSM_ALGID_RC4
CSSM_GenerateKeyPair	Yes	Algorithms Supported: (see Note 1 on page 40) CSSM_ALGID_RSA (Key length 362-2048, even numbers only) CSSM_ALGID_DSA (Key length 512, 1024, 2048)CSSM_ALGID_DH
CSSM_GenerateRandom	Yes	Algorithms Supported: CSSM_ALGID_SHARandom is the only algorithm used to generate a random number. However, to maintain compatibility with SWCSP, CSSM_ALGID_MD2Random and CSSM_ALGID_MD5Random are accepted without error. The number generated will use CSSM_ALGID_SHARandom, regardless.
CSSM_GenerateAlgorithmParams	Yes	Algorithm Supported: (see Note 2 on page 40) CSSM_ALGID_DH
CSSM_WrapKey	No	
CSSM_UnwrapKey	No	
CSSM_DeriveKey	Yes	Algorithm Supported: (see Note 3 on page 40) CSSM_ALGID_DH
CSSM_CSP_PassThrough	No	

Table 3. IBM Software Cryptographic Service Provider 2 OCSF Functions (continued)

Function	Suppported	Comments
CSSM_CSP_Login	No	
CSSM_CSP_Logout	No	
CSSM_CSP_ChangeLoginPassword	No	

Note: The Cryptographic strength allowed is dependent on the policy module that you have the OCSF feature that you have installed.

Table 4. Algorithms/Modes Supported for CSSM_Encrypt and CSM_Decrypt Functions

Algorithm	Mode
CSSM_ALGID_RSA (See Note 1 on page 40)	----
CSSM_ALGID_RSA_PKCS (See Note 1 on page 40)	Same as CSSM_ALGID_RSA.
CSSM_ALGID_DES	CSSM_ALGMODE_CBCPadIV8 CSSM_ALGMODE_CBC_IV8
CSSM_ALGID_3DES_3KEY	CSSM_ALGMODE_CBCPadIV8 CSSM_ALGMODE_CBC_IV8 CSSM_ALGMODE_CBC
CSSM_ALGID_RC2	CSSM_ALGMODE_CBCPadIV8 CSSM_ALGMODE_CBC_IV8 The key size in bits must be larger than the 'effective key size' specified on the call to CSSM_CSP_CreateSymmetricContext(). Otherwise, encrypt/decrypt operations will fail with CSSM_INVALID_KEY_LENGTH error.
CSSM_ALGID_RC4	CSSM_ALGMODE_NONE

Note:

1. The total input must be no more than *k*-11 bytes long; where *k* is the key size length in bytes. Supports encryption with public key and decryption with private key only. Does not support encryption with private key or decryption with public key.

Note:

1. **CSSM_GenerateKeyPair** - For CSSM_ALGID_RSA, the key attribute specified on the CSSM_GenerateKeyPair invocation determines the format of the key. If CSSM_KEYATTR_PERMANENT is specified then the key pair that is generated is in typical IBM software CSP key format. If CSSM_KEYATTR_SENSITIVE is specified then the key pair that is generated is in an ICSF (token) readable format. This format allows RSA key pairs to be generated by the software CSP which can be utilized by the IBM (hardware) CCA module.

   For CSSM_ALGID_DH, the public key contains the public part to be exchanged with the other side. The private key contains a temporary handle that is valid only during the attach session. The private key and the other side's public key will be input to the CSSM_DeriveKey to derive the agreed upon symmetric key.

   Invoke CSSM_CSP_CreateKeyGenContext in the IBM Software Cryptographic Service Provider with values in the KeyHeader set as follows:
   - KeyAttr for both private and public keys set to CSSM_KEYATTR_SENSITIVE
   - KeyUsage and KeySizeInBits set to the appropriate value.

Intended Use of Key	Key Usage	KeySizeInBits
OAEP SET Block Compose OAEP SET Block DeCompose	CSSM_KEYUSE_SIGN	1024
Wrap key	CSSM_KEYUSE_WRAP CSSM_KEYUSE_ANY	256-1024
Unwrap key	CSSM_KEYUSE_UNWRAP CSSM_KEYUSE_ANY	256-1024
Signature Generate	CSSM_KEYUSE_SIGN CSSM_KEYUSE_ANY	256-1024
Signature Verify	CSSM_KEYUSE_VERIFY CSSM_KEYUSE_ANY	256-1024

2. **CSSM_GenerateAlgorithmParams** - Generating a key pair for Diffie-Hellman requires an additional input called key generation parameters.

   These are usually supplied from an external source, but if they are not, you need to generate them by:

   a. Invoking CSSM_CSP_CreateKeyGenContext with *Params* set to NULL.
   b. Invoking CSSM_GenerateAlgorithParams. The output *Params* from this function contains the key generation parameters.
   c. Deleting the KeyGenContext built in step a; you will need a new KeyGenContext to generate the Diffie-Hellman key pair itself.

3. **CSSM_DeriveKey** - The *BaseKey* parameter should be set to the private key returned from the CSSM_GenerateKeyPair function. *Param* should be set to the public key received from the other side of the key exchange operation. The *DeriveKeyLength* parameter in CSSM_CSP_CreateDeriveKeyContext is ignored. The derived key length is equal to the length of the private key supplied in the BaseKey parameter.

IBM Weak Software Cryptographic Service Provider 2, Version 1.0

The files required for the IBM Weak Software Cryptographic Service Provider 2, Version 1.0 are:
- ibmwkcsp2.so
- ibmwkcsp2.h

The Weak Software Cryptographic Provider 2 offers the same OCSF API functions as the Software Cryptographic Service Provider 2 (see Table 3 on page 37), except for DES and 3DES_3KEY.

The maximum cryptographic strengths allowed are 40 bit for RC2 and RC4. The maximum cryptographic strengths allowed are 512 bits for RSA and DSA requests.

IBM CCA Cryptographic Module Version 1.0

Note: The IBM CCA Cryptographic Module, Version 1.0, is always installed when the OCSF Installation Script is run (see "Running the Installation Script" on page 5). However, the function provided by the IBM CCA Cryptographic Module Version 1.0 is available to your application only if the Cryptographic Hardware feature is installed on your processor. Additionally, the z/OS Integrated Cryptographic Service Facility (ICSF) must be installed, configured to run with the Cryptographic Hardware feature, and must be active. Refer to the *z/OS Cryptographic Services ICSF Administrator's Guide*, SA22-7521, for more information. For ICSF error codes not related to OCSF activities and for more detailed information on ICSF data and functions, refer to the *z/OS Cryptographic Services ICSF Application Programmer's Guide*, SA22-7522.

The files required by the IBM CCA Cryptographic Module, Version 1.0 are:
- ibmcca.so
- ibmcca.h

The IBM Common Cryptographic Architecture (CCA) Cryptographic Module provides cryptographic capabilities to OCSF applications running in a UNIX System Services environment. Table 3 lists the OCSF API functions that this module supports. The IBM CCA Cryptographic Module relies on the Integrated Cryptographic Services Facility (ICSF) and its underlying cryptographic hardware to provide its services. It currently supports these capabilities:

- Data digesting using MD5 and SHA-1 hashing algorithms (CSSM_ALGID_MD5 and CSSM_ALGID_SHA1)
- Generation of random numbers
- DES encryption/decryption algorithm (CSSM_ALGID_DES). These encryption/decryption modes (one of which must be explicitly included into the correspondent cryptographic context) are supported:
  - CSSM_ALGMODE_CBC
  - CSSM_ALGMODE_CBC_IV8
  - CSSM_ALGMODE_CBCPadIV8

  If CSSM_ALGMODE_CBC or CSSM_ALGMODE_CBC_IV8 is used during encryption, the length of the data must be an integral multiple of 8 bytes.
- Capability of wrapping single or double or triple-length DES keys algorithms
- RSA key pairs up to 1024 bits long for these operations:
  - Signature/verification
  - DES key exchange

  These RSA family algorithms are supported:
  - CSSM_ALGID_RSA_PKCS
  - CSSM_ALGID_RSA_ISO9796
- Data encryption/decryption using RSA Optimal Asymmetric Encryption Padding (OAEP) algorithm (part of Secure Electronic Transaction (SET) protocol) – CSSM_ALGID_WrapSET_OAEP. The optional encryption hashing mode

supported for this algorithm is CSSM_ALGMODE_OAEP_HASH. If the mode is not specified, encryption using default (non-hashing) mode is performed.

Multiple buffers are not supported during encryption and decryption operations. Although encryption/decryption using the RSA OAEP algorithm makes use of two buffers, these buffers have a different significance than described in this information. (See the description of the CSSM_EncryptData() and CSSM_DecryptData() functions.)

If a function expects a CSSM_DATA structure as a parameter describing the output, and the Length element is zero and Data element is NULL, then the necessary memory will be allocated by the function. If the user specifies a CSSM_DATA structure then it is the user's responsibility to ensure that the Length element specified matches the length of the Data block allocated. Failure to do will produce PROTECTION EXCEPTIONs.

Table 5. IBM CCA Cryptographic Module OCSF Functions

Cryptographic Library Functions		
Function Name	**Supported**	**Comments**
CSSM_QuerySize	Yes	See Note 1 on page 43.
CSSM_SignData	Yes	See Note 2 on page 43.
CSSM_SignDataInit	Yes	
CSSM_SignDataUpdate	Yes	
CSSM_SignDataFinal	Yes	
CSSM_VerifyData	Yes	See Note 3 on page 44.
CSSM_VerifyDataInit	Yes	See Note 3 on page 44.
CSSM_VerifyDataUpdate	Yes	
CSSM_VerifyDataFinal	Yes	
CSSM_DigestData CSSM_DigestDataInit CSSM_DigestDataUpdate CSSM_DigestDataFinal	Yes	Algorithms Supported: CSSM_ALGID_MD5 CSSM_ALGID_SHA1
CSSM_DigestDataClone	Yes	
CSSM_GenerateMac CSSM_GenerateMacInit CSSM_GenerateMacUpdate CSSM_GenerateMacFinal	Yes	Algorithms Supported: CSSM_ALGID_DES
CSSM_VerifyMac CSSM_VerifyMacInit CSSM_VerifyMacUpdate CSSM_VerifyMacFinal	Yes	Algorithms Supported: CSSM_ALGID_DES
CSSM_EncryptData CSSM_EncryptDataInit CSSM_EncryptDataUpdate CSSM_EncryptDataFinal	Yes	See Note 4 on page 44.
CSSM_DecryptData	Yes	See Note 5 on page 44.
CSSM_DecryptDataInit CSSM_DecryptDataUpdate CSSM_DecryptDataFinal	Yes	See Note 6 on page 45.
CSSM_QueryKeySizeInBits	Yes	

Table 5. IBM CCA Cryptographic Module OCSF Functions (continued)

Cryptographic Library Functions		
Function Name	**Supported**	**Comments**
CSSM_GenerateKey	Yes	No DES wrap support (import/export). For DES, Encrypt, Decrypt support only. See Note 7 on page 45.
CSSM_GenerateKeyPair	Yes	See Note 8 on page 45.
CSSM_GenerateRandom	Yes	
CSSM_GenerateAlgorithmParams	No	
CSSM_WrapKey CSSM_UnwrapKey	Yes	See Note 9 on page 46. See Note 10 on page 46.
CSSM_DeriveKey	No	
CSSM_CSP_PassThrough	No	
CSSM_CSP_Login	No	Does not apply. ICSF is a started task, you do not log into or out of it.
CSSM_CSP_Logout	No	
CSSM_CSP_ChangeLoginPassword	No	

Note:

1. **CSSM_QuerySize** - In addition to the conventional usage, this function may be used in order to find out the sizes of the necessary output buffers for the RSA OAEP encryption/decryption. In order to do this, an application must set the ContextType field of the Context parameter to CSSM_ALGCLASS_ASYMMETRIC. The function will expect these input parameters:
   - DataBlock should be an array of two CSSM_QUERY_SIZE_DATA structures.

   These values are expected in these structures on input and stored there on output.

   if Encrypt parameter equals CSSM_TRUE:

	Input	**Output**
Block 1	Size of plain text data	Size of encrypted data
Block 2	Size of XDATA	Size of OAEP block

   if Encrypt parameter equals CSSM_FALSE:

	Input	**Output**
Block 1	Size of encrypted data	Size of decrypted data
Block 2	Size of OAEP block	Size of XDATA

2. **CSSM_SignData** - The SignData and VerifyData services allow you to sign or verify using a digest. To sign or verify using a digest, the Context algorithm must be specified as CSSM_ALGID_RSA. (The algorithm CSSM_ALGID_RSA applies only to CSSM_SignData and CSSM_VerifyData. It does not apply to DataInit, DataUpdate, or DataFinal functions for Sign or Verify.) Using the CSSM_ALGID_RSA algorithm, SignData assumes the data passed is a digest. It encrypts the data with the RSA private key using PKCS 1.1 formatting.

3. **CSSM_VerifyData, CSSM_VerifyDataInit** - In addition to standard verification, verification of a RSA signature using a clear RSA key is supported. The RSA key has to have been inserted into the encryption context as the CSSM_ATTRIBUTE_KEY attribute. The *BlobType* element of the key header needs to be set to CSSM_KEYBLOB_RAW.
   - The SignData and VerifyData services allow you to sign or verify using a digest. To sign or verify using a digest, the Context algorithm must be specified as CSSM_ALGID_RSA. (The algorithm CSSM_ALGID_RSA applies only to CSSM_SignData and CSSM_VerifyData. It does not apply to DataInit, DataUpdate, or DataFinal functions for Sign or Verify.) The VerifyData function decrypts the algorithm using the RSA public key. It recovers the digest from the PKCS 1.1 formatting and compares it to the digest (data) provided. When generating key pairs for signing, it is necessary to specify KeyUsage CSSM_KEYUSE_SIGN. If the key is used for other operations (such as, encryption) they must also be specified.
4. **CSSM_EncryptData** - Mulitiple input and output buffers are not supported.
   - Asymmetric encryption using RSA OAEP algorithm is supported. The significance of the parameters in this case is as follows. (See the *Secure Electronic Transaction* specification for additional information.)
     – The ClearBufCount and CipherBufCount parameters should both equal 2.
     – The first (index 0) ClearBuf buffer should contain BC byte at the offset 0, and XDATA starting at the offset of 1.
     – The second (index 1) ClearBuf buffer should contain the data to be encrypted.
     – The OAEP block will be stored in the first (index 0) CipherBuf.
     – The encrypted data will be stored in the second (index 1) CipherBuf.

     (See also the CSSM_DecryptData() function description in Note 5.)
   - In addition to standard encryption, symmetric encryption using clear single length (8 bytes) DES key is supported. The DES key has to have been inserted into the encryption context as the CSSM_ATTRIBUTE_KEY attribute. The *BlobType* element of the key header needs to be set to CSSM_KEYBLOB_RAW.
   - Support is provided for data encryption using an RSA key. The data in the ClearBuf is encrypted using the RSA key in the Context. The length of the data cannot exceed the size of the RSA key (modulus length). The CiphBuf must be at least the size of the RSA key. The data must be formatted using the PKCS-1.2 algorithm. These Context quantities must be specified for data encryption using an RSA key:
     AlgorithmType = CSSM_ALGID_RSA_PKCS or CSSM_ALGID_RSA
     Key AlgorithmId = CSSM_ALGID_RSA_PKCS

   CSSM_EncryptDataInit - In addition to standard encryption, symmetric encryption using clear single length (8 bytes) DES key is supported. The DES key has to have been inserted into the encryption context as the CSSM_ATTRIBUTE_KEY attribute. The *BlobType* element of the key header needs to be set to CSSM_KEYBLOB_RAW.

   CSSM_EncryptDataUpdate, CSSM_EncryptDataFinal - Multiple input and output buffers are not supported.
5. **CSSM_DecryptData** - Mulitple input/output buffers are not supported.
   - Asymmetric decryption using RSA OAEP algorithm is supported. The significance of the parameters in this case is as follows (see the *Secure Electronic Transaction* specification for additional information):
     – The ClearBufCount and CipherBufCount parameters should both equal 2.

- The first (index 0) CipherBuf contains the OAEP block.
- The second (index 1) CipherBuf contains the encrypted data. The Output CipherBuf buffers from CSSM_EncryptData() my be supplied without any modifications as parameters for CSSM_DecryptData()).
- After decryption, BC byte will be stored at the offset 0 of the first (index 0) ClearBuf buffer, and XDATA will be stored in the same buffer starting at the offset of 1 byte.
- The decrypted data will be stored in the second (index 1) ClearBuf buffer.

Because of the specifics of the SET implementation, the length returned for the first (index 0) ClearBuf is always going to be 95 regardless of the actual size of the XDATA supplied during the encryption. It is therefore recommended that an application initialize this buffer with zeros before comparing it with the XDATA supplied as input for CSSM_EncryptData(). (See also the CSSM_EncryptData() function description.)

- In addition to standard decryption, symmetric decryption using clear single length (8 bytes) DES key is supported. The DES key has to have been inserted into the decryption context as the CSSM_ATTRIBUTE_KEY attribute. The *BlobType* element of the key header needs to be set to CSSM_KEYBLOB_RAW.
- Support is provided for data decryption using an RSA key. The data in the CiphBuf is decrypted using the RSA key in the Context. The length of the ciphered text cannot exceed the size of the RSA key. The data must be formatted using the PKCS-1.2 algorithm. These Context quantities must be specified for data decryption using an RSA key:
  AlgorithmType = CSSM_ALGID_RSA_PKCS or CSSM_ALGID_RSA
  Key AlgorithmId = CSSM_ALGID_RSA_PKCS

6. **CSSM_DecryptDataInit** - Symmetric decryption using clear single length (8 bytes) DES key is supported. The DES key has to have been inserted into the decryption context as the CSSM_ATTRIBUTE_KEY attribute. The *BlobType* element of the key header needs to be set to CSSM_KEYBLOB_RAW.

   CSSM_DecryptDataUpdate, CSSM_DecryptDataFinal - Multiple input and output buffers are not supported.
7. **CSSM_GenerateKey** - For use in generating regular DES keys (keys with CSSM_KEYUSE_ENCRYPT and CSSM_KEYUSE_DECRYPT key usage properties), this function can not be used for wrapping or unwrapping keys.
8. **CSSM_GenerateKeyPair** - RSA key generation can also be accomplished by the IBM software CSP service provider by specifying the CSSM_KEYATTR_SENSITIVE key attribute on the CSSM_GenerateKeyPair invocation. See "CSSM_GenerateKeyPair" on page 157.

   The key pair generated is an RSA Internal Private key in Modulus Exponent form. This key can be used in all of the cryptographic services allowed in Table 5 on page 42. It cannot be used in the Certificate Library calls. The IBM software CSP service provider should be used to generate keys for use in CL calls.

   When generating key pairs, the fields for Public Key Usage and Private Key Usage must be specified.

Intended Use of Key	Key Usage	KeySizeInBits
OAEP SET Block Compose OAEP SET Block DeCompose	CSSM_KEYUSE_SIGN	512-1024
Wrap key	CSSM_KEYUSE_WRAP	512-1024

Intended Use of Key	Key Usage	KeySizeInBits
Unwrap key	CSSM_KEYUSE_UNWRAP	512-1024
Signature Generate	CSSM_KEYUSE_SIGN	512-1024
Signature Verify	CSSM_KEYUSE_VERIFY	512-1024

9. **CSSM_WrapKey** - If the key to be wrapped is an RSA public key, it is exported "in the clear" to facilitate RSA public key exchange between cryptographic nodes. (See also the CSSM_UnwrapKey() function description.) If the key to be wrapped is a DES key the clear key value is recovered from the DES key internal format and encrypted under the RSA key provided.
10. **CSSM_UnwrapKey** - In addition to standard semantics, if the key to be unwrapped is a previously wrapped RSA public key (see the CSSM_WrapKey() function), it is imported into the module's internal format to facilitate RSA public key exchange between cryptographic nodes.

    An application imports a DES key into the modules internal format and imports an RSA public key as an RSA public key value. An appropriate CSSM_KEY structure must be supplied as a wrapped key parameter. The *BlobType* element of the key header needs to be set to CSSM_KEYBLOB_RAW for both DES and RSA clear keys. Additionally, for clear RSA public keys the *Format* element of the key header has to be as shown in Table 6.

Table 6. CSSM_Key Function

Keyblob Format	KeyData.Data Points To
CSSM_KEYBLOB_RAW_FORMAT_CDSA	CSSM_RSA_PUBLIC structure
CSSM_KEYBLOB_RAW_FORMAT_CCA	Structure containing an RSA public key stored in CCA internal format

IBM Standard Trust Policy Library, Version 1.0

The files required for the IBM Standard Trust Policy Library, Version 1.0 are:
- ibmtp.so
- ibmtp.h

The IBM Standard Trust Policy Library provides a simple generic service for verifying chains of X.509 certificates. The current version does not support operations that require DL operations. This module expects X.509 Version 3 signed certificates in ASN/DER-encoded format. In order to verify a given certificate, the application should supply the complete chain (see Table 7). This is to be used in conjunction with the IBM Certificate Library, Version 1.0 service provider and the IBM Software Service Cryptographic Provider, Version 1.0.

Table 7. IBM Standard Trust Policy Library OCSF Functions

Functions	Supported	Comments
CSSM_TP_CertSign	No	
CSSM_TP_CertRevoke	No	
CSSM_TP_CrlSign	No	
CSSM_TP_CrlVerify	No	
CSSM_TP_ApplyCrlToDb	No	
CSSM_TP_CertGroupConstruct	No	
CSSM_TP_CertGroupPrune	No	

Table 7. IBM Standard Trust Policy Library OCSF Functions (continued)

Functions	Supported	Comments
CSSM_TP_CertGroupVerify	Yes	See Note 1
CSSM_TP_PassThrough	No	

Note:

1. **CSSM_TP_CertGroupVerify** - The application should supply one anchor certificate and an *ordered* chain of certificates in the CertToBeVerified argument.

   These function arguments are ignored: Evidence, EvidenceSize, Action, policyIdentifers, NumberOfPolicyIdentifiers, VerificationAbortOn, VerifyScope, ScopeSize, DBList, Data.

   This function returns these error codes as shown in Table 8.

Table 8. CSSM_TP_CertGroupVerify Error Codes

Error Code	Description
CSSM_TP_INVALID_TP_HANDLE	TPHandle argument is NULL or invalid.
CSSM_TP_INVALID_CL_HANDLE	CLHandle argument is NULL or invalid.
CSSM_TP_INVALID_CSP_HANDLE	CSPHandle argument is NULL or invalid.
CSSM_TP_INVALID_DATA_POINTER	CertToBeVerified argument is NULL or invalid. This argument is invalid if the length is set to 0 or the pointer to data is NULL.
CSSM_TP_INVALID_CC_HANDLE	This error occurs if TP is unable to create a cryptographic context using the supplied CSPHandle and the certificates.
CSSM_TP_ANCHOR_NOT_SELF_SIGNED	The supplied anchor certificate is not self-signed.
CSSM_TP_ANCHOR_NOT_FOUND	The supplied anchor certificate is not the anchor for any of the certificates in the supplied chain.
CSSM_TP_CERT_VERIFY_FAIL	The supplied certificate chain cannot be verified.

IBM Extended Trust Policy Library, Version 1.0

The files required for the IBM Extended Trust Policy Library, Version 1.0 are:
- ibmtp2.so
- ibmtp2.h

The previous files also need to be used in conjunction with these files:
- ibmcl2.so
- ibmcl2.h

Some additional requirements include:
- Lightweight Directory Access Protocol (LDAP) product
- An IBM Software Cryptographic Service Provider CSP and IBM DL modules
- An IBM Software Cryptographic Service Provider 2 CSP and IBM DL modules

The Extended Trust Policy Library validates X.509 Version 3 certificates and CRLs using two types of trust policies: Entrust and X.509. The module can accept the

complete certificate chain or an incomplete certificate chain. If the module receives an incomplete chain, it attempts to fill in the missing certificates by searching the associated data store. Table 9 lists the OCSF API functions that this module supports.

This module ignores these arguments in all TP API:

```
const CSSM_FIELD_PTR Scope,
uint32 ScopeSize
```

Trust Policy (ibmtp2) can query an LDAP server when verifying certain certificates. LDAP is used to find issuers of Entrust certificates when the issuers are not otherwise found from input or Data Library (DL). LDAP is necessary to find certificate revocation lists (CRLs) for certificates with CRL extensions. It is the responsibility of the application to log into the appropriate LDAP server before invoking TP services, and to log out afterwards.

Table 9. IBM Extended Trust Policy Library OCSF Functions

Function Name	Supported	Comments
CSSM_TP_CertSign	Yes	The argument pair (*SignScope*, *ScopeSize*) is ignored. This function takes the input *CertToBeSigned* as an unsigned X509 certificate and signs it entirely.
CSSM_TP_CertRevoke	Yes	The Reason argument is ignored.
CSSM_TP_CrlSign	Yes	The argument pair (*SignScope*, *ScopeSize*) is ignored. This function takes the input *CrlToBeSigned* as an unsigned CRL and signs it entirely. A NULL pointer must be passed as the value to the CLHandle argument.
CSSM_TP_CrlVerify	Yes	
CSSM_TP_ApplyCrlToDb	Yes	
CSSM_TP_CertGroupConstruct	No	
CSSM_TP_CertGroupPrune	No	
CSSM_TP_CertGroupVerify	Yes	The parameter values passed to this function must be set as follows: • The argument *PolicyIdentifiers* should be given as one of the four policies specified in ibmtp.h or queried from IBMTP_GUID by CSSM_GetModuleInfo. If zero, or more than one policy is given, the default policy (X.509 certificate verification policy) is followed. • The argument *VerificationAbortOn* is ignored. • The argument *Action* is left for the caller to perform. This function verifies only the certificates.
CSSM_TP_PassThrough	No	

IBM Certificate Library, Version 1.0

The files required for the IBM Certificate Library, Version 1.0 are:
- ibmcl.so
- ibmcl.h

This module is used in conjunction with one of the IBM Software Service Cryptographic Providers. This module performs X.509 Version 3 certificate operations. It provides a library of functions needed for creating, signing, verifying, and querying a certificate. The current version does not support X.509 Version 3 extensions. The IBM CL expects X.509 Version 3 signed certificates in ASN/DER-encoded format. It uses a set of object identifiers (OIDs) to exchange certificate information with the application. The list of supported OIDs is defined in the file, ibmcl.h, which should be included in every application that uses the services of IBM CL.

This example demonstrates the purpose and use of OIDs. If an application asks for the version of a given certificate, the CL builds the version object that is returned to the application as follows:

```
CSSM_FIELD_PTR   p_version;

/* p_version is a pointer to a generic structure containing FieldOid and
FieldValue. FieldOid contains a number that indicates the type of the field,
e.g. version, serial number, etc. FieldValue contains the actual data.
*/


/* allocate memory for p_version for the sizeof(CSSM_FIELD)...*/

 /* allocate memory for p_version->FieldOid for the sizeof(CSSM_OID)...*/
      P_version->FieldOid.Length=sizeof(unit32);
  * allocate memory for p_version->FieldOid.Data for the sizeof(unit32)...*/
      *(unit32 *)p_version->FieldOid.Data=IBMCL_OID_VERSION;

 /* allocate memory for p_version->FieldValue for the sizeof(CSSM_DATA)...*/
      P_version->FieldValue.Length=Version.length;
  /* allocate memory for p_version->FieldValue.Data for the sizeof(Version.Data)...*/
  Copy(Version.value, p_version->FieldValue.Data);
```

All fields are returned as unsigned character arrays, which in turn need to be cast to the appropriate type. The OID indicates the type of the field and the structure it should be cast to. This example shows an instance where OID is used to build the relevant data structure:

```
CSSM_FIELD_PTR     p_field;
X500Name      *p_name;

/* call a CL function to obtain some field in the Cert */

switch ( *p_field->FieldOid.Data) {
    case IBMCL_OID_VERSION:
    break;
case IBMCL_OID_ISSUER_NAME:
    /* cast to the correct structure */
    p_name = (X500Name *) p_field->FieldValue.Data;
    break;
default:
    break;
}
```

The IBM CL functions in Table 10 on page 50 comply with the information in Chapter 15, “Certificate Library Services API,” on page 207. Most of the functions return error codes that are specific to this implementation and not defined in the OCSF API. These error codes are defined in ibmcl.h and described as part of supported API functions. Also note that function arguments *Scope* and *ScopeSize* are

ignored. Moreover, in order to construct an X.500, the name-only country name (C), organization name (O), organization name unit (OU), and common name (CN) are supported.

Table 10. IBM Certificate Library OCSF Functions

Function	Supported	Comments
CSSM_CL_CertSign	Yes	See Table 13 on page 52 for the error codes.
CSSM_CL_CertVerify	Yes	See Table 14 on page 52 for the error codes.
CSSM_CL_CertCreateTemplate	Yes	See Note 1.
CSSM_CL_CertGetFirstFieldValue	Yes	The ResultHandle will always be set to NULL and the NumberOfMatchedFields will be set to 1 if any field is found, regardless of how many. See Table 15 on page 52 for the error codes.
CSSM_CL_CertGetNextFieldValue	No	
CSSM_CL_CertAbortQuery	No	
CSSM_CL_CertGetKeyInfo	Yes	This function returns the DER-encoded subject public key. The encoding contains the public key, algorithm ID, and parameters, if applicable (see Table 16 on page 53).
CSSM_CL_CertGetAllFields	Yes	See Note 2 on page 51.
CSSM_CL_CertImport	No	
CSSM_CL_CertExport	No	
CSSM_CL_CertDescribeFormat	No	
CSSM_CL_CrlCreateTemplate	No	
CSSM_CL_CrlSetFields	No	
CSSM_CL_CrlAddCert	No	
CSSM_CL_CrlRemoveCert	No	
CSSM_CL_CrlSign	No	
CSSM_CL_CrlVerify	No	
CSSM_CL_IsCertInCrl	No	
CSSM_CL_CrlGetFirstFieldValue	No	
CSSM_CL_CrlGetNextFieldValue	No	
CSSM_CL_CrlAbortQuery	No	

Note:

1. **CSSM_CL_CertCreateTemplate** - This function accepts the public key field in two formats:
   - If the key algorithm requires any parameters, they can be put in the template with a separate OID. Thus, the application can pass in three OIDs and the respective values:
     – IBMCL_OID_SUBJECT_PUB_KEY: The value is passed in as a string. The key should not be DER-encoded.

- IBMCL_OID_PUB_KEY_PARAMETERS: Data should point to the DER encoding of the parameters.
- IBMCL_OID_PUB_KEY_ALGID: Data indicates what algorithm ID is used for generating the key, e.g., CSSM_ALGID_RSA.

- The algorithm ID, parameters, and the key can be DER-encoded and passed in with OID IBMCL_OID_SUBJECT_PUB_KEY. There is no need to supply the other two OIDs.

  The template requires these fields in one of the two formats: signature algorithm ID, validity, subject name, issuer name, and subject public key. Validity is specified as an array of two CSSM_DATE elements. Index 0 should contain the start date and index 1 the end date of certificate validity. This function returns error codes as shown in Table 11.

Table 11. CSSM_CL_CertCreateTemplate Error Codes

Error Code	Description
CSSM_CL_INVALID_CL_HANDLE	CLHandle argument passed in is invalid.
CSSM_CL_INVALID_INPUT_PTR	CertTemplate argument passed in is NULL.
CSSM_CL_INVALID_DATA	NumberOfFields argument passed in is 0.
CSSM_CL_SIGN_ALGID_NOT_SUPPORTED	The supplied signature algorithm ID in the template is not supported by IBM CL.
CSSM_CL_INVALID_TEMPLATE	The given template is missing or contains an invalid pointer to one of these mandatory items: serial number, signature algorithm ID, validity, subject name, or subject public key. Also, if an extension or unique ID is present in the template, but the pointers are invalid, this error is returned.
CSSM_CL_ INVALID_ CERT_ISSUER_NAME	The supplied issuer name is invalid.
CSSM_CL_ MISSING_CERT_ISSUER_NAME	The field for issuer name is not present in the template. This field is required for creating a valid certificate.
CSSM_CL_KEY_ALGID_NOT_SUPPORTED	The supplied algorithm ID for the subject public key is not supported.
CSSM_CL_KEY_FORMAT_UNKNOWN	The supplied subject public key is not in the correct format.
CSSM_CL_CERT_CREATE_FAIL	Failed to DER encode the certificate. This error could be caused by invalid data in the template or memory problem.

2. **CSSM_CL_CertGetAllFields** - This function returns DER encoding of the unsigned part of the certificate; signature algorithm ID; parameters, if applicable; and the signature (length in bytes). To view the specific fields in the certificate, such as version or validity, use CSSM_CL_GetFirstFieldValue with the appropriate OID. If the signature algorithm ID is not recognized by IBM CL, it is set to CSSM_ALGID_NONE. The other fields, however, are still returned to the application. This function returns error codes as shown in Table 12 on page 52.

Table 12. CSSM_CL_CertGetAllFields Error Codes

Error Code	Description
CSSM_CL_INVALID_CL_HANDLE	CLHandle argument passed in is invalid.
CSSM_CL_INVALID_CERT_POINTER	Cert argument passed in is NULL.
CSSM_CL_CERT_GET_FIELD_VALUE_FAIL	Unable to decode the certificate correctly.
CSSM_MALLOC_FAILED	Failed to allocate memory in the application space.

Table 13. CSSM_CL_CertSign Error Codes

Error Code	Description
CSSM_CL_INVALID_CL_HANDLE	CLHandle argument passed in is invalid.
CSSM_CL_INVALID_CC_HANDLE	CCHandle argument passed in is invalid.
CSSM_CL_INVALID_CERT_POINTER	CertToBeSigned or SignerCert arguments are invalid.
CSSM_CL_INVALID_CONTEXT	Unable to obtain a valid context using the CCHandle passed in.
CSSM_CL_GET_KEY_ATTRIBUTE_FAIL	Unable to obtain a valid key attribute using the CCHandle passed in.
CSSM_CL_KEY_ALGID_NOT_SUPPORTED	The specified algorithm ID in the signature context is not supported.
CSSM_CL_CERT_SIGN_FAIL	The signature operation failed. This could be caused by invalid attributes in the signature context.
CSSM_CL_CERT_ENCODE_FAIL	Failed to DER encode the signed certificate. This error could be caused by memory problems or invalid context attributes.

Table 14. CSSM_CL_CertVerify Error Codes

Error Code	Description
CSSM_CL_INVALID_CL_HANDLE	CLHandle argument passed in is invalid.
CSSM_CL_INVALID_CC_HANDLE	CCHandle argument passed in is invalid.
CSSM_CL_INVALID_CERT_POINTER	Either CertToBeVerified or SignerCert argument is NULL.
CSSM_CL_CERT_VERIFY_FAIL	Failed to verify the signature on the certificate.
CSSM_CL_CERT_GET_FIELD_VALUE_FAIL	Failed to decode the CertToBeVerified correctly.
CSSM_MALLOC_FAILED	Failed to allocate memory.

Table 15. CSSM_CL_CertGetFirstFieldValue Error Codes

Error Code	Description
CSSM_CL_INVALID_CL_HANDLE	CLHandle argument passed in is invalid.
CSSM_CL_INVALID_CERT_POINTER	Cert argument passed in is NULL.
CSSM_CL_INVALID_INPUT_PTR	CertField or CertField->Data argument passed in is NULL.
CSSM_MALLOC_FAILED	Unable to allocate memory in the application space.

Table 15. CSSM_CL_CertGetFirstFieldValue Error Codes (continued)

Error Code	Description
CSSM_CL_FIELD_NOT_PRESENT	The requested field is not in the certificate.
CSSM_CL_KEY_ALGID_NOT_SUPPORTED	If the key field is requested, the algorithm ID is not supported.

Table 16. CSSM_CL_CertGetKeyInfo Error Codes

Error Code	Description
CSSM_CL_INVALID_CL_HANDLE	CLHandle argument passed in is invalid.
CSSM_CL_INVALID_CERT_POINTER	Cert argument passed in is NULL.
CSSM_CL_CERT_GET_KEY_INFO_FAIL	Failed to decode the cert and obtain the public key.
CSSM_MALLOC_FAILED	Failed to allocate memory in the application memory space.
CSSM_CL_KEY_ALGID_NOT_SUPPORTED	The algorithm id of the subject public key is not supported.

IBM Data Library, Version 1.0

The files required for the IBM Data Library, Version 1.0 are:
- ibmdl2.so
- ibmdl2.h

The IBM Data Library provides support for the persistence and retrieval of security-related objects to and from a flat-file database maintained in the local file system. This module is semantic-free and allows the application developer to define the database record structure and index. Table 15 lists the OCSF API functions that this module supports.

All errors returned by this module are reported as CSSM_DL_PRIVATE_ERROR. If an error occurs within this module, it is possible to determine the exact cause of the error by enabling exception logging. The environment variable IBMFILEDL_LOG may be set to a file in which all exceptions will be logged by this module. If an error occurs, it is possible to look in the specified file to get an object dump of the exception, which will indicate the file and line number where the error occurred thus allowing the module developer to determine the exact cause of the failure.

Table 17. IBM Data Library OCSF Functions

Function Name	Supported	Comments
CSSM_DL_Authenticate	Yes	See Note 1 on page 54.
CSSM_DL_DbOpen	Yes	See Note 2 on page 55.
CSSM_DL_DbClose	Yes	The *DLHandle* parameter must not be NULL. The *DBHandle* parameter must reference an opened data store.
CSSM_DL_DbCreate	Yes	See Note 3 on page 55.
CSSM_DL_DbDelete	Yes	See Note 4 on page 55.
CSSM_DL_DbImport	No	

Table 17. IBM Data Library OCSF Functions (continued)

Function Name	Supported	Comments
CSSM_DL_DbExport	No	
CSSM_DL_DbSetRecordParsingFunctions	Yes	See Note 5 on page 55.
CSSM_DL_DbGetRecordParsingFunctions	Yes	The *DLHandle* parameter must not be NULL. The *DbName* specifies the absolute or relative path name to the file data store containing the record parsing functions. This parameter must not be NULL.
CSSM_DL_GetDbNameFromHandle	Yes	*DLHandle* parameter must not be NULL. *DBHandle* parameter must reference an opened data store.
CSSM_DL_DataInsert	Yes	The *DLHandle*, *Attributes*, and *Data* parameters must not be NULL. The *DBHandle* parameter must reference an opened data store. The write access permissions flag must be true.
CSSM_DL_DataDelete	Yes	The *DLHandle* parameter must not be NULL. *DBHandle* must reference an opened data store. *UniqueRecordIdentifier* must not be NULL. The write access permissions flag must be true.
CSSM_DL_DataGetFirst	Yes	See Note 6 on page 56.
CSSM_DL_DataGetNext	Yes	See Note 7 on page 56.
CSSM_DL_FreeUniqueRecord	Yes	The *DLHandle* parameter must not be NULL. The *DBHandle* parameter is ignored.
CSSM_DL_AbortQuery	Yes	The *DLHandle* parameter must not be NULL. *DBHandle* must reference an opened data store. *ResultsHandle* must reference a valid query. The read access permissions flag must be true.
CSSM_DL_PassThrough	No	

Note:

1. **CSSM_DL_Authenticate** - The parameter values passed to this function must be set as follows:
   - DLHandle must not be NULL.
   - DBHandle must reference an opened data store.
   - AccessRequest must not be NULL.
   - UserAuthentication must not be NULL.
   - UserAuthentication->Credential must not be NULL.
   - UserAuthentication->Credential->Length must not be NULL.
   - UserAuthentication->Credential->Data must not be NULL.
   - The password will be passed in the Credential portion of the user authentication, and will be applied to the opened data store only if the password has changed.

- The access request flags are applied to the opened data store. Note that only read/write access flags are used in this module.

2. **CSSM_DL_DbOpen** - The parameter values passed to this function must be set as follows:
   - DbHandle must not be NULL.
   - DbName must not be NULL.
   - AccessRequest must not be NULL.
   - UserAuthentication must not be NULL.
   - UserAuthentication->Credential must not be NULL.
   - UserAuthentication->Credential->Length must not be NULL.
   - UserAuthentication->Credential->Data must not be NULL.
   - UserAuthentication->MoreAuthenticationData is ignored.
   - OpenParameters is ignored.
   - The DbName specifies the absolute or relative
   - The password is to be passed in the Credential portion of the user authentication.
3. **CSSM_DL_DbCreate** - The parameter values passed to this function must be set as follows:
   - DLHandle must not be NULL.
   - DbName must not be NULL.
   - DBInfo must not be NULL.In addition this DL does not support the transparent integrity option. Record Signing Implemented must be set to false and Signing Certificate and Signing CSP fields must be set to zero.
   - AccessRequest must not be NULL.
   - UserAuthentication must not be NULL.
   - UserAuthentication->Credential must not be NULL.
   - UserAuthentication->Credential->Length must not be NULL.
   - UserAuthentication->Credential->Data must not be NULL.
   - UserAuthentication->MoreAuthenticationData is ignored.
   - OpenParameters is ignored.
   - The DbName specifies the absolute or relative path name to the file data store to be created.
   - The password is to be passed in the Credential portion of the user authentication.
4. **CSSM_DL_DbDelete** - The parameter values passed to this function must be set as follows:
   - DLHandle must not be NULL.
   - DbName must not be NULL.
   - UserAuthentication must not be NULL.
   - UserAuthentication->Credential must not be NULL.
   - UserAuthentication->Credential->Length must not be NULL.
   - UserAuthentication->Credential->Data must not be NULL.
   - UserAuthentication->MoreAuthenticationData is ignored.
   - The DbName specifies the absolute or relative path name to the file data store to be deleted.
   - The password is to be passed in the Credential portion of the user authentication.
5. **CSSM_DL_DbSetRecord ParsingFunctions** - The parameter values passed to this function must be set as follows:
   - DLHandle must not be NULL.
   - DbName must not be NULL.
   - FunctionTable must not be NULL.
   - FunctionTable->RecordGetFirstFieldValue must not be NULL.
   - FunctionTable->RecordGetNextFieldValue must not be NULL.
   - FunctionTable->RecordAbortQuery must not be NULL.

- The DbName specifies the absolute or relative path name to the file data store to be have the record parsing functions manipulated.

6. **CSSM_DL_DataGetFirst** - The parameter values passed to this function must be set as follows:
   - DLHandle must not be NULL.
   - DBHandle must reference an opened data store.
   - Query must not be NULL.
   - Query->Conjunctive must equal CSSM_DB_NONE.
   - Query->NumSelectionPredicates must be 0 or 1.
   - Query->SelectionPredicate must not be NULL if Query->NumSelectionPredicates is 1.
   - ResultsHandle must be an allocated pointer.
   - EndOfDataStore must be an allocated pointer.
   - Attributes must be an allocated pointer.
   - Data must be an allocated pointer.
   - The read access permissions flag must be true.
   - Query->NumSelectionPredicates equals 1 denotes an indexed query for a given record type.
   - Query->NumSelectionPredicates equals 0 denotes a sequential query for a given record type.
7. **CSSM_DL_DataGetNext** - The parameter values passed to this function must be set as follows:
   - DLHandle must not be NULL.
   - DBHandle must reference an opened data store.
   - ResultsHandle must reference a valid query.
   - EndOfDataStore must be an allocated pointer.
   - Attributes must be an allocated pointer.
   - Data must be an allocated pointer.
   - The read access permissions flag must be true.

IBM LDAP Data Library, Version 1.0

The files required for the LDAP Data Library, Version 1.0 are:
- ldapdl.so
- ldapdl.h

The IBM LDAP Data Library provides access to generic and security-related objects (for example, certificates, certificate revocation lists) stored in LDAP-compliant directory servers. This module is semantic-free and allows the application developer to specify any attribute types specified in the schema of the destination LDAP server. Table 16 lists the OCSF LDAP Data Library API functions that this module supports.

All errors returned by this module are in ldapdl.h. If an error occurs within this module, it is possible to determine the exact cause of the error by enabling exception logging. The environment variable LDAPDL_LOG may be set to a file in which all exceptions will be logged by this module. If an error occurs, it is possible to look in the specified file to get an object dump of the exception, which will indicate the file and line number where the error occurred, therefore allowing the module developer to determine the exact cause of the failure. The use of the LDAP_DL log can supplement the information provided by CSSM since in some instances LDAP_DL can throw an exception without that necessarily resulting in a call to "CSSM_SetError".

Table 18. IBM LDAP Data Library OCSF Functions

Function Name	Supported	Comments
CSSM_DL_Authenticate	Yes	See Note 1
CSSM_DL_DbOpen	Yes	See Note 2 on page 58.
CSSM_DL_DbClose	Yes	The *DLHandle* parameter must not be NULL. The *DBHandle* parameter must reference an opened LDAP session.
CSSM_DL_DbCreate	No	
CSSM_DL_DbDelete	No	
CSSM_DL_DbImport	No	
CSSM_DL_DbExport	Yes	See Note 3 on page 58.
CSSM_DL_DbSetRecordParsingFunctions	No	
CSSM_DL_DbGetRecordParsingFunctions	No	
CSSM_DL_GetDbNameFromHandle	Yes	*DLHandle* parameter must not be NULL. *DBHandle* parameter must reference an opened LDAP session.
CSSM_DL_DataInsert	Yes	The *DLHandle*, *Attributes*, and *Data* parameters must not be NULL. *DBHandle* parameter must reference an opened LDAP session.
CSSM_DL_DataDelete	Yes	The *DLHandle* parameter must not be NULL. *DBHandle* must reference an opened LDAP session. *UniqueRecordIdentifier* must not be NULL.
CSSM_DL_DataGetFirst	Yes	See Note 4 on page 58.
CSSM_DL_DataGetNext	Yes	See Note 5 on page 59.
CSSM_DL_FreeUniqueRecord	Yes	The *DLHandle* parameter must not be NULL. The *DBHandle* parameter is ignored.
CSSM_DL_AbortQuery	Yes	The *DLHandle* parameter must not be NULL. *DBHandle* must reference an opened LDAP session. *ResultsHandle* must reference a valid query.
CSSM_DL_PassThrough	Yes	See Note 6 on page 59.

Note:

1. **CSSM_DL_Authenticate** - The parameter values passed to this function must be set as follows:
   - DLHandle must not be NULL.
   - DBHandle must reference an LDAP session for which authentication is being performed.
   - AccessRequest must not be used and can be set to NULL.
   - UserAuthentication must not be NULL.
   - UserAuthentication->Credential must not be NULL.
   - UserAuthentication->Credential->Length must not be NULL.

- UserAuthentication->Credential->Data must not be NULL. The data portion of UserAuthentication must also have been typecast from a pointer to *BindParms*, which contains the *dn* of the entry to bind as the authentication mechanism and the credentials.

The LDAP access control model is based on the identity of the client requesting access to the directory. The format and capabilities of access control information, however, is highly dependent on the server's implementation, which varies from system to system. It is therefore the responsibility of the caller to know in advance which access rights are associated with a given entry.

2. **CSSM_DL_DbOpen** - The parameter values passed to this function must be set as follows:
   - DbHandle must not be NULL.
   - DbName must not be NULL. It must be a null-terminated string containing either:
     a. A host name or dotted string representing the IP address of the target LDAP server, with optional port number separated by a colon, or
     b. An LDAP URL specifying the host/port of the LDAP server to connect and the dn of the entry to server as the default starting point for search operations.
   - AccessRequest is not used. It can be set to NULL.
   - UserAuthentication can be set to NULL if no credentials are required for the specified LDAP server. The data portion of UserAuthentication must have been typecast from a pointer to *BindParms*, which contains the *dn* of the entry to bind as the authentication mechanism and the credentials.
   - OpenParameters is ignored.
3. **CSSM_DL_DbExport** - The parameter values passed to this function must be set as follows:
   - DbHandle must not be NULL.
   - DbSourceName must be a null-terminated string containing either:
     a. A host name or dotted string representing the IP address of the target LDAP server, with optional port number separated by a colon, or
     b. An LDAP URL specifying the host/port of the LDAP server to connect and the *dn* identifying the rest of the subtree to be exported.
   - DbDestinationName must be the full path of the file which will contain a snapshot of the requested information subtree written in LDAP Data Interchange Format (LDIF).
   - InfoOnly is ignored.
   - UserAuthentication represents the caller's credential as required for authorization to list a subtree. If access control of the portion of the directory tree to be exported requires no additional credentials to perform this operation, then user authentication can be NULL.
4. **CSSM_DL_DataGetFirst** - The parameter values passed to this function must be set as follows:
   - DLHandle must not be NULL.
   - DBHandle must reference an opened LDAP session.
   - Query must not be NULL.
   - Query->Conjunctive can be CSSM_DB_NONE, CSSM_DB_AND, CSSM_DB_OR.
   - Query->SelectionPredicate must not be NULL if Query->NumselectionPredicates is 1 or more.
   - ResultsHandle must be an allocated pointer.
   - EndofDataStore must be an allocated pointer.
   - Attributes must be an allocated pointer.
   - Data must be an allocated pointer.

The query structure specifying the selection predicates are 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 and Data parameters.

Special Attribute Names:

URL - This attribute name, if specified, must be the only one. The attribute value will be taken to be an LDAP URL conforming to RFC XXXX. All other predicates will be ignored.

DL SEARCH SCOPE - This is a psuedo attribute indicating to DL the scope of the search. The attribute value is one of "BASE", "ONE", or "SUB", corresponding to base object search, one-level search and subtree search, respectively.

DL_SEARCH_BASE - This is a psuedo attribute indicating to the DL the starting point of the search. The attribute value should be the string representation of a DN.

5. **CSSM_DL_DataGetNext** - The parameter values passed to this function must be set as follows:
   - DLHandle must not be NULL.
   - DBHandle must reference an opened LDAP session.
   - ResultsHandle must reference a valid query.
   - EndOfDataStore must be an allocated pointer.
   - Attributes must be an allocated pointer.
   - Data must be an allocated pointer.
6. **CSSM_DL_PassThrough** - The parameter values passed to this function must be set as follows:
   - DLHandle must not be NULL.
   - DBHandle must reference an opened LDAP session.

Chapter 9. Developing Security Applications

Chapter 9, "Developing Security Applications" presents a high-level overview of the steps involved in creating an OCSF application to incorporate the encryption provided by the IBM OCSF.

Export

Any application you create and export or reexport from the U.S. utilizing the Open Cryptographic Services Facility Cryptographic Services may be subject to special export licensing requirements by the Bureau of Export Administration of the U.S. Department of Commerce.

Writing OCSF Applications

"Writing OCSF Applications" describes the structure of a typical OCSF application.

CSSM_Init

Applications must call CSSM_Init to initialize the OCSF framework. This must be done prior to calling any framework functions. CSSM_Init will determine if the active framework version is compatible with the one the application was built with. It will also define the memory functions that will be used for allocating and freeing storage for the application.

Memory Management

The OCSF memory management functions are a class of routines for reclaiming memory allocated by OCSF on behalf of an application from the OCSF memory heap. When OCSF allocates objects from its own heap and returns them to an application, the application must inform OCSF when it no longer requires the use of that object. Applications use specific APIs to free OCSF-allocated memory. When an application invokes an API free function, OCSF can choose to retain or free the indicated object depending on other conditions known only to OCSF. In this way, OCSF and applications work together to manage these objects in the OCSF memory heap.

Finding and Listing Service Providers

Before attaching a service module, an application can query the OCSF Framework registry using the CSSM_ListModules function to obtain information on the:

- Modules installed on the system
- Capabilities (and functions) implemented by those modules
- Globally Unique ID (GUID) associated with a given module.

Applications use this information to dynamically select a module for use. A multiservice module has multiple capability descriptions associated with it, at least one per functional area supported by the module. Some areas (such as Cryptographic Service Provider (CSP) and Trust Policy (TP)) may have multiple independent capability descriptions for a single functional area. There is one OCSF Framework registry entry for a multiservice module, which records all service

types for the module. OCSF returns all information about a module's capabilities when queried by the application. Each set of capabilities includes a type identifier to distinguish CSPinfo from CLinfo, etc.

Applications can query about the OCSF Framework itself. One function, CSSM_GetInfo, returns version information about the running OCSF Framework. Another function, CSSM_Init, verifies whether the OCSF Framework version the application expects is compatible with the currently running OCSF Framework version. The general function to query service provider module information also returns the module's version information.

Getting Service Provider Information

CSSM_GetModuleInfo can be used to determine if a specific service provider (represented by a GUID) provides the services required by the application. CSSM_ListModules can be used to get a list of installed GUIDs.

Attaching a Service Provider

Applications select the particular security services they will use by selectively attaching service provider modules. Each module has an assigned GUID and a set of descriptive attributes to assist applications in selecting appropriate modules for their use. A module can implement a range of services across the OCSF APIs (e.g., cryptographic functions, data storage functions) or a module can restrict its services to a single OCSF category of service (e.g., Certificate Library (CL) services only). Modules that span service categories are called multiservice modules.

Applications use a module's GUID to specify the module to be attached. The attach function, CSSM_ModuleAttach, returns a handle representing a unique pairing between the caller and the attached module. This handle must be provided as an input parameter when requesting services from the attached module. OCSF uses the handle to match the caller with the appropriate service module.

The calling application uses the handle to obtain all types of services implemented by the attached module. Figure 2 on page 12 shows how the handle for an attached Dual Provider service provider is used to perform cryptographic operations and persistent storage of certificates. The single handle value can be used as the CSPHandle in cryptographic operations and as the DLHandle in data storage operations.

Multiple calls to attach are viewed as independent requests. Each attach request returns separate, independent handles that do not share execution state. Service provider modules may be detached using the CSSM_ModuleDetach function. However, an application should not invoke this operation unless all requests to the target service provider have been completed.

Using Service Provider Functions

After attaching a service provider and obtaining a handle, the application may use APIs supported by the service provider, using the service provider's handle to direct the call to the proper provider.

Service Context Management

Security context management provides secured run-time caching of user-specific state information and secrets. Multistep cryptographic operations, such as staged hashing, require multiple calls to a CSP and the intermediate operation states must be managed. These intermediate states are stored in run-time data structures

known as security contexts. The OCSF API provides a number of context functions that applications can use to create, initialize, and cache security contexts.

Security contexts provide mechanisms that:

- Allow an application to use multiple CSPs concurrently.
- Allow an application to concurrently use different parameters for a single CSP algorithm.
- Support layered implementations in their transparent use of multiple CSPs or different algorithm parameters for the same CSP.
- Enable development of reentrant CSPs, layered services, and applications.

Applications retain handles to each security context used during execution. The context handle is a required input parameter to many security service functions. Most applications instantiate and use multiple security contexts. Only one context may be passed to a function, but the application is free to switch among contexts at will, or as required (even per function call).

An application may create multiple contexts directly or indirectly. Indirect creation may occur when invoking layered services, system utilities, TP modules, CL modules, or DL modules that create and use their own appropriate security context as part of the service they provide to the invoking application. Figure 4 on page 15 shows an example of a hidden security context. An application creates a context specifying the use of sec_context1. The application invokes func1 in the CL using sec_context1 as a parameter. The CL performs two calls to the CSP. For the call to func5, the hidden security context is used. For the call to func6, the application's security context, sec_context2, is passed as a parameter to the CSP.

These transparent contexts do not concern the application developer, as they are managed entirely by the layered service or service provider module that created them. Each process or thread that creates a security context is responsible for explicitly terminating that context.

OCSF provides a number of API functions to create security contexts. The function used and type of context created depends on the cryptographic operation being performed. For example, the CSSM_CSP_CreateSymmetricContext is used in cryptographic operations involving a symmetric key; the CSSM_CSP_CreateAsymmetricContext is used in operations involving an asymmetric key.

The CSSM_DeleteContext function is paired up with the create context functions. These functions are designed to be used by applications and force notify events to be sent to a service provider module.

In contrast, the CSSM_GetContext and CSSM_FreeContext functions are designed to be used by service provider modules since they do not generate events.

Multi-threaded Applications

The OCSF framework supports multi-threaded applications. Framework initialization creates mutexes that are used to protect critical sections. Service providers specify whether or not they are threadsafe and if one is not, the framework creates a mutex that is locked prior to passing control to that provider. Applications do not need to be aware that these mutexes exist. The framework locks and unlocks these as necessary.

Error Management

OCSF provides error management through the functions CSSM_InitError, CSSM_SetError, CSSM_GetError and CSSM_ClearError. When an application receives an API return code of CSSM_FAIL, it should call CSSM_GetError to determine the error code. CSSM_ClearError should be used to remove the current error code after processing the error. CSSM_InitError may be used to initialize the error structure and CSSM_SetError may also be used by an application if appropriate.

There is an error code for each application thread. The error APIs only affect the error code for the calling thread.

Building OCSF Applications

“Building OCSF Applications” describes building OCSF applications.

Include Files for OCSF Services

The necessary header files are in /usr/include and also in /usr/lpp/ocsf/include. Applications must include cssm.h and any header files for the service providers used by the application.

OCSF Libraries

The OCSF framework library is implemented as cssm32.dll, which resides in /usr/lpp/ocsf/lib and has a symbolic link in /usr/lib. The linkage to the dll is the cssm32.x exports file, which is located in /usr/lpp/ocsf/lib. The compiler's -L option specifies additional directories to be searched for libraries. The z/OS -L option does not find exports files, and so they must be explicitly linked with the application, in the same manner that object files (.o) are linked.

Service provider libraries are loaded dynamically during CSSM_ModuleAttach and are not specified during application build.

The sample makefile /usr/lpp/ocsf/samples/ocsf_baseivp/Makefile.os390 provides an example. Note that the sample specifies the compiler flags dll and sscom. The dll flag allows an application to refer to symbols exported by a dll through the exports file. The sscom flag (slash-slash-comments) allows C programs to use C++ "//" comments. Certain OCSF header files use "//" for comments. Both of these flags are required.

Running OCSF Applications

When running applications, the LIBPATH environment variable must be set correctly in order to access the OCSF framework and supporting libraries. For example:

```
LIBPATH=$LIBPATH:/usr/lib
```

File_encrypt Sample Application

The file_encrypt application is a sample program that shows how the OCSF API can be used to encrypt a clear file. The file_encrypt application demonstrates the details involved in encrypting files and illustrates the steps necessary to create any OCSF-based application. These steps include:

1. Initialize the OCSF framework.
2. Attach the necessary service provider modules.
3. Perform the desired security operations.
4. Detach the modules when they are no longer needed.

The source code for the file_encrypt application is shown in “FILE_ENCRYPT.H” on page 72. The file_encrypt application is written in C language and can be built and run in the z/OS UNIX System Services environment.

To run this application you must have installed on your system a Cryptographic Service Provider (CSP) module that supports Data Encryption Standard (DES). If you have not already done so, you can install the Cryptographic modules by running the setup programs for the OCSF. You also must have access to the z/OS C/C++ compiler and run-time library. Once you have compiled the application, you can run it from the command line by typing this statement:

```
/home/G123456 file_encrypt <filename>
```

where *filename* is a file that is 4096 bytes in size or less. The file_encrypt application will encrypt the input file and generate one output file, the encrypted file (*filename*.enc).

The sample demonstrates the client's process of performing strong encryption, followed by the decryption of the message. The OCSF API calls for both the client and server are listed in pseudocode, without proper arguments or other details. They are meant to give a general overview of the changes needed rather than show sample code.

The sample assumes that the session key has been generated outside of the OCSF Framework, and the key exchange has already been performed. For the case in which the session key needs to be distributed by using the OCSF Framework, a sample of Diffie-Hellman key exchange is provided in "Diffie-Hellman Key Exchange Scenario" on page 67.

OCSF API Calls

"OCSF API Calls" provides the OCSF API calls that may be used by an application in order to enable it for&tab; encryption. The file_encrypt application is assumed to use a client/server architecture and use an OCSF Cryptographic Service Provider. The OCSF and the selected OCSF feature must be installed and configured on your system prior to use.

Table 19. Client Application OCSF API Calls

OCSF API Function	Description
Application Startup:	
CSSM_Init	Initializes the framework and passes pointers to memory functions.
CSSM_ListModules(CSP)	Lists all installed Cryptographic Service Providers (CSPs).
CSSM_GetModuleInfo	For each installed CSP, get information about the services it provides.
CSSM_ModuleAttach(CSP)	Actually loads the CSP module.
Encryption:	
CSSM_CSP_CreateSymmetricContext	Specifies all information relevant to performing symmetric encryption, including algorithm, mode, key, and initialization vector.
CSSM_EncryptData	Encrypts the message to the server using the parameters specified in the symmetric context. If the application has requested an encryption strength greater than the policy allows, the request will be denied.
Transmission Send: (Not performed through framework)	Sends the ciphertext. Could be socket transmission or any other protocol. This need not change from the way the application previously transmitted data.
Clean Up: CSSM_ModuleDetach(CSP)	Unloads the crypto.

Table 20. Sever Application OCSF API Calls

OCSF API Function	Description
Application Startup:	Performs the same startup steps as the client program.
Transmission Receive:	
(Not performed through framework)	Receives the ciphertext from the client application.
Strong Decryption:	
CSSM_CSP_CreateSymmetricContext	Specifies all information for symmetric decryption.
CSSM_DecryptData	Decrypts the message from the client.
Clean Up:	Performs the usual OCSF cleanup.

Diffie-Hellman Key Exchange Scenario

"Diffie-Hellman Key Exchange Scenario" outlines the procedure for performing Diffie-Hellman key exchange on both the client and the server machine. These steps are in addition to those described in "OCSF API Calls" on page 66.

Table 21. Client Application OCSF API Calls

OCSF API Application	Description
Application Startup:	Client performs normal startup procedure.
Key Exchange: CSSM_GenerateAlgorithmParameters	Specifies that you are generating Diffie-Hellman key exchange parameters.
CSSM_CSP_CreateAsymmetricContext	Creates a context for key pair generation using the parameters generated.
CSSM_GenerateKeyPair	Creates a Diffie-Hellman asymmetric key pair.
Transmission Send: (Not performed by framework)	Sends the public key to the server.
CSSM_CSP_CreateDeriveKeyContext	Specifies the information required to derive a session key from the Diffie-Hellman key pair.
CSSM_DeriveKey	Derives the session key.
Encryption:	Client performs encryption and cleanup operations previously described.

Table 22. Server Application OCSF API Calls

OCSF API Application	Description
Application Startup:	Server performs normal startup procedure.
Transmission Receive: (Not performed by framework)	Receives the Diffie-Hellman public key from the client.
CSSM_CSP_CreateDeriveKeyContext	Specifies the information required to derive a session key from the Diffie-Hellman key sent by the client.
CSSM_DeriveKey	Derives the session key.
Decryption:	Server performs decryption and cleanup operations previously described.

File_encrypt Structure

"File_encrypt Structure" presents an overview of the file_encrypt structure. Program execution begins in main.c, which calls subroutines that are discussed in:

- ProcessArguments
- Initialize
- AttachCSPByAlgorithm
- GenerateContextAndEncrypt.

ProcessArguments:

Located in file: main.c

This routine simply checks the input entered by the end user. If too many or too few parameters were entered, ProcessArguments displays a message informing the user of the correct command format and exits. Otherwise, the pointer ClearFilename is set to the input character array and returned to main.

Initialize:

Located in file: initialize.c

This function demonstrates how to initialize the OCSF framework. First, the initialize function sets the Version data structure to the current version level. (CSSM_MAJOR and CSSM_MINOR are defined in cssmtype.h.) Next, the MemoryFuncs data structure is initialized to the memory management function wrappers declared at the beginning of the initialize.c file. Since applications may have their own procedures for creating, managing, and freeing memory, the MemoryFuncs table is the way these functions can be made available to OCSF and the service provider modules. Applications register memory functions with OCSF using CSSM_Init. They register memory functions with the service provider modules using CSSM_ModuleAttach.

Both the Version and the MemoryFuncs data structures are passed to the CSSM_INIT function in this statement:

CSSM_Init(&Version, &MemoryFuncs, NULL)

OCSF ensures the version information matches and stores a pointer to the MemoryFuncs table within the framework memory heap. This function should be called only once in any application.

AttachCSPByAlgorithm:

Located in file: attach.c

There are various levels of detail that applications can use when attaching to modules using the OCSF API. In the simplest case, an application can hardcode a particular module ID, a Globally Unique ID (GUID), so that it only works when a particular module is installed. A more flexible application can be designed to look into the installed list of modules and choose one based on some attribute it has such as capability, vendor name, hardware/software, etc.

In AttachCSPByAlgorithm, the list of installed software CSPs is searched to find one that supports the required algorithm. The function accepts two input parameters: a pointer to the CSP handle and an unsigned integer indicating the

type of cryptographic algorithm desired; in this case, CSSM_ALGID_DES. (The header file, cssmtype.h, defines the supported algorithms.)

The function first determines which cryptographic modules are currently installed by calling CSSM_ListModules in this statement:

pModuleList = CSSM_ListModules(CSSM_SERVICE_CSP, CSSM_TRUE)

This function generates a data structure of type CSSM_LIST and returns a pointer to that structure called pModuleList. The CSSM_LIST data structure contains a GUID/name pair for each of the currently installed modules that match the service mask for cryptographic modules CSSM_SERVICE_CSP. If there are no CSP modules installed, the CSSM_LIST.NumberOfItems element contains a zero.

When a module is installed on a system, it must provide certain information about itself. This information is stored in a series of data structures in the operating system registry facility. Module information is made available to OCSF applications through the CSSM_GetModuleInfo function call using this statement:

```
pModuleInfo = CSSM_GetModuleInfo(&(pModuleList->Items[i].GUID),
                                 CSSM_SERVICE_CSP,
                                 0,
                                 CSSM_INFO_LEVEL_ALL_ATTR);
```

CSSM_GetModuleInfo returns a pointer, pModuleInfo, to a data structure containing the module information. In the code that follows the CSSM_GetModuleInfo call, the system searches the module information retrieved for each module (using its GUID) for a match on CSSM_ALGID_DES. Once the appropriate module is found, CSSM_ModuleAttach is called, which returns a handle to that module. This statement is used:

```
*hCSP = CSSM_ModuleAttach(&(pModuleList->Items[i-1].GUID), /*module GUID*/
                          &pModuleInfo->Version, /*version info*/
                          &MemoryFuncs, /*MemoryFuncs table*/
                          0,
                          0,
                          0,
                          NULL,
                          NULL);
```

OCSF uses module handles to match a calling application with the appropriate service module. Handles represent a one-to-one pairing between an application and a module. Multiple calls to CSSM_ModuleAttach are viewed as independent requests. Each attach request returns separate, independent handles that do not share execution state.

GenerateContextAndEncrypt:

Located in file: encrypt.c

GenerateContextAndEncrypt performs several operations. It generates a symmetric key for use in encrypting the input file, and also generates a context for use in the encryption process. Finally, the input file is encrypted and the encrypted file is written to a separate file. These operations are performed in these subroutines:

- GenerateKey
- GenerateSymmetricContext
- WriteOutputFile.

GenerateKey:

GenerateKey function creates a symmetric key. It does this by creating a security context, generating a symmetric key using information in the context, and destroying the context. Security contexts perform two functions: to provide security for user-specific information and to package information for easy exchange between functions. Rather than declare, pass, and delete multiple parameters, contexts allow this information to be assembled into one temporary data structure. The type of context to be created depends upon the type of operation to be performed. Since the application requires a symmetric key, it must create a key generation context. However, later in the program the execution of a symmetric context will be in order to encrypt this data.

GenerateKey first calls CSSM_CSP_CreateKeyGenContext and passes it the parameters to be used when creating the key and specifies, among other things, a key size of 64 bits and the desired encryption algorithm – DES. This statement is used:

```
hKeyGenContext = CSSM_CSP_CreateKeyGenContext(hCSP,
                                 CSSM_ALGID_DES,
                                 NULL,
                                 64,
                                 NULL,NULL,NULL,NULL,NULL);
```

GenerateKey next initializes the Key data structure, of type CSSM_KEY, to zero using this statement:

```
memset (Key, 0, sizeof(CSSM_KEY));
```

By setting the Key.KeyData.Data and Key.KeyData.Length fields to zero, the user requests OCSF to allocate the memory necessary to represent the key when CSSM_GenerateKey is called using this statement:

```
CSSM_GenerateKey(hKeyGenContext, CSSM_KEYUSE_ENCRYPT | CSSM_KEYUSE_DECRYPT,
            CSSM_KEYATTR_MODIFIABLE, NULL, Key)
```

CSSM_GenerateKey generates the key and updates the Key data structure accordingly. Once the key has been generated, it is up to the application to delete the security context now that it is no longer needed. It does this by calling CSSM_DeleteContext using this statement:

```
CSSM_DeleteContext(hKeyGenContext)
```

GenerateSymmetricContext:

The GenerateSymmetricContext function creates and returns a cryptographic context handle by calling CSSM_CSP_CreateSymmetricContext. The resulting context is used for the file encryption operations that use a symmetric key. The function parameters specify the CSP module handle, the desired algorithm ID (DES) and algorithm mode (cipher block chain mode), the key data, an initialization vector for the encryption, the type of padding (none), and the number of encryption rounds, in this case 0. This statement is used.

```
*hCryptoContext = CSSM_CSP_CreateSymmetricContext(hCSP,
              CSSM_ALGID_DES,
              CSSM_ALGMODE_CBCPadIV8,
              Key,
              &DESIVData,
              CSSM_PADDING_NONE,
              0);
```

Note that if the encryption were being performed using an asymmetric key, the application would call CSSM_CSP_CreateAsymmetricContext instead.

WriteOutputFile:

This function is called to write the encrypted file. The actual file encryption is performed in GenerateContextAndEncrypt using the CSSM_EncryptData function. This statement is used:

```
CSSM_EncryptData(hCryptoContext,
                 &ClearData,      /*pointer to the input buffer*/
                 1,               /*number of input buffers*/
                 &EncryptedData,  /*pointer to output buffer*/
                 1,               /*number of output buffers*/
                 &BytesEncrypted, /*size of the encrypted data*/
                 &RemData);       /*buffer for padding encrypted data*/
```

File_encrypt Source Code

"File_encrypt Source Code" contains the source code for the file_encrypt program. The program consists of these files:

file_encrypt.h
This files contains the prototypes of public functions.

main.c
This file is the main program and command line parser.

initialize.c
This file shows how to initialize the OCSF for use.

attach.c
This file attaches to one service provider module, a key recovery module, and a Cryptographic module.

encrypt.c
This file performs actual encryption. It generates one output file containing the encrypted data.

makefile.os390
This file contains directives used by the program /bin/make for building applications. &tab;To build the file_encrypt application type in '/bin/make -f makefile.os390'. This will compile all of the C programs to object format and link-edit them in the directory &tab;where you have created all of the code. You must have write access to this directory and your system programmer must have installed the OCSF code.

FILE_ENCRYPT.H

```
//------------------------------------------------------------------------
//
// COMPONENT_NAME: file_encrypt
//
// (C) COPYRIGHT International Business Machines Corp. 1999
// All Rights Reserved
// Licensed Materials - Property of IBM
//
//------------------------------------------------------------------------
//
// FILE: file_encrypt.h
//
// This file contains functions to take a clear file and produce its
// associated encrypted file.  Although
// the symmetric encryption algorithm being used here is DES, others
// could be easily substituted with minimal change.
//
//------------------------------------------------------------------------

void Initialize(
    void);

void AttachCSPByAlgorithm(
    CSSM_CSP_HANDLE *hCSP,
    uint32 AlgorithmRequired);

void GenerateContextAndEncrypt(
    CSSM_CSP_HANDLE hCSP,
      char *InputFilename);

extern CSSM_API_MEMORY_FUNCS MemoryFuncs;
```

MAIN.C

```
//--------------------------------------------------------------------------
//
// COMPONENT_NAME: file_encrypt
//
// (C) COPYRIGHT International Business Machines Corp. 1999
// All Rights Reserved
// Licensed Materials - Property of IBM
//
//--------------------------------------------------------------------------
//
// FILE: main.c
//
// This file contains the main program of the file_encrypt program.
// The command line arguments are processed here and other functions
// are called to perform subtasks such as initializing the CSSM,
// attaching the required service providers.
//
//--------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>

#include "cssm.h"
#include "file_encrypt.h"

//--------------------------------------------------------------------------
//
// Function: ProcessArguments
//
// This function checks the command line arguments and provides syntax.
//
//--------------------------------------------------------------------------
static void ProcessArguments(int argc, char *argv[], char **ClearFilename)
{
    // Check the number of arguments
    if (argc != 2) {
        printf("\n");
        printf("Usage: file_encrypt <file to encrypt>\n");
        printf("\n");
        printf("  This utility encrypts the given file and \n");
        printf("  the and creates the encrypted file.  This is the file\n");
        printf("  generated:\n");
        printf("\n");
        printf("    <filename>.enc - the encrypted file\n");
        printf("\n");
        exit(1);
    }

    // Get the name of the clear file
    *ClearFilename = argv[1];
}

//--------------------------------------------------------------------------
//
// Function: main
//
//--------------------------------------------------------------------------
int main(int argc, char *argv[])
{
    // Handle to the cryptographic service provider
    CSSM_CSP_HANDLE             hCSP;
    char                        *ClearFilename;

    ProcessArguments(argc, argv, &ClearFilename);

    Initialize();

    // Set up cryptographic service provider
    AttachCSPByAlgorithm(&hCSP, CSSM_ALGID_DES);

    // Generate required key recovery fields and then encrypt
    GenerateContextAndEncrypt(hCSP, ClearFilename);

    return 0;
}
```

INITIALIZE.C

```
//------------------------------------------------------------------------
//
// COMPONENT_NAME: file_encrypt
//
// (C) COPYRIGHT International Business Machines Corp. 1999
// All Rights Reserved
// Licensed Materials - Property of IBM
//
//------------------------------------------------------------------------
//
// FILE: initialize.c
//
// This file encapsulates how an application initializes the CSSM.  Memory
// management function tables are passed and versions are checked.
//
//------------------------------------------------------------------------

#include <stdlib.h>
#include <stdio.h>

#include "cssm.h"
#include "file_encrypt.h"

//
// Memory management function table.  See below.
//

CSSM_API_MEMORY_FUNCS   MemoryFuncs;

//
// This set of memory management function wrappers are required by CSSM
// to manage memory on behalf of the calling application.  Note: since the
// calling application is linked separately, it may have its own distinct
// implementation of memory management functions.
//

void *app_malloc(uint32 size, void *ref)           { return(malloc(size));     }
void app_free(void * ptr, void *ref)               { free(ptr);                }
void *app_calloc(uint32 n, uint32 size, void *ref) { return(calloc(n, size));  }
void *app_realloc(void *p, uint32 size, void *ref) { return(realloc(p, size)); }

//------------------------------------------------------------------------
//
// Function: Initialize
//
// This function sets up memory management functions and calls CSSM_Init.
//
//------------------------------------------------------------------------
void Initialize(void)
{
    CSSM_ERROR_PTR          pError;
    // This is the version of the CSSM itself.
    CSSM_VERSION            Version = { CSSM_MAJOR, CSSM_MINOR };

    //
    // Initialize the application's memory management function table
    //

    MemoryFuncs.malloc_func     = app_malloc;
    MemoryFuncs.free_func       = app_free;
    MemoryFuncs.realloc_func    = app_realloc;
    MemoryFuncs.calloc_func     = app_calloc;

    //
    // The CSSM_Init function must be called before performing any other
    // CSSM API calls.  The expected CSSM major/minor version numbers
    // and the memory management function table are passed down.
    //

    if (CSSM_Init(&Version, &MemoryFuncs, NULL) != CSSM_OK)
    {
        printf("Error: could not intialize CSSM\n");
        pError = CSSM_GetError();
        printf("CSSM_Init error code = %d\n", pError->error);
        exit(1);
    }
}
```

ATTACH.C

```
//--------------------------------------------------------------------------
//
// COMPONENT_NAME: file_encrypt
//
// (C) COPYRIGHT International Business Machines Corp. 1999
// All Rights Reserved
// Licensed Materials - Property of IBM
//
//--------------------------------------------------------------------------
//
// FILE: attach.c
//
// There are various levels of detail that applications can use when
// attaching to modules using the CSSM API.  In the simplest case, an
// application can hardcode a particular GUID so that it only works when
// a particular module is installed.  On the other hand, a more flexible
// application can be designed to look into the installed list of modules
// and choose one based on some attribute it has (capability, vendor
// name, hardware/software, etc.).
//
// This file shows two methods (among many) that can be used to attach a
// module.  In AttachCSPByAlgorithm(), the installed list of software
// cryptographic service providers is searched to find one that supports
// the required algorithm.
//
//--------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>

#include <cssm.h>
#include <file_encrypt.h>

//--------------------------------------------------------------------------
//
// Function: AttachCSPByAlgorithm
//
// This function searches the list of all installed modules for a
// CSP that supports the required algorithm.
//
//--------------------------------------------------------------------------
void AttachCSPByAlgorithm(
    CSSM_CSP_HANDLE *hCSP,
    uint32 AlgorithmRequired)
{
    CSSM_ERROR_PTR          pError;         // error information
    CSSM_LIST_PTR           pModuleList;    // list of modules
    CSSM_MODULE_INFO_PTR    pModuleInfo;    // module info
    CSSM_CSPSUBSERVICE_PTR  pCspInfo;       // CSP module info
    CSSM_SOFTWARE_CSPSUBSERVICE_INFO_PTR pInfo;  // software CSP module info
    CSSM_CSP_CAPABILITY_PTR pCap;           // capabilities list
    uint32                  Total;          // miscellaneous
    CSSM_BOOL               Found;          // boolean for search
    uint32                  i;              // index
    uint32                  j;              // index
    uint32                  k;              // index
    uint32                  l;              // index

    //
    // Retrieve the total list of CSPs installed on the system at this time.
    //

    if ((pModuleList = CSSM_ListModules(CSSM_SERVICE_CSP, CSSM_TRUE)) == NULL)
    {
        pError = CSSM_GetError();
        printf("Error: could not list installed modules\n");
        printf("CSSM_ListModules error code = %d\n", pError->error);
        exit(1);
    }

    if (pModuleList->NumberItems == 0)
    {
        printf("Error: no CSPs installed.\n");
        exit(1);
    }

    //
    // Search through installed software CSPs for one that supports the
```

```
    // encryption algorithm required
    //

    Found = CSSM_FALSE;

    for (i = 0; !Found && i < (int)pModuleList->NumberItems; i++)
    {
        pModuleInfo = CSSM_GetModuleInfo(&(pModuleList->Items[i].GUID),
                                         CSSM_SERVICE_CSP,
                                         0,
                                         CSSM_INFO_LEVEL_ALL_ATTR);

        for (j = 0; !Found && j < (int) pModuleInfo->NumberOfServices; j++)
        {
#ifdef OS390
     pCspInfo = pModuleInfo>ServiceList[j].SubserviceList.CspSubServiceList;
#else
          pCspInfo = pModuleInfo->ServiceList[j].CspSubServiceList;
#endif

  for (k = 0; !Found && k < pModuleInfo->ServiceList[j].NumberOfSubServices; k++)
           {
               //
               // Note: to extend the search to hardware CSPs, a case
               // could be added to this switch construct.
               //
               switch (pCspInfo->CspType)
               {
                   case CSSM_CSP_SOFTWARE:
#ifdef OS390
                  pInfo = &(pCspInfo->SubServiceInfo.SoftwareCspSubService);
#else
                       pInfo = &(pCspInfo->SoftwareCspSubService);
#endif
                       Total = pInfo->NumberOfCapabilities;
                       for (l = 0; l < Total; l++)
                       {
                           pCap = &(pInfo->CapabilityList[l]);
                           if (pCap->AlgorithmType == AlgorithmRequired)
                           {
                               Found = CSSM_TRUE;
                           }
                       }
                   break;

                   default:
                   break;
               } // switch
           } // for each subservice
        } // for each usage type
    } // for each module

    if (!Found)
    {
        //
        // There were CSPs, but none of them matched
        //
        printf("Error: there are no suitable cryptographic service providers installed\n");
        exit(1);
    }
    else
    {
        *hCSP = CSSM_ModuleAttach(&(pModuleList->Items[i-1].GUID),
                                  &pModuleInfo->Version,
                                  &MemoryFuncs,
                                  0,
                                  0,
                                  0,
                                  NULL,
                                  NULL);
        if (*hCSP == 0)
        {
            pError = CSSM_GetError();
            printf("Error: could not attach to suitable cryptographic service provider\n");
            printf("CSSM_ModuleAttach error code = %d\n", pError->error);
            exit(1);
        }
```

```
    }

    // Successfully attached to desired CSP
}
```

ENCRYPT.C

```
//--------------------------------------------------------------------------
//
// COMPONENT_NAME: file_encrypt
//
// (C) COPYRIGHT International Business Machines Corp. 1999
// All Rights Reserved
// Licensed Materials - Property of IBM
//
//--------------------------------------------------------------------------
//
// FILE: encrypt.c
//
// This file contains functions to take a clear file and produce its
// associated encrypted file. Although
// the symmetric encryption algorithm being used here is DES, others
// could be easily substituted with minimal change.
//
//--------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>

#include "cssm.h"
#include "file_encrypt.h"

//
// Suffixes used for the name of the generated file
//

#define ENCRYPTED_FILE_SUFFIX   ".enc"

//
// File maximums
//
#define MAX_CLEAR_FILE_SIZE     4096        // for simplification
#define PATH_MAX                 256        // for simplification
//
// DES algorithm parameters
//

#define DES_PAD_LEN     8
#define DES_IV_LEN      8

static unsigned char
DESIVBuffer[DES_IV_LEN] = { 0x03, 0xC4, 0x98, 0x1E, 0x71, 0xFF, 0xA2, 0x23 };

static CSSM_DATA
DESIVData = { sizeof DESIVBuffer, DESIVBuffer };


//--------------------------------------------------------------------------
//
// Function: GenerateKey
//
// This function generates a key using the given CSP.
//
//--------------------------------------------------------------------------
static void GenerateKey(
    CSSM_CSP_HANDLE hCSP,
    CSSM_KEY_PTR Key)
{
    CSSM_CC_HANDLE  hKeyGenContext;     // key generation context
    CSSM_ERROR_PTR  pError;             // error info

    //
    // Create a key generation context which basically packages all
    // parameters into a "handle" for later reference
    //

    hKeyGenContext =
        CSSM_CSP_CreateKeyGenContext(hCSP,
                                     CSSM_ALGID_DES,
                                     NULL,
                                     64,
                                     NULL,NULL,NULL,NULL,NULL);
```

```
    if (hKeyGenContext == 0)
    {
        printf("Error: could not perform key generation setup.\n");
        pError = CSSM_GetError();
        printf("CSSM_CSP_CreateKeyGenContext error code = %d\n", pError->error);
        exit(1);
    }

    //
    // Generate a key
    //

    memset(Key, 0, sizeof(CSSM_KEY));

    if (CSSM_GenerateKey(hKeyGenContext, CSSM_KEYUSE_ENCRYPT | CSSM_KEYUSE_DECRYPT,
            CSSM_KEYATTR_MODIFIABLE, NULL, Key) != CSSM_OK)
    {
        printf("Error: could not generate a key.\n");
        pError = CSSM_GetError();
        printf("CSSM_CSP_GenerateKey error code = %d\n", pError->error);
        exit(1);
    }

    //
    // Delete the unneeded key generation context
    //

    if (CSSM_DeleteContext(hKeyGenContext) != CSSM_OK)
    {
        printf("Error: could not delete key generation context\n");
        pError = CSSM_GetError();
        printf("CSSM_DeleteContext error code = %d\n", pError->error);
        exit(1);
    }
}
//---------------------------------------------------------------------------
//
// Function: GenerateSymmetricContext
//
// This function sets the encryption algorithm parameters including the key
// itself, the algorithm mode, etc.
//---------------------------------------------------------------------------
static void GenerateSymmetricContext(
    CSSM_CSP_HANDLE hCSP,
    CSSM_KEY *Key,
    CSSM_CC_HANDLE *hCryptoContext)
{
    CSSM_ERROR_PTR  pError;             // error info

    //
    // Create a symmetric encryption context to package encryption parameters
    //

    *hCryptoContext =
        CSSM_CSP_CreateSymmetricContext(hCSP,
                                        CSSM_ALGID_DES,
                                        CSSM_ALGMODE_CBCPadIV8,
                                        Key,
                                        &DESIVData,
                                        CSSM_PADDING_NONE,
                                        0);

    if (hCryptoContext == 0)
    {
        printf("Error: could not perform symmetric encryption setup\n");
        pError = CSSM_GetError();
        printf("CSSM_CSP_CreateSymmetricContext error code = %d\n", pError->error);
        exit(1);
    }
}


//---------------------------------------------------------------------------
//
// Function: WriteOutputFile
//
// This function takes a data buffer represented by a CSSM_DATA type and
// writes it out to new file.  The new file's name is composed of the base
// and suffix strings provided.  This function is used to write out the
// encrypted data.
```

```
//
//-------------------------------------------------------------------------
static void WriteOutputFile(
    CSSM_DATA DataToWrite,
    char *FilenameBase,
    char *FilenameSuffix)
{
    char    OutputFilename[PATH_MAX];
    FILE    *OutputFile;
    int     BytesLeft;
    char    *LastByte;
    int     CurrentWritten;
    int     CurrentSize;
    char    *pCurrent;

    //
    // Compose name and open output file
    //

    strcpy(OutputFilename, FilenameBase);
    strcat(OutputFilename, FilenameSuffix);

    if ((OutputFile = fopen(OutputFilename, "wb")) == NULL)
    {
        printf("Error: could not open %s\n", OutputFilename);
        perror("fopen");
        exit(1);
    }

    //
    // Write data
    //

    LastByte    = DataToWrite.Data + DataToWrite.Length - 1;
    BytesLeft   = DataToWrite.Length;

    pCurrent = DataToWrite.Data;

    while (BytesLeft > 0)
    {
        if (pCurrent + BUFSIZ > LastByte)
            CurrentSize = LastByte - pCurrent;
        else
            CurrentSize = BUFSIZ;

        CurrentWritten = fwrite(pCurrent, 1, CurrentSize, OutputFile);

        if (ferror(OutputFile))
        {
            printf("Error: failed to write to file %s\n", OutputFilename);
            perror("fwrite");
            exit(1);
        }

        BytesLeft -= CurrentWritten;
    }

    fclose(OutputFile);

}

//-------------------------------------------------------------------------
//
// Function: GenerateContextAndEncrypt
//
// This function encrypts a file using strong encryption.  It performs all
// the necessary prerequisites such as generation of a key (could be
// replaced by string to key derivation) for the encryption.
//
//-------------------------------------------------------------------------
void GenerateContextAndEncrypt(
    CSSM_CSP_HANDLE hCSP,
    char *InputFilename)
{
    FILE                *ClearFile;         // clear file's handle
    CSSM_CC_HANDLE      hCryptoContext;     // context handle for encryption
    CSSM_KEY            Key;                // the symmetric key for encryption
    int                 BytesRead;          // byte reading counter
    uint32              BytesEncrypted;     // byte encrypting counter
    unsigned char     ClearBuf[MAX_CLEAR_FILE_SIZE]; // buffer for cleartext
```

```
    CSSM_DATA           ClearData;          // buffer for cleartext
    CSSM_DATA          EncryptedData;          // buffer for ciphertext
    unsigned char       RemBuf[DES_PAD_LEN];// buffer for padding
    CSSM_DATA           RemData;            // buffer for padding
    CSSM_RETURN         RC;                 // return code

    //
    // Normally one would prompt the user for a string and convert it to
    // a clear key, but here is an example of the key generation APIs
    //

    GenerateKey(hCSP, &Key);

    GenerateSymmetricContext(hCSP, &Key, &hCryptoContext);

    //
    // Read the clear file in one buffer for simplification
    //

    if ((ClearFile = fopen(InputFilename, "rb")) == NULL)
    {
        printf("Error: could not open %s\n", InputFilename);
        perror("fopen");
        exit(1);
    }

    BytesRead = fread(ClearBuf, 1, MAX_CLEAR_FILE_SIZE, ClearFile);
    ClearData.Length = BytesRead;
    ClearData.Data = ClearBuf;

    if (BytesRead == 0)
    {
        printf("Error: did not read any bytes from file\n");
        exit(1);
    }

    if (!feof(ClearFile))
    {
       printf("Error: exceeded currently supported maximum clear file size\n");
        exit(1);
    }

    fclose(ClearFile);

    //
    // Encrypt the buffer
    //

    // Initialize the buffer that will hold the final block of the encryption
    memset(RemBuf, 0, sizeof(RemBuf));
    RemData.Length  = sizeof(RemBuf);
    RemData.Data    = RemBuf;

    // setup CipherBuf with the same length as ClearBuf
    EncryptedData.Data = (uint8 *) malloc (ClearData.Length);
    EncryptedData.Length = ClearData.Length;

    RC = CSSM_EncryptData(hCryptoContext,
                          &ClearData,
                          1,
                          &EncryptedData,
                          1,
                          &BytesEncrypted,
                          &RemData);

    // Move the final block of data to the end of the EncryptedBuf
    memcpy(EncryptedData.Data + BytesEncrypted, RemData.Data, RemData.Length);
    EncryptedData.Length =BytesEncrypted + RemData.Length;

    //
    // Write the encrypted file
    //

    WriteOutputFile(EncryptedData, InputFilename, ENCRYPTED_FILE_SUFFIX);
}
```

MAKEFILE.OS390

```
#*****************************************************************/
#*                                                               */
#* This file contains sample code. IBM PROVIDES THIS CODE ON AN  */
#* 'AS IS' BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR */
#* IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES */
#* OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.       */
#*                                                               */
#*****************************************************************/

INSTALL_DIR = /usr/lpp/ocsf
LIB_DIR     = $(INSTALL_DIR)/lib

CFLAGS = -c -I$(INSTALL_DIR)/inc  \
         -I. -DOS390              \
         -I/proj/cdsa/sboxes/theZIN/sb_base/7b/src/inc \
         -Wc,dll,sscom

LFLAGS = -I$(INSTALL_DIR)/inc  \
         -I.                   \
         -I/proj/cdsa/sboxes/theZIN/sb_base/7b/src/inc \
         -Wc,dll,sscom
CC = /bin/c89

all: file_encrypt

file_encrypt: encrypt.o attach.o initialize.o main.o
     $(CC) $(LFLAGS) -o file_encrypt encrypt.o attach.o initialize.o main.o
$(LIB_DIR)/cssm32.x

encrypt.o: encrypt.c file_encrypt.h
     $(CC) $(CFLAGS) encrypt.c

attach.o: attach.c file_encrypt.h
     $(CC) $(CFLAGS) attach.c

initialize.o: initialize.c file_encrypt.h
     $(CC) $(CFLAGS) initialize.c

main.o: main.c file_encrypt.h
     $(CC) $(CFLAGS) main.c

clean:
     rm -f *.o
     rm -f encrypt
```

Chapter 10. Core Services API

The OCSF provides this set of services:
- Module Management
- Memory Management Support
- Security Context Management
- Integrity Verification Services

These Application Programming Interfaces (APIs) are implemented by the OCSF, not by service provider modules. For information on the service provider modules, refer to the *z/OS Open Cryptographic Services Facility Service Provider Module Developer's Guide and Reference*.

Module Management Services

The OCSF module management functions support module installation, dynamic selection and loading of modules, and querying of module features and status. System administration utilities use OCSF install and uninstall functions to maintain service provider modules on a local system.

Applications select the particular security services they will use by selectively attaching service provider modules. These modules are provided by Independent Software Vendors (ISVs). Each module has an assigned Globally Unique ID (GUID) and a set of descriptive attributes to assist applications in selecting appropriate modules for their use. A module can implement a range of services across the OCSF APIs (e.g., cryptographic functions, data storage functions) or a module can restrict its services to a single OCSF category of service (e.g., Certificate Library (CL) services only). Modules that span service categories are called multiservice modules.

Applications use a module's GUID to specify the module to be attached. The CSSM_ModuleAttach function returns a handle representing a unique pairing between the caller and the attached module. Applications must provide this handle as an input parameter when requesting services from the attached module. OCSF uses the handle to match the caller with the appropriate service module.

The calling application uses the handle to obtain all types of services implemented by the attached module. Figure 5 on page 84 shows how the handle for an attached Dual_Provider service provider is used to perform cryptographic operations and persistent storage of certificates. The single handle value can be used as the CSPHandle in cryptographic operations and as the DLHandle in data storage operations.

Multiple calls to attach are viewed as independent requests. Each attach request returns separate, independent handles that do not share execution state.

Before attaching a service module, an application can query the OCSF registry to obtain information on:
- Modules installed on the system
- Capabilities (and functions) implemented by those modules
- GUID associated with a given module.

Applications use this information to select a module for use. A multiservice module has multiple capability descriptions associated with it. Some areas, such as Cryptographic Service Provider (CSP) and Trust Policy (TP), may have multiple independent capability descriptions for a single functional area. There is one OCSF registry entry for a multiservice module. That entry records all service types for the module. OCSF returns all information about a module's capabilities when queried by the application.

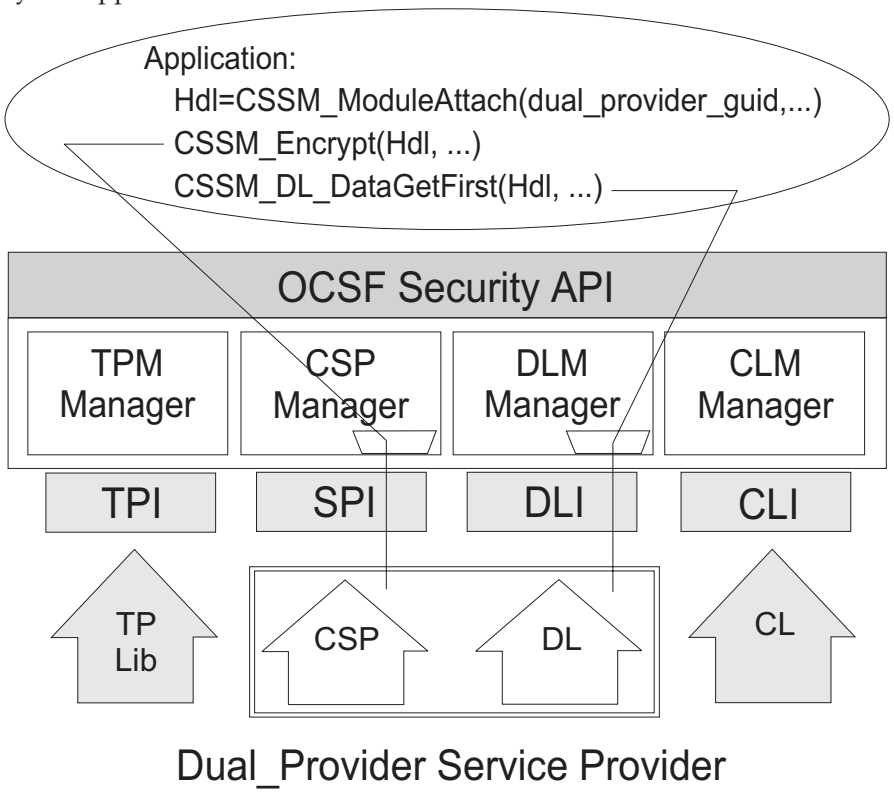

Figure 5. Dual_Provider Cryptographic Services and Persistent Storage Services

Applications can query about OCSF themselves. OCSF provides several functions to assist applications in ensuring that the current OCSF version meets the application's needs. CSSM_GetInfo returns version information about OCSF. CSSM_Init verifies whether the application's expected OCSF version is compatible with the currently running OCSF version. (The general function to query service provider module information also returns the module's version information.)

Memory Management Support

The OCSF memory management functions are a class of routines for reclaiming memory allocated by OCSF on behalf of an application from the OCSF memory heap. When OCSF allocates objects from its own heap and returns them to an application, the application must inform OCSF when it no longer requires the use of that object. Applications use specific APIs to free OCSF-allocated memory. When an application invokes a free function, OCSF can choose to retain or free the indicated object depending on other conditions known only to OCSF. In this way, OCSF and applications work together to manage these objects in the OCSF memory heap.

Security Context Management

The OCSF framework is responsible for maintaining data that may be required to perform cryptographic and security operations. The internal context structure maintains information pertaining to the parameters of the cryptographic operation, such as the type of algorithm to be performed, and maintains a list of attributes to customize the information stored in the context. These attributes can be of different types, including keys, dates, and raw data buffers. When the application creates a context, it supplies a set of parameters based on what type of context it is, and the framework returns a handle to that context. The application can then use that handle to add additional attributes to the framework, and update the contents of the existing attributes. The context handle is passed to the functions that perform the actual cryptographic operations. The data and attributes are retrieved from the context management system for use by the addin performing the operations. When the application is done with a context, it should pass the handle to the CSSM_DeleteContext function in order to free up the memory used by that context.

Integrity Verification Services

As a security framework, OCSF provides each application with checking of the integrity of the OCSF environment in which the application is running. OCSF requires all code including OCSF binaries and the invoking application to be program controlled. Non-program controlled binaries causes the environment to become "dirty" and the result will be failure of attaching OCSF Service Providers. In addition, Cryptographic Service Providers and Policy Modules have additional checks to verify their validity.

Data Structures for Core Services

"Data Structures for Core Services" discusses the data structures for the core services.

Note: Some application interfaces use data structures defined by other OCSF services. Those data structures are defined with those particular OCSF services.

Basic Data Types

```
typedef unsigned char uint8;
typedef unsigned short uint16;
typedef short sint16;
typedef unsigned int uint32;
typedef int sint32;

#define CSSM_MODULE_STRING_SIZE  64
typedef char CSSM_STRING [CSSM_MODULE_STRING_SIZE + 4];
```

CSSM_ALL_SUBSERVICES

This data type is used to identify that information on all of the subservices is being requested or returned.

```
#define  CSSM_ALL_SUBSERVICES     (-1)
```

CSSM_API_MEMORY_FUNCS_PTR

This is data structure is used by an application to pass in its own memory management routines ot OCSF. It is defined by this set of declarations:

```
/* structure for passing a memory function table to cssm */
typedef struct cssm_memory_funcs {
    void *(*malloc_func) (uint32 Size, void *AllocRef);
    void (*free_func) (void *MemPtr, void *AllocRef);
```

```
    void *(*realloc_func) (void *MemPtr, uint32 Size, void *AllocRef);
    void *(*calloc_func) (uint32 Num, uint32 Size, void *AllocRef);
    void *AllocRef;
} CSSM_MEMORY_FUNCS, *CSSM_MEMORY_FUNCS_PTR;

typedef CSSM_MEMORY_FUNCS CSSM_API_MEMORY_FUNCS;
typedef CSSM_API_MEMORY_FUNCS *CSSM_API_MEMORY_FUNCS_PTR;
```

CSSM_BOOL

This data type is used to indicate a true or false condition.

```
typedef uint32 CSSM_BOOL;

#define CSSM_TRUE 1
#define CSSM_FALSE 0
```

Definitions:

CSSM_TRUE
: Indicates a true result or a true value.

CSSM_FALSE
: Indicates a false result or a false value.

CSSM_COUNTRY_ORIGIN

```
typedef enum cssm_country_origin {
    CSSM_COUNTRY_US = 1,
    CSSM_COUNTRY_NONUS = 2
} CSSM_COUNTRY_ORIGIN;
```

CSSM_CRYPTO_TYPE

```
typedef enum cssm_crypto_type {
    CSP_TYPE_NONE = 0,
    CSP_TYPE_EXPORT = 1,
    CSP_TYPE_SSL = 2,
    CSP_TYPE_FINANCIAL = 3,
    CSP_TYPE_EXPORTVERIFY = 4,
    CSP_TYPE_AUTHENTICATE = 5
} CSSM_CRYPTO_TYPE;
```

CSSM_CSP_MANIFEST

```
typedef struct cssm_csp_manifest {
    char *Vendor;
    CSSM_COUNTRY_ORIGIN CountryOrigin;
    CSSM_CRYPTO_TYPE CryptoType;
    uint32 NumberCapabilities;
    CSSM_CSP_CAPABILITY_PTR Capabilities;
} CSSM_CSP_MANIFEST, * CSSM_CSP_MANIFEST_PTR;
```

CSSM_CSSMINFO

This data structure represents the information associated with an installation of OCSF.

```
typedef struct cssm_cssminfo {
    CSSM_VERSION Version;
    char *Description
    char *Vendor
    CSSM_BOOL ThreadSafe;
    char *Location;
    CSSM_GUID GUID;
}CSSM_CSSMINFO, *CSSM_CSSMINFO_PTR
```

CSSM_DATA

The CSSM_DATA structure is used to associate a length, in bytes, with an arbitrary block of contiguous memory. This memory must be allocated and freed using the memory management routines provided by the calling application through OCSF.

TP modules and CLs use this structure to hold certificates and Certificate Revocation Lists (CRLs). Other service modules, such as CSPs, use this same structure to hold general data buffers. DL modules use this structure to hold persistent security-related objects.

```
typedef struct cssm_data{
    uint32 Length;/* in bytes */
    uint8 *Data;
} CSSM_DATA, *CSSM_DATA_PTR
```

Definitions:

Length Length of the data buffer in bytes.

Data Points to the start of an arbitrary length data buffer.

CSSM_EVENT_TYPE

```
typedef uint32 CSSM_EVENT_TYPE, *CSSM_EVENT_TYPE_PTR;

#define CSSM_EVENT_ATTACH                  (0)
#define CSSM_EVENT_DETACH                  (1)
#define CSSM_EVENT_INFOATTACH          &tab;(2)
#define CSSM_EVENT_INFODETACH          &tab;(3)
#define CSSM_EVENT_CREATE_CONTEXT   (4)
#define CSSM_EVENT_DELETE_CONTEXT  (5)
```

CSSM_GUID

This structure designates a Globally Unique ID (GUID) that distinguishes one service provider module from another. All GUID values should be computer-generated to guarantee uniqueness. (The GUID generator in Microsoft Developer Studio, the RPC UUIDGEN/uuid_gen program can be used on a number of UNIX-based platforms, and the UUIDEN of the DCE on z/OS can be used to generate a GUID.)

```
typedef struct cssm_guid{
    uint32 Data1;
    uint16 Data2;
    uint16 Data3;
    uint8Data4[8];
} CSSM_GUID, *CSSM_GUID_PTR
```

Definitions:

Data1 Specifies the first 8 hexadecimal digits of the GUID.

Data2 Specifies the first group of 4 hexadecimal digits of the GUID.

Data3 Specifies the second group of 4 hexadecimal digits of the GUID.

Data4 Specifies an array of 8 elements that contains the third and final group of 8 hexadecimal digits of the GUID in elements 0 and 1, and the final 12 hexadecimal digits of the GUID in elements 2 through 7.

CSSM_HANDLE

This structure is an opaque handle used to refer to data or state retained by one OCSF API call for use by subsequent API calls.

```
typedef uint32 CSSM_HANDLE, *CSSM_HANDLE_PTR;
```

CSSM_INFO_LEVEL

This enumerated list defines the levels of information detail that can be retrieved about the services and capabilities implemented by a particular module. Modules can implement multiple OCSF service types. Modules can also present their services as subservices. Modules can also be dynamic with respect to the services and features they provide.

```
typedef enum cssm_info_level {
    CSSM_INFO_LEVEL_MODULE   = 0,
         /* values from XXinfo struct */
    CSSM_INFO_LEVEL_SUBSERVICE   = 1,
         /* values from XXinfo and XXsubservice struct */
    CSSM_INFO_LEVEL_STATIC_ATTR   = 2,
         /* values from XXinfo and XXsubservice and all
          static-valued attributes of a subservice */
    CSSM_INFO_LEVEL_ALL_ATTR   = 3,
         /* values from XXinfo and XXsubservice and all attributes,
          static and dynamic, of a subservice */
} CSSM_INFO_LEVEL;
```

CSSM_LIST

This structure is used to encapsulate an array of CSSM_LIST_ITEMs, where the array length is given by the Length variable.

```
typedef struct cssm_list{
    uint32 NumberItems;
    CSSM_LIST_ITEM_PTR Items;
} CSSM_LIST, *CSSM_LIST_PTR
```

Definitions:

NumberItems
: The number of items in the list.

Items
: An array of pointers to the item structures.

CSSM_LIST_ITEM

This structure is used to encapsulate the name and GUID of a service provider module.

```
typedef struct cssm_list_item{
    CSSM_GUID GUID;
    char *Name;
} CSSM_LIST_ITEM, *CSSM_LIST_ITEM_PTR
```

Definitions:

GUID
: The global unique identifier of the module.

Name
: The name of the module.

CSSM_MODULE_FLAGS

```
typedef uint32 CSSM_MODULE_FLAGS;

#define CSSM_MODULE_THREADSAFE   0x1
#define CSSM_MODULE_EXPORTABLE&tab;0x2
```

CSSM_MODULE_HANDLE

This structure is a unique identifier for an attached service provider module.

```
typedef uint32 CSSM_MODULE_HANDLE
```

CSSM_MODULE_INFO

This structure aggregates all service descriptions about all service types of a module implementation.

```
typedef struct cssm_module_info {
    CSSM_VERSION Version;              /* Module version */
    CSSM_VERSION CompatibleCSSMVersion;/* Module written for CSSM version */
    CSSM_STRING Description;           /* Module description */
    CSSM_STRING Vendor;                /* Vendor name, etc */
    CSSM_MODULE_FLAGS Flags;   /* Flags to describe and control module use */
    CSSM_SERVICE_MASK ServiceMask;     /* Bit mask of supported services */
    uint32 NumberOfServices;           /* Num of services in ServiceList */
    CSSM_SERVICE_INFO_PTR ServiceList;/* Pointer to list of service infos */
    void* Reserved;
} CSSM_MODULE_INFO, *CSSM_MODULE_INFO_PTR;
```

Definitions:

Version
The major and minor version numbers of this service provider module.

CompatibleCSSMVersion
The version of OCSF that this module was written to.

Description
A text description of this module and its functionality.

Vendor The name and description of the module vendor.

Flags Characteristics of this module, such as whether or not it is threadsafe.

ServiceMask
&tab;A bit-mask identifying the types of services available in this module.

NumberOfServices
The number of services for which information is provided. Multiple descriptions &tab;(as subservices) can be provided for a single service category.

ServiceList
An array of pointers to the service information structures. This array contains NumberOfServices entries.

Reserved
This field is reserved for future use. It should always be set to NULL.

CSSM_NOTIFY_CALLBACK

The CSSM_NOTIFY_CALLBACK is used by the application to provide a function pointer to a callback routine. It is typically supplied in the CSSM_ModuleAttach API when the application developer wishes something to be called in response to a particular event happening. It is defined as follows:

```
typedef CSSM_RETURN (CSSMAPI *CSSM_NOTIFY_CALLBACK) (CSSM_MODULE_HANDLE
                                                     uint32 Application,ModuleHandle,
                                                     uint32 Reason,
                                                     void * Param);
```

Definitions:
ModuleHandle
The handle of the attached service provider module.
Application
Input data to identify the callback.
Reason The reason for the notification.
Param Any additional information about the event

Reason	Description
CSSM_NOTIFY_SURRENDER	The service provider module is temporarily surrendering control of the process.
CSSM_NOTIFY_COMPLETE	An asynchronous operation has completed.
CSSM_NOTIFY_DEVICE_REMOVED	A device, such as a token, has been removed.
CSSM_NOTIFY_DEVICE_INSERTED	A device, such as a token, has been inserted.

CSSM_RETURN

This data type is used to indicate whether a function was successful.

```
typedef enum cssm_return {
            CSSM_OK = 0,
            CSSM_FAIL = -1
} CSSM_RETURN
```

Definitions:

CSSM_OK
: Indicates operation was successful.

CSSM_FAIL
: Indicates operation was unsuccessful.

CSSM_SERVICE_FLAGS

This defines a bit-mask that categorizes the type of service provided by a service provider module. It can contain any combination of CSSM_SERVICE_MASK values.

```
typedef uint32 CSSM_SERVICE_FLAGS;

#define CSSM_SERVICE_ISWRAPPEDPRODUCT 0x1
                        /* On  = Contains one or more embedded products
                           Off = Contains no embedded products */
```

CSSM_SERVICE_INFO

This structure holds a description of a module service. The service described is of the OCSF service type specified by the module type.

```
typedef struct cssm_serviceinfo {
    CSSM_STRING Description; /* Service description */
    CSSM_SERVICE_TYPE Type;  /* Service type */
    CSSM_SERVICE_FLAGS Flags;/* Service flags */

    uint32 NumberOfSubServices;/*Number of sub services in SubServiceList */
    union { /* List of sub services */
        void *SubServiceList;
        CSSM_CSPSUBSERVICE_PTR CspSubServiceList;
        CSSM_DLSUBSERVICE_PTR DlSubServiceList;
        CSSM_CLSUBSERVICE_PTR ClSubServiceList;
        CSSM_TPSUBSERVICE_PTR TpSubServiceList;
#ifdef KEY_RECOVERY
        CSSM_KRSPSUBSERVICE_PTR KrSubServiceList;
#endif
#ifndef_MVS_
    };
#else
    /* Use the CDSA Version 2.0 definition instead of the anonymous union of
       The Version 1.x spec which unforturnately is not ANSI-C compatible. */

    } SubserviceList;
#endif

    void* Reserved;
} CSSM_SERVICE_INFO, *CSSM_SERVICE_INFO_PTR;
```

Definitions:

Description
: A text description of the service.

Type Specifies exactly one type of service structure, such as CSSM_SERVICE_CSP, CSSM_SERVICE_CL, etc.

Flags Characteristics of this service, such as whether it contains any embedded products.

NumberOfSubServices
: The number of elements in the module *SubServiceList*.

SubServiceList
A list of descriptions of the encapsulated subservices (not of the basic service types).

CspSubServiceList
A list of descriptions of the encapsulated CSP subservices.

DlSubServiceList
A list of descriptions of the encapsulated DL subservices.

ClSubServiceList
A list of descriptions of the encapsulated CL subservices.

TpSubServiceList
A list of descriptions of the encapsulated TP subservices.

KrSubServiceList[3]
A list of descriptions of the encapsulated key recovery subservices.

Reserved
This field is reserved for future use. It should always be set to NULL.

Note: _MVS_ is a z/OS compiler definition (by default) therefore on our platform the union will take on the name SubserviceList given in the declaration.

CSSM_SERVICE_MASK

This defines a bit-mask of the possible categories of OCSF services that may be implemented by a single service provider module.

```
typedef uint32 CSSM_SERVICE_MASK;

#define CSSM_SERVICE_CSSM  0x1
#define CSSM_SERVICE_CSP   0x2
#define CSSM_SERVICE_DL    0x4
#define CSSM_SERVICE_CL    0x8
#define CSSM_SERVICE_TP    0x10
#define CSSM_SERVICE_KR    0x20
#define CSSM_SERVICE_LAST  CSSM_SERVICE_TP
```

CSSM_USER_AUTHENTICATION

This structure holds the user's credentials for authenticating to the data storage library module. The type of credentials required is defined by the DL module and specified as a CSSM_USER_AUTHENTICATION_MECHANISM.

```
typedef struct cssm_user_authentication {
    CSSM_DATA_PTR Credential;
    CSSM_CRYPTO_DATA_PTR MoreAuthenticationData;
} CSSM_USER_AUTHENTICATION, *CSSM_USER_AUTHENTICATION_PTR;
```

Definitions:

Credential
A certificate, a shared secret, a token, or whatever the service provider module requires for user authentication. The required credential type is specified as a CSSM_USER_AUTHENTICATION_MECHANISM.

MoreAuthenticationData
A passphrase or other data that can be provided as immediate data within this structure or via a callback function to the user/caller.

3. This is not supported in z/OS.

CSSM_USER_AUTHENTICATION_MECHANISM

This enumerated list defines different methods a service provider module can require when authenticating a caller. The module specifies which mechanism the caller must use for each subservice type provided by the module. For example, the DL modules may require password-based authentication, may require a login sequence, or may accept a certificate and passphrase.

```
typedef enum cssm_user_authentication_mechanism {
    CSSM_AUTHENTICATION_NONE = 0,
    CSSM_AUTHENTICATION_CUSTOM = 1,
    CSSM_AUTHENTICATION_PASSWORD = 2,
    CSSM_AUTHENTICATION_USERID_AND_PASSWORD = 3,
    CSSM_AUTHENTICATION_CERTIFICATE_AND_PASSPHRASE = 4,
    CSSM_AUTHENTICATION_LOGIN_AND_WRAP = 5,
} CSSM_USER_AUTHENTICATION_MECHANISM;
```

CSSM_VERSION

This structure is used to represent the version of OCSF components.

```
typedef struct cssm_version {
    uint32 Major;
    uint32 Minor;
} CSSM_VERSION, *CSSM_VERSION_PTR
```

Definitions:

Major The major version number of the component.
Minor The minor version number of the component.

APIs for Core Services

"APIs for Core Services" describes the Application Programming Interfaces (APIs) for Core Services.

CSSM_FreeInfo

Purpose

This function frees the memory allocated for the CSSM_CSSMINFO structure by the CSSM_GetInfo function.

Format

```
CSSM_RETURN CSSMAPI CSSM_FreeInfo (CSSM_CSSMINFO_PTR CssmInfo)
```

Parameters

Input/Output

CssmInfo
A pointer to the CSSM_CSSMINFO structure to be freed.

Return Value

A CSSM_OK return value signifies the memory has been freed. When CSSM_FAIL is returned, an error occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_GetInfo

CSSM_GetInfo

Purpose

This function returns the version information of the OCSF Framework.

Format

```
CSSM_CSSMINFO_PTR CSSMAPI CSSM_GetInfo (void)
```

Parameters

None.

Return Value

A pointer to a CSSM_CSSMINFO structure. If the pointer is NULL, an error occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_FreeInfo

CSSM_Init

Purpose

This function initializes OCSF and verifies that the version of OCSF expected by the application is compatible with the version of OCSF on the system. This function should be called only once by each application.

Format

```
CSSM_RETURN CSSMAPI CSSM_Init
                          (const CSSM_VERSION_PTR Version,
                           const CSSM_API_MEMORY_FUNCS_PTR MemoryFuncs,
                           const void * Reserved)
```

Parameters

Input

Version
The major and minor version number of the OCSF release the application is compatible with.

MemoryFuncs
Memory functions for OCSF to use when allocating data structures for the application.

Reserved
A reserved input.

Return Value

A CSSM_OK return value signifies the initialization operation was successful. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Module Management Functions

"Module Management Functions" describes the module management functions for core services.

CSSM_FreeModuleInfo

Purpose

This function frees the memory allocated by CSSM_GetModuleInfo to hold the module info structures. All substructures within the information structure are freed by this function.

Format

```
CSSM_RETURN CSSMAPI CSSM_FreeModuleInfo (CSSM_MODULE_INFO_PTR ModuleInfo)
```

Parameters

Input

ModuleInfo
A pointer to the CSSM_MODULE_INFO structure to be freed.

Return Value

This function returns CSSM_OK if successful, and returns CSSM_FAIL if an error has occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_GetModuleInfo
CSSM_SetModuleInfo

CSSM_GetCSSMRegistryPath

Purpose

This function gets the directory path of the OCSF registry.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_GetCSSMRegistryPath (void)
```

Parameters

None

Return Value

A pointer to a CSSM_DATA structure containing the registry path information or a NULL, if an error occurred in getting the information. Use CSSM_GetError to determine the exact error.

CSSM_GetGUIDUsage

Purpose

Returns a bit-mask describing the OCSF function categories of service provided by the module identified by GUID.

Format

```
CSSM_SERVICE_MASK CSSMAPI CSSM_GetGUIDUsage (const CSSM_GUID_PTR ModuleGUID)
```

Parameters

Input

ModuleGUID
Pointer to a Globally Unique Identifier for the module of interest.

Return Value

A CSSM_SERVICE_MASK from the info structure describing the services provided by the module referenced by the GUID.

Related Information

CSSM_GetHandleUsage

CSSM_GetHandleUsage

Purpose

Returns a bit-mask describing the OCSF function categories of service provided by the module, which is identified by the specified handle for an attached module.

Format

```
CSSM_SERVICE_MASK CSSMAPI CSSM_GetHandleUsage (CSSM_HANDLE ModuleHandle)
```

Parameters

Input

ModuleHandle
Handle of the module for which information should be returned.

Return Value

A CSSM_SERVICE_MASK from the info structure describing the services provided by the module referenced by the handle.

Related Information

CSSM_GetGUIDUsage

CSSM_GetModuleGUIDFromHandle

Purpose

This function determines the GUID associated with a specific module handle.

Format

```
CSSM_GUID_PTR CSSMAPI CSSM_GetModuleGUIDFromHandle (CSSM_HANDLE ModuleHandle)
```

Parameters

Input

ModuleHandle
The handle that describes the service provider module.

Return Value

A CSSM_GUID_PTR to a data structure containing the GUID associated with ModuleHandle. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

CSSM_GetModuleInfo

Purpose

This function returns descriptive information about the module identified by the *ModuleGUID*. The information returned can include: all of the capability information, information for each subservice, or information 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 *ServiceMask* bit-mask. The request may be further limited to one or all of the subservices implemented in one or all of the service categories. Finally, the detail level of the information returned can be controlled by the *InfoLevel* input parameter. This is particularly important for a module with dynamic capabilities. *InfoLevel* can be used to request static attribute values only or dynamic values.

Format

```
CSSM_MODULE_INFO_PTR CSSMAPI CSSM_GetModuleInfo
           (const CSSM_GUID_PTR ModuleGUID
           CSSM_SERVICE_MASK ServiceMask
           uint32 SubserviceID
           CSSM_INFO_LEVEL InfoLevel)
```

Parameters

Input

ModuleGUID
A pointer to the CSSM_GUID structure containing the GUID for the service provider module.

ServiceMask
A bit-mask specifying the module service types used to restrict the capabilities information returned by this function. An input value of zero specifies all services for the specified module.

SubserviceID
A single subservice ID or the value CSSM_ALL_SUBSERVICES must be provided. If a subservice ID is provided, the get operation is limited to the specified subservice. Note that a service mask may already limit the operation. If so, the subservice ID applies to all service categories selected by the service mask. If CSSM_ALL_SUBSERVICES is specified, information for all subservices (as limited by the service mask) is returned by this function.

InfoLevel
Indicates the level of detail returned by this function. Information retrieval can be restricted as follows. Note that not all service provider modules support all of these values.

- **CSSM_INFO_LEVEL_MODULE** - Returns only the information contained in the cssm_moduleinfo structure.

- **CSSM_INFO_LEVEL_SUBSERVICE** - Returns the information returned by CSSM_INFO_LEVEL_MODULE and the information contained in the cssm_XXsubservice structure, where XX corresponds to the module type, such as cssm_tpsubservice.
- **CSSM_INFO_LEVEL_STATIC_ATTR** - Returns the information returned by CSSM_INFO_LEVEL_SUBSERVICE and the attribute and capability values that are statically defined for the module.
- **CSSM_INFO_LEVEL_ALL_ATTR** - Returns the information returned by CSSM_INFO_LEVEL_SUBSERVICE and the attribute and capability values that are statically or dynamically defined for the module. Dynamic modules, whose capabilities change over time, support a query function used by OCSF to interrogate the module's current capability status.

Return Value

A CSSM_MODULE_INFO_PTR to an array of one or more info structures. Each structure contains type information identifying the capability description as representing CL capabilities, DL capabilities, etc. The capability descriptions can also be subclassed into subservices.

Related Information

CSSM_FreeModuleInfo

CSSM_GetModuleLocation

Purpose

This function returns the directory path of the service provider module specified by the GUID input parameter.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_GetModuleLocation (const CSSM_GUID_PTR GUID)
```

Parameters

Input

GUID
A pointer to the CSSM_GUID structure containing the GUID for the service provider module.

Return Value

A pointer to a CSSM_DATA data structure containing the directory path of the module associated with GUID. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

CSSM_ListModules

Purpose

This function returns a list containing the GUID/name pair for each of the currently installed service provider modules that provide services in any of the OCSF functional categories selected in the service mask.

Format

```
CSSM_LIST_PTR CSSMAPI CSSM_ListModules (CSSM_SERVICE_MASK ServiceMask, CSSM_BOOL MatchAll)
```

Parameters

Input

ServiceMask
&tab;A bit-mask selecting the OCSF functional categories. This information can be used to select information about potential service provider modules.

MatchAll
A Boolean value defining how the multiple bits in the service mask are interpreted. CSSM_TRUE means the service modules selected must match all service areas specified by the service mask. CSSM_FALSE means the service module selected must specify one or more of the service areas specified by the service mask.

Return Value

A pointer to the CSSM_LIST structure containing the GUID/name pair for each of the modules. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_GetModuleInfo
CSSM_FreeModuleInfo

CSSM_ModuleAttach

Purpose

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 (as described in the service provider's information). The caller can specify a specific subservice provided by the module. Subservice flags may be required to set parameters for the service.

Format

```
CSSM_MODULE_HANDLE CSSMAPI CSSM_ModuleAttach
           (const CSSM_GUID * GUID,
           const CSSM_VERSION_PTR Version,
           const CSSM_API_MEMORY_FUNCS_PTR MemoryFuncs,
           uint32 SubserviceID,
           uint32 SubserviceFlags,
           uint32 Application,
           const CSSM_NOTIFY_CALLBACK Notification,
           const void * Reserved)
```

Parameters

Input

GUID
A pointer to the CSSM_GUID structure containing the GUID for the service provider module.

Version
The major and minor version number of the service provider module with which the application is compatible.

MemoryFuncs
Memory functions for OCSF to use when allocating data structures for the application.

SubserviceID
The number of a subservice provided by the module. This value should always be taken from the CSSM_MODULE_INFO structure to insure that a compatible identifier is used. (Service provider modules that implement only one service can use zero as the subservice identifier.)

SubserviceFlags
Bit-mask of service options defined by a particular subservice of the module. Valid values are described in module-specific information. A default set of flags is specified in the CSSM_MODULE_INFO structure for use by the caller.

Reserved
A reserved input.

Input/optional

Application
This is passed to the application when its callback is invoked allowing the application to determine the proper context of operation.

Notification
Callback provided by the application that is called by the service provider module when one of these occurs: a parallel operation completes, a token running in serial mode surrenders control to the application, or the token is removed (hardware-specific).

Return Value

A module handle for the attached service provider module is returned. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_ModuleDetach

CSSM_ModuleDetach

Purpose

This function detaches the application from the service provider module.

Format

```
CSSM_RETURN CSSMAPI CSSM_ModuleDetach (CSSM_MODULE_HANDLE ModuleHandle)
```

Parameters

Input

ModuleHandle
The handle that describes the service provider module.

Return Value

A CSSM_OK return value signifies that the application has been detached from the service provider module. If CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_ModuleAttach

Utility Functions

"Utility Functions" describes the utility functions for the core services.

CSSM_FreeList

Purpose

This function frees the memory allocated to hold a list of strings.

Format

```
CSSM_RETURN CSSMAPI CSSM_FreeList (CSSM_LIST_PTR List)
```

Parameters

Input

List
&tab;A pointer to the CSSM_LIST structure containing the GUID/name pair of service provider modules.

Return Value

A CSSM_OK return value signifies that the allocated memory has been freed. If CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

CSSM_GetAPIMemoryFunctions

Purpose

This function retrieves the memory function table associated with the service provider module.

Format

```
CSSM_API_MEMORY_FUNCS_PTR CSSMAPI CSSM_GetAPIMemoryFunctions(CSSM_HANDLE AddInHandle)
```

Parameters

Input

AddInHandle
The handle to the service provider module whose memory function table is being requested.

Return Value

A pointer to the CSSM_API_MEMORY_FUNCS table associated with the service provider module. If an error condition occurred, the function returns NULL. Use CSSM_GetError to obtain the error code.

Chapter 11. OCSF Privilege Mechanism

The OCSF does not support the Privilege Mechanism as defined by the Keyworks derivative implementation. The OCSF does provide Privilege Mechanism toleration support as follows:

- All Privilege APIs can be invoked. There are no privileges tied to the application. For compatability purposes, where required by the Privilege API, an application pathname and application file name still must be specified, but will not be checked. The privileges returned will be based on the policy modules on the system.
- When the OCSF Security Level 3 feature is installed, all privileges are returned.
- When only the OCSF base is installed, no privileges are returned.

Data Structures

"Data Structures"describes the data structure for the OCSF Privilege Mechanism.

Note: Some application interfaces use data structures defined by other OCSF services. Those data structures are defined with those particular OCSF services.

CSSM_EXEMPTION_MASK

This data type defines a bit-mask of exemptions or privileges pertaining to the OCSF framework. Exemptions are defined to correspond to built-in checks performed by OCSF framework and the module managers. The caller must possess the necessary credentials to be granted the exemptions. At this time, the CSSM_EXEMPTION_MASK can hold a maximum of 32 distinct privileges. The mask data type may be changed in the future to allow expansion to support larger sets of privileges.

```
typedef uint32 CSSM_EXEMPTION_MASK;

#define CSSM_EXEMPT_NONE                          0x00  &tab;/* no privileges*/

#define CSSM_EXEMPT_MULTI_ENCRYPT_CHECK&tab;           0x01  /* privilege that allows the */
                                                    /* caller to perform repeated nested */
                                                    /* encryption of a data buffer */

#define CSSM_STRONG_CRYPTO_WITH_KR                0x02  /* privilege that allows the caller*/
                                                    /* to obtain any strength cryptography as*/
                                                    /* long as key recovery operations are*/
                                                    /* performed based on key recovery policy*/
                                                    /* tables */

#define CSSM_EXEMPT_LE_KR                         0x04&tab; /* privilege that allows the caller*/
                                                    /* to obtain any strength cryptography*/
                                                    /* without the need to perform law*/
                                                    /* enforcement key recovery operations.*/

#define CSSM_EXEMPT_ENT_KR                        0x08 /* privilege that allows the caller*/
                                                    /* to obtain any strength cryptography*/
                                                    /* without the need to perform enterprise*/
                                                    /* key recovery operations.*/

#define CSSM_EXEMPT_ALL                           0xff /* privilege that allows the caller*/
                                                    /* to obtain the services corresponding to*/
                                                    /* the combination of all the privileges*/
                                                    /* defined.*/
```

Operations

This describes the operations APIs for the OSCF Privilege Mechanism.

CSSM_CheckCssmExemption

Purpose

This describes the operations APIs for the OSCF Privilege Mechanism.This function returns the exemptions possessed by the current thread. For OCSF, if the exemption returned is non-zero, it implies that the CSSM_RequestCssmExemption API had been called to request the specific set of exemptions and the application is running with the OCSF Security Level 3 feature installed.

Format

```
CSSM_CheckCssmExemptionCSSM_RETURN CSSMAPICSSM_CheckCssm ExemptionCSSM_EXEMPTION_MASK *Exemptions)
```

Parameters

Output

Exemptions
A bit-mask of all exemptions possessed by the calling thread.

Return Value

A CSSM_OK return value signifies the operation was successful and that the exemption returned is valid. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_RequestCssmExemption

CSSM_QueryModulePrivilege

Purpose

The function returns the privileges available to the application. On z/OS, the privileges available depend upon whether only the OCSF base is installed or if the OCSF Security Level 3 feature is installed. When the OCSF Security Level 3 feature is installed, all privileges are available. When only the OCSF base is installed, no privileges are available. The application file name and application path name must be specified for compatability with other implementations of the interface, but are not used.

An application may invoke this function to determine privileges available to the application.

Format

```
CSSM_RETURN CSSM_QueryModulePrivilege (const char *AppFileName,
                                       const char *AppPathName,
                                       CSSM_EXEMPTION_MASK *PrivilegeSet)
```

Parameters

Input

AppFileName
The module file name for the application.

AppPathName
The path to the file that implements the module.

Output

PrivilegeSet
A bitmask specifying all the privileges that the module has.

Return Value

This function returns CSSM_OK if credential verification was successful and a privilege set was retrieved. On error CSSM_FAIL is returned. Use CSSM_GetError to obtain the error code.

CSSM_RequestCssmExemption

Purpose

Privileges/Exemptions are only tolerated on the OCSF and therefore behave differently than on other implementations. When the OCSF Security Level 3 feature is installed, the requested exemptions are granted automatically. In this case, the *AppFileName* and *AppPathName* parameters may be left as NULL. When only the OSCF base is installed, no exemptions are available. For compatability the *AppFileName* and *AppPathName* must be specified but they will not be used.

The exemption mask defines the requested exemptions. Applications may invoke this function multiple times. Each successful verification replaces the previously granted exemptions. If the *ExemptionRequest* parameter is zero, all privileges are dropped for that thread.

It may be noted that the *AppFileName* and *AppPathName* parameters may be left as NULL if it is known for sure that the requested exemptions are a subset of the currently possessed exemptions.

Format

```
CSSM_RETURN CSSMAPI CSSM_RequestCssmExemption(CSSM_EXEMPTION_MASK ExemptionRequest,
                                              const char *AppFileName,
                                              const char *AppPathName,
                                              const void * Reserved)
```

Parameters

Input

ExemptionRequest
A bit-mask of all exemptions being requested by the caller. If the value is CSSM_EXEMPT_ALL, the caller is requesting all possible privileges that may be granted according to the credentials that are presented and checked.

AppFileName
The name of the file that implements the application. This file is not used by OCSF but is required for compatibility.

AppPathName
The path to the file that implements the application. This file is not used by OCSF but is required for compatibility.

Input/optional

Reserved
A reserved input.

Return Value

A CSSM_OK return value signifies the verification operation was successful and the exemption has been granted. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CheckCssmExemption

Chapter 12. Cryptographic Services API

Cryptographic Service Providers (CSPs) are service provider modules which perform cryptographic operations including encryption, decryption, digital signaturing, key and key pair generation, random number generation (RNG), message digest, key wrapping, key unwrapping, and key exchange. Cryptographic services can be implemented by a hardware-software combination, by software only, or by hardware only. Besides the traditional cryptographic functions, CSPs may provide other vendor-specific services. The set of services provided can be dynamic even after a caller has attached the CSP for service. This means the capabilities registered when the CSP was installed can change during execution based on changes internal or external to the system.

The CSP is always responsible for the secure storage of private keys. Optionally, the CSP may assume responsibility for the secure storage of other object types, such as symmetric keys and certificates. The implementation of secured persistent storage for keys can use the services of a Data Storage Library (DL) module within the OCSF Framework or some approach internal to the CSP. Accessing persistent objects managed by the CSP, other than keys, is performed using OCSF's DL application programming interfaces (APIs).

CSPs optionally support a password-based login sequence. When login is supported, the caller is allowed to change passwords as deemed necessary. This is part of a standard user-initiated maintenance procedure. Some CSPs support operations for privileged CSP administrators. The model for CSP administration varies widely among CSP implementations. For this reason, OCSF does not define APIs for vendor-specific CSP administration operations. CSP vendors can makes these services available to CSP administration tools using the CSSM_CSP_Passthrough function.

The range and types of cryptographic services a CSP supports are at the discretion of the vendor. A registry and query mechanism is available through the OCSF for CSPs to disclose the services and details about the services. As an example, a CSP may register this with the OCSF: Encryption is supported, algorithms present are Data Encryption Standard (DES) with cipher block chaining for key sizes 40 and 56 bits, and triple DES with three keys for key-size 168 bits.

All cryptographic services requested by applications will be channeled to one of the CSPs through OCSF. CSP vendors only need target their modules to OCSF for all security-conscious applications to have access to their product.

Calls made to a CSP to perform cryptographic operations occur within a framework called a *session*, which is established and terminated by the application. Applications must create a *session context* (simply referred to as the *context*) prior to starting CSP operations and delete it as soon as possible upon completion of the operation. Context information is not persistent; it is not saved permanently in a file or database.

Before an application calls a CSP to perform a cryptographic operation, the application uses the query services function to determine what CSPs are installed and what services they provide. Based on this information, the application then can determine which CSP to use for subsequent operations; the application creates a session with this CSP and performs the operation.

Depending on the class of cryptographic operations, individualized attributes are available for the cryptographic context. Besides specifying an algorithm when creating the context, the application may also initialize a session key, pass an initialization vector and/or pass padding information to complete the description of the session. A successful return value from the create function indicates the desired CSP is available. Functions are also provided to manage the created context. When a context is no longer required, the application calls CSSM_DeleteContext. Resources that were allocated for that context can be reclaimed by the operating system.

There are two basic types of cryptographic operations – a single call to perform an operation and a staged method of performing the operation. For the single call method, only one call is needed to obtain the result. For the staged method, there is an initialization call followed by one or more update calls, and ending with a completion (final) call. The result is available after the final function completes its execution for most cryptographic operations – staged encryption/decryption are an exception in that each update call generates a portion of the result.

Data Structures

This describes the data structures for the CSP.

CSSM_CALLBACK

```
typedef CSSM_DATA_PTR (CSSMAPI *CSSM_CALLBACK) (void *allocRef, uint32 ID);
```

Definitions:

allocRef
Memory heap reference specifying which heap to use for memory allocation.

ID Input data to identify the callback

CSSM_CC_HANDLE

```
typedef uint32 CSSM_CC_HANDLE/* Cryptographic Context Handle */
```

CSSM_CONTEXT

```
typedef struct cssm_context {
    uint32 ContextType;
    uint32 AlgorithmType;
    uint32 Reserve;
    uint32 NumberOfAttributes;
    CSSM_CONTEXT_ATTRIBUTE_PTR ContextAttributes;
    CSSM_BOOL Privileged;
    uint32 EncryptionProhibited;
    uint32 WorkFactor;
} CSSM_CONTEXT, *CSSM_CONTEXT_PTR
```

Definitions:

ContextType
An identifier describing the type of services for this context, as shown in Table 23.

AlgorithmType
An ID number describing the algorithm to be used (see Table 24 on page 110).

Table 23. Context Types

Value	Description
CSSM_ALGCLASS_NONE	Null Context type
CSSM_ALGCLASS_CUSTOM	Custom algorithms
CSSM_ALGCLASS_KEYXCH	Key Exchange algorithms
CSSM_ALGCLASS_SIGNATURE	Signature algorithms
CSSM_ALGCLASS_SYMMETRIC	Symmetric Encryption algorithms
CSSM_ALGCLASS_DIGEST	Message Digest algorithms
CSSM_ALGCLASS_RANDOMGEN	Random Number Generation algorithms
CSSM_ALGCLASS_UNIQUEGEN	Unique ID Generation algorithms

Table 23. Context Types (continued)

Value	Description
CSSM_ALGCLASS_MAC	Message Authentication Code (MAC) algorithms
CSSM_ALGCLASS_ASYMMETRIC	Asymmetric Encryption algorithms
CSSM_ALGCLASS_KEYGEN	Key Generation algorithms
CSSM_ALGCLASS_DERIVEKEY	Key Derivation algorithms
CSSM_ALGCLASS_KEY_RECOVERY_ENABLEMENT	Key Recovery Enablement algorithms
CSSM_ALGCLASS_KEY_RECOVERY_REGISTRATION	Key Recovery Registration algorithms
CSSM_ALGCLASS_KEY_RECOVERY_REQUEST	Key Recovery Request algorithms

Table 24. Algorithms for a Session Context

Value	Description
CSSM_ALGID_NONE	Null algorithm
CSSM_ALGID_CUSTOM	Custom algorithm
CSSM_ALGID_DH	Diffie-Hellman key exchange algorithm
CSSM_ALGID_PH	Pohlig-Hellman key exchange algorithm
CSSM_ALGID_KEA	Key Exchange algorithm
CSSM_ALGID_MD2	MD2hash algorithm
CSSM_ALGID_MD4	MD4hash algorithm
CSSM_ALGID_MD5	MD5hash algorithm
CSSM_ALGID_SHA1	Secure Hash algorithm
CSSM_ALGID_NHASH	N-Hash algorithm
CSSM_ALGID_HAVAL	HAVAL hash algorithm (MD5 variant)
CSSM_ALGID_RIPEMD	RIPE-MD hash algorithm (MD4 variant - developed for the European Community's RIPE project)
CSSM_ALGID_IBCHASH	IBC-Hash (keyed hash algorithm or MAC)
CSSM_ALGID_RIPEMAC	RIPE-MAC
CSSM_ALGID_DES	Data Encryption Standard block cipher
CSSM_ALGID_DESX	DESX block cipher (DES variant from RSA)
CSSM_ALGID_RDES	RDES block cipher (DES variant)
CSSM_ALGID_3DES_3KEY	Triple-DES block cipher (with 3 keys)
CSSM_ALGID_3DES_2KEY	Triple-DES block cipher (with 2 keys)
CSSM_ALGID_3DES_1KEY	Triple-DES block cipher (with 1 key) Lucifer block cipher
CSSM_ALGID_IDEA	International Data Encryption Algorithm (IDEA) block cipher
CSSM_ALGID_RC2	RC2 block cipher
CSSM_ALGID_RC5	RC5 block cipher
CSSM_ALGID_RC4	RC4 stream cipher

Table 24. Algorithms for a Session Context (continued)

Value	Description
CSSM_ALGID_SEAL	SEAL stream cipher
CSSM_ALGID_CAST	CAST block cipher
CSSM_ALGID_BLOWFISH	BLOWFISH block cipher
CSSM_ALGID_SKIPJACK	Skipjack block cipher
CSSM_ALGID_LUCIFER	Lucifer block cipher
CSSM_ALGID_MADRYGA	Madryga block cipher
CSSM_ALGID_FEAL	FEAL block cipher
CSSM_ALGID_REDOC	REDOC 2 block cipher
CSSM_ALGID_REDOC3	REDOC 3 block cipher
CSSM_ALGID_LOKI	LOKI block cipher
CSSM_ALGID_KHUFU	KHUFU block cipher
CSSM_ALGID_KHAFRE	KHAFRE block cipher
CSSM_ALGID_MMB	MMB block cipher (IDEA variant)
CSSM_ALGID_GOST	GOST block cipher
CSSM_ALGID_SAFER	SAFER K-40, K-64, K-128 block cipher
CSSM_ALGID_CRAB	CRAB block cipher
CSSM_ALGID_RSA	RSA public key cipher
CSSM_ALGID_DSA	Digital Signature algorithm
CSSM_ALGID_MD5WithRSA	MD5/RSA signature algorithm
CSSM_ALGID_MD2WithRSA	MD2/RSA signature algorithm
CSSM_ALGID_ElGamal	ElGamal signature algorithm
CSSM_ALGID_MD2Random	MD2-based random numbers
CSSM_ALGID_MD5Random	MD5-based random numbers
CSSM_ALGID_SHARandom	SHA-based random numbers
CSSM_ALGID_DESRandom	DES-based random numbers
CSSM_ALGID_SHA1WithRSA	SHA-1/RSA signature algorithm
CSSM_ALGID_RSA_PKCS	RSA as specified in PKCS#1
CSSM_ALGID_RSA_ISO9796	RSA as specified in International Organization for Standardization (ISO) 9796
CSSM_ALGID_RSA_RAW	Raw RSA as assumed in X.509
CSSM_ALGID_CDMF	CDMF block cipher
CSSM_ALGID_CAST3	Entrust's CAST3 block cipher
CSSM_ALGID_CAST5	Entrust's CAST5 block cipher
CSSM_ALGID_GenericSecret	Generic secret operations
CSSM_ALGID_ConcatBaseAndKey	Concatenate two keys, base key first
CSSM_ALGID_ConcatKeyAndBase	Concatenate two keys, base key last
CSSM_ALGID_ConcatBaseAndData	Concatenate base key and random data, key first
CSSM_ALGID_ConcatDataAndBase	Concatenate base key and data, data first
CSSM_ALGID_XORBaseAndData	XOR a byte string with the base key

Table 24. Algorithms for a Session Context (continued)

Value	Description
CSSM_ALGID_ExtractFromKey	Extract a key from base key, starting at arbitrary bit position
CSSM_ALGID_SSL3PreMasterGen	Generate a 48-byte SSL 3 premaster key
CSSM_ALGID_SSL3MasterDerive	Derive an SSL 3 key from a premaster key
CSSM_ALGID_SSL3KeyAndMacDerive	Derive the keys and MACing keys for the SSL cipher suite
CSSM_ALGID_SSL3MD5_MAC	Performs SSL 3 MD5 MACing
CSSM_ALGID_SSL3SHA1_MAC	Performs SSL 3 SHA-1 MACing
CSSM_ALGID_MD5Derive	Generate key by MD5 hashing a base key
CSSM_ALGID_MD2Derive	Generate key by MD2 hashing a base key
CSSM_ALGID_SHA1Derive	Generate key by SHA-1 hashing a base key
CSSM_ALGID_WrapLynks	Spyrus LYNKS DES based wrapping scheme w/checksum
CSSM_ALGID_WrapSET_OAEP	SET key wrapping
CSSM_ALGID_BATON	Fortezza BATON cipher
CSSM_ALGID_ECDSA	Elliptic Curve DSA
CSSM_ALGID_MAYFLY	Fortezza MAYFLY cipher
CSSM_ALGID_JUNIPER	Fortezza JUNIPER cipher
CSSM_ALGID_FASTHASH	Fortezza FASTHASH
CSSM_ALGID_3DES	Generic 3DES
CSSM_ALGID_SSL3MD5	SSL3MD5
CSSM_ALGID_SSL3SHA1	SSL3SHA1
CSSM_ALGID_FortezzaTimestamp	FortezzaTimestamp
CSSM_ALGID_SHA1WithDSA	SHA1WithDSA
CSSM_ALGID_SHA1WithECDSA	SHA1WithECDSA
CSSM_ALGID_DSA_BSAFE	BSAFE Key format

Some of the algorithms in Table 24 on page 110 operate in a variety of modes. The desired mode is specified using an attribute of type CSSM_ATTRIBUTE_MODE. The valid values for the mode attribute are as follows in Table 25.

Table 25. Modes of Algorithms

Value	Description
CSSM_ALGMODE_NONE	Null algorithm mode
CSSM_ALGMODE_CUSTOM	Custom mode
CSSM_ALGMODE_ECB	Electronic Code Book
CSSM_ALGMODE_ECBPad	ECB with padding
CSSM_ALGMODE_CBC	Cipher Block Chaining
CSSM_ALGMODE_CBC_IV8	CBC with Initialization Vector of 8 bytes
CSSM_ALGMODE_CBCPadIV8	CBC with padding and Initialization Vector of 8 bytes
CSSM_ALGMODE_CFB	Cipher FeedBack

Table 25. Modes of Algorithms (continued)

Value	Description
CSSM_ALGMODE_CFB_IV8	CFB with Initialization Vector of 8 bytes
CSSM_ALGMODE_CFBPad_IV8	CFB with Initialization Vector of 8 bytes and padding
CSSM_ALGMODE_OFB	Output FeedBack
CSSM_ALGMODE_OFB_IV8	OFB with Initialization Vector of 8 bytes
CSSM_ALGMODE_OFBPadIV8	OFB with Initialization Vector of 8 bytes and padding
CSSM_ALGMODE_COUNTER	Counter
CSSM_ALGMODE_BC	Block Chaining
CSSM_ALGMODE_PCBC	Propagating CBC
CSSM_ALGMODE_CBCC	CBC with Checksum
CSSM_ALGMODE_OFBNLF	OFB with Nonlinear Function
CSSM_ALGMODE_PBC	Plaintext Block Chaining
CSSM_ALGMODE_PFB	Plaintext FeedBack
CSSM_ALGMODE_CBCPD	CBC of Plaintext Difference
CSSM_ALGMODE_PUBLIC_KEY	Use the public key
CSSM_ALGMODE_PRIVATE_KEY	Use the private key
CSSM_ALGMODE_SHUFFLE	Fortezza shuffle mode
CSSM_ALGMODE_ECB64	Electronic Code Book (64 bits)
CSSM_ALGMODE_CBC64	Cipher Block Chaining (64 bits)
CSSM_ALGMODE_OFB64	Output FeedBack (64 bits)
CSSM_ALGMODE_CFB64	Cipher FeedBack (64 bits)
CSSM_ALGMODE_CFB32	Cipher FeedBack (32 bits)
CSSM_ALGMODE_CFB16	Cipher FeedBack (16 bits)
CSSM_ALGMODE_CFB8	Cipher FeedBack (8 bits)
CSSM_ALGMODE_WRAP	SKIPJACK Wrap mechanism
CSSM_ALGMODE_PRIVATE_WRAP	SKIPJACK Private Wrap mechanism
CSSM_ALGMODE_RELAYX	SKIPJACK RELAYX mechanism
CSSM_ALGMODE_ECB128	Electronic Code Book (128 bits)
CSSM_ALGMODE_ECB96	Electronic Code Book (96 bits)
CSSM_ALGMODE_CBC128	Cipher Block Chaining (128 bits)
CSSM_ALGMODE_OAEP_HASH	Optimal Asymmetric Encryption Padding (OAEP) for RSA

Definitions:

NumberOfAttributes
Number of attributes associated with this service.

ContextAttributes
Pointer to data that describes the attributes. To retrieve the next attribute, advance the attribute pointer.

Privileged
When this flag is CSSM_TRUE, the context can perform cryptographic operations without being forced to follow the key recovery policy.

EncryptionProhibited
An integer indicating whether encryption is allowed. If encryption is allowed, this field is zero. Otherwise, the flags indicate which policy disallowed encryption.

WorkFactor
WorkFactor is the maximum number of bits that can be left out of Key Recovery Fields (KRFs) when they are generated. The recovery of the key must then search this number of bits to recover the key.

CSSM_CONTEXT_ATTRIBUTE

```
typedef struct cssm_context_attribute{
    uint32 AttributeType;
    uint32 AttributeLength;
    union {
        char *String;
        uint32 Uint32;
        CSSM_CRYPTO_DATA_PTR Crypto;
        CSSM_KEY_PTR Key;
        CSSM_DATA_PTR Data;
        CSSM_DATE_PTR Date;
        CSSM_RANGE_PTR Range;
        CSSM_VERSION_PTR Version;
        CSSM_KR_PROFILE_PTR KRProfile;
    } Attribute;
} CSSM_CONTEXT_ATTRIBUTE, *CSSM_CONTEXT_ATTRIBUTE_PTR;
```

Definitions:

AttributeType
An identifier describing the type of attribute. Valid attribute types are as follows in Table 26.

Table 26. Attribute Types

Value	Description	Data Type
CSSM_ATTRIBUTE_NONE	No attribute	None
CSSM_ATTRIBUTE_CUSTOM	Custom data	Opaque pointer
CSSM_ATTRIBUTE_DESCRIPTION	Description of attribute	String
CSSM_ATTRIBUTE_KEY	Key Data	CSSM_KEY
CSSM_ATTRIBUTE_INIT_VECTOR	Initialization vector	CSSM_DATA
CSSM_ATTRIBUTE_SALT	Salt	CSSM_DATA
CSSM_ATTRIBUTE_PADDING	Padding information	uint32
CSSM_ATTRIBUTE_RANDOM	Random data	CSSM_DATA
CSSM_ATTRIBUTE_SEED	Seed	CSSM_CRYPTO_DATA
CSSM_ATTRIBUTE_PASSPHRASE	Passphrase	CSSM_CRYPTO_DATA
CSSM_ATTRIBUTE_KEY_LENGTH	Key length specified in bits	uint32

Table 26. Attribute Types (continued)

Value	Description	Data Type
CSSM_ATTRIBUTE_KEY_LENGTH_RANGE	Key length range specified in bits	CSSM_RANGE
CSSM_ATTRIBUTE_BLOCK_SIZE	Block size	uint32
CSSM_ATTRIBUTE_OUTPUT_SIZE	Output size	uint32
CSSM_ATTRIBUTE_ROUNDS	Number of runs or rounds	uint32
CSSM_ATTRIBUTE_IV_SIZE	Size of initialization vector	uint32
CSSM_ATTRIBUTE_ALG_PARAMS	Algorithm parameters	CSSM_DATA
CSSM_ATTRIBUTE_LABEL	Label placed on an object when it is created	CSSM_DATA
CSSM_ATTRIBUTE_KEY_TYPE	Type of key to generate or derive	uint32
CSSM_ATTRIBUTE_MODE	Algorithm mode to use for encryption	uint32
CSSM_ATTRIBUTE_EFFECTIVE_BITS	Number of effective bits used in the RC2 cipher	uint32
CSSM_ATTRIBUTE_START_DATE	Starting date for an object's validity	CSSM_DATE
CSSM_ATTRIBUTE_END_DATE	Ending date for an object's validity	CSSM_DATE
CSSM_ATTRIBUTE_KEYUSAGE	Key usage	uint32
CSSM_ATTRIBUTE_KEYATTR	Key attributes	uint32
CSSM_ATTRIBUTE_VERSION	Object version	CSSM_VERSION
CSSM_ATTRIBUTE_ALG_ID	Algorithm ID	uint32
CSSM_ATTRIBUTE_ITERATION_COUNT	Number of iterations	uint32
CSSM_ATTRIBUTE_ROUNDS_RANGE	Minimum and maximum number of rounds	CSSM_RANGE
CSSM_ATTRIBUTE_KRPROFILE_LOCAL	Key Recovery Profile for the local user	CSSM_KR_PROFILE

Table 26. Attribute Types (continued)

Value	Description	Data Type
CSSM_ATTRIBUTE_KRPROFILE_REMOTE	Key Recovery Profile for the remote user	CSSM_KR_PROFILE

The data referenced by a CSSM_ATTRIBUTE_CUSTOM attribute must be a single continuous memory block. This allows the OCSF to appropriately release all dynamically allocated memory resources.

Definitions:

AttributeLength
Length of the attribute data.

Attribute
Union representing the attribute data. The union member used is named after &tab;the type of data contained in the attribute. See Table 26 on page 114 for the data types associated with each attribute type.

CSSM_CONTEXT_INFO

```
typedef CSSM_CONTEXT CSSM_CONTEXT_INFO
```

CSSM_CRYPTO_DATA

```
typedef struct cssm_crypto_data {
                 CSSM_DATA_PTR Param;
                 CSSM_CALLBACK Callback;
                 uint32 CallbackID;
}CSSM_CRYPTO_DATA, *CSSM_CRYPTO_DATA_PTR
```

Definitions:

Param
A pointer to the parameter data and its size in bytes.

Callback
An optional callback routine for the service provider modules to obtain the parameter.

ID A tag that identifies the callback.

CSSM_CSP_CAPABILITY

```
typedef CSSM_CONTEXT CSSM_CSP_CAPABILITY, *CSSM_CSP_CAPABILITY_PTR;
```

CSSM_CSP_FLAGS

```
typedef uint32 CSSM_CSP_FLAGS;
```

CSSM_CSP_HANDLE

The CSSM_CSP_HANDLE is used to identify the association between an application thread and an instance of a CSP module. It is assigned when an application causes OCSF to attach to a CSP. It is freed when an application causes OCSF to detach from a CSP. The application uses the CSSM_CSP_HANDLE with every CSP function call to identify the targeted CSP. The CSP uses the CSSM_CSP_HANDLE to identify the appropriate application's memory management routines when allocating memory on the application's behalf.

```
typedef uint32 CSSM_CSP_HANDLE/* Cryptographic Service Provider Handle */
```

CSSM_CSP_SESSION_TYPE

The CSSM_CSP_SESSION_TYPE is provided in Table 27.

Table 27. Session Types

Value		Descriptions
CSSM_CSP_SESSION_EXCLUSIVE	0 x 0001	Single user CSP.
CSSM_CSP_SESSION_READWRITE	0 x 0002	Caller can read and write objects such as keys in the CSP.
CSSM_CSP_SESSION_SERIAL	0 x 0004	Multiuser, reentrant CSP that requires serial access.

CSSM_CSPSUBSERVICE

Three structures are used to contain all of the static information that describes a CSP module: cssm_moduleinfo, cssm_serviceinfo, and cssm_cspsubservice. This descriptive information is securely stored in the OCSF registry when the CSP module is installed with CSSM. A CSP module may implement multiple types of services and organize them as subservices.

The descriptive information stored in these structures can be queried using the function CSSM_GetModuleInfo and specifying the CSP module Globally Unique ID (GUID).

```
typedef struct cssm_cspsubservice {
    uint32 SubServiceId;
    CSSM_STRING Description;
    CSSM_CSP_FLAGS CspFlags; /* General flags defined by CSSM for CSPs */
    uint32 CspCustomFlags;   /* Flags defined by individual CSP */
    uint32 AccessFlags;      /* Access Flags used by CSP */
    CSSM_CSPTYPE CspType;    /* CSP type number for dereferencing CspInfo */
    union {                  /* info struct of type defined by CspType */
        CSSM_SOFTWARE_CSPSUBSERVICE_INFO SoftwareCspSubService;
        CSSM_HARDWARE_CSPSUBSERVICE_INFO HardwareCspSubService;
#ifndef _MVS_
    };
#else
  /* Use the CDSA Version 2.0 definition instead of the anonymous union of
     the Version 1.x spec which unfortunately is not ANSI-C compatible. */

  }SubServiceInfo;
#endif
  CSSM_CSP_WRAPPEDPRODUCT_INFO WrappedProduct;
}CSSM_CSPSUBSERVICE, *CSSM_CSPSUBSERVICE_PTR;
```

Definitions:

SubServiceId
: The subservice ID required for an attach call to connect a CSP to an individual subservice within a CSP.

Description
: A NULL-terminated character string containing a text description of the subservice.

CspFlags
: A bit-mask containing general flags defined by OCSF for CSPs. The mask may contain one or a combination of these in Table 28.

Table 28. CSP Flags

CSSM_CSP_FLAGS Values	Description
CSSM_CSP_STORES_PRIVATE_KEYS	CSP can store private keys.

Table 28. CSP Flags (continued)

CSSM_CSP_FLAGS Values	Description
CSSM_CSP_STORES_PUBLIC_KEYS	CSP can store public keys.
CSSM_CSP_STORES_SESSION_KEYS	CSP can store session/secret keys.

CspCustomFlags
Flags defined by the vendor. Consult the individual CSP User's Guide for the list of valid flags.

AccessFlags
Flags that are required to be provided by the application during an attach call when specifying the subservice ID given in *SubServiceId*.

CspType
Identifier that determines the type of CSP information structure referenced by *CspInfo*. The values and their corresponding CSP information structures are currently defined in Table 29.

Table 29. CSP Informtation Type Identifiers and Associated Structure Types

CSP Information Structure Identifier	Structure Type
CSSM_CSP_TYPE_SOFTWARE	CSSM_CSP_TYPE_SOFTWARE_INFO
CSSM_CSP_TYPE_PKCS11	CSSM_CSP_TYPE_PKCS11_INFO

SoftwareCspSubService/HardwareCspSubService
A CSP information structure of the type specified by *CspType*.

WrappedProduct
Pointer to a CSSM_CSP_WRAPPEDPRODUCTINFO structure describing a product that is wrapped by the CSP.

CSSM_CSPTYPE

```
typedef uint32 CSSM_CSPTYPE;
#define CSSM_CSP_SOFTWARE 1
#define CSSM_CSP_HARDWARE 2
```

CSSM_CSP_WRAPPEDPRODUCTINFO

```
typedef struct cssm_csp_wrappedproductinfo {
    CSSM_VERSION StandardVersion;
    CSSM_STRING StandardDescription;
    CSSM_VERSION ProductVersion;
    CSSM_STRING ProductDescription;
    CSSM_STRING ProductVendor;
    uint32 ProductFlags;
} CSSM_CSP_WRAPPEDPRODUCTINFO,*CSSM_CSP_WRAPPEDPRODUCTINFO_PTR;
```

Definitions:

StandardVersion
Version of the standard to which the wrapped product complies.

StandardDescription
A NULL-terminated character string containing a text description of the standard to which the wrapped product complies.

ProductVersion
Version of the product wrapped by the CSP.

ProductDescription
A NULL-terminated character string containing a text description of the product wrapped by the CSP.

ProductVendor
A NULL-terminated character string containing the name of the wrapped product's vendor.

ProductFlags
This version of OCSF has no flags defined. This field must be set to zero.

CSSM_DATA

The CSSM_DATA structure is used to associate a length, in bytes, with an arbitrary block of contiguous memory. This memory must be allocated and freed using the memory management routines provided by the calling application via OCSF.

```
typedef struct cssm_data{
    uint32 Length;/* in bytes */
    uint8 *Data;
} CSSM_DATA, *CSSM_DATA_PTR
```

Definitions:

Length
Length of the data buffer in bytes.

Data
Points to the start of an arbitrary length data buffer.

CSSM_DATE

```
typedef struct cssm_date {
    uint8 Year[4];
    uint8 Month[2];
    uint8 Day[2];
} CSSM_DATE, *CSSM_DATE_PTR
```

Definitions:

Year
Four-digit integer array representation of the year.

Month
Two-digit representation of the month.

Day
Two-digit representation of the day.

CSSM_HARDWARERECSPSUBSERVICEINFO

```
typedef struct cssm_hardwarecspsubserviceinfo {
    uint32 NumberOfCapabilities;
    CSSM_CSP_CAPABILITY_PTR CapabilityList;
    void * Reserved;

    /* Reader/Slot Info */
    CSSM_STRING ReaderDescription;
    CSSM_STRING ReaderVendor;
    CSSM_STRING ReaderSerialNumber;
    CSSM_VERSION ReaderHardwareVersion;
    CSSM_VERSION ReaderFirmwareVersion;
    uint32 ReaderFlags;
    uint32 ReaderCustomFlags;

    CSSM_STRING TokenDescription;
    CSSM_STRING TokenVendor;
    CSSM_STRING TokenSerialNumber;
    CSSM_VERSION TokenHardwareVersion;
    CSSM_VERSION TokenFirmwareVersion;

    uint32 TokenFlags;
    uint32 TokenCustomFlags;
```

```
    uint32 TokenMaxSessionCount;
    uint32 TokenOpenedSessionCount;
    uint32 TokenMaxRWSessionCount;
    uint32 TokenOpenedRWSessionCount;
    uint32 TokenTotalPublicMem;
    uint32 TokenFreePublicMem;
    uint32 TokenTotalPrivateMem;
    uint32 TokenFreePrivateMem;
    uint32 TokenMaxPinLen;
    uint32 TokenMinPinLen;
    char TokenUTCTime[16];

    char *UserLabel;
    CSSM_DATA UserCACertificate;
} CSSM_HARDWARE_CSPSUBSERVICE_INFO,*CSSM_HARDWARE_CSPSUBSERVICE_INFO_PTR;
```

Definitions:

NumberOfCapabilities
Number of capabilities in list.

CapabilityList
A context list that specifies the capabilities of the CSP.

Reserved
This field is reserved for future use and must always be set to NULL.

ReaderDescription
A NULL-terminated character string containing a description of the device reader.

ReaderVendor
A NULL-terminated string that contains the name of the reader vendor.

ReaderSerialNumber
A NULL-terminated string that contains the serial number of the reader.

ReaderHardwareVersion
Hardware version of the reader.

ReaderFirmwareVersion
Firmware version of the reader.

ReaderFlags
Bit-mask containing information about the reader. The flags specified in the mask are as follows in Table 30.

Table 30. PKCS#11 CSP Reader Flags

Reader Flag	Description
CSSM_CSP_RDR_TOKENPRESENT	Token is present in the reader.
CSSM_CSP_RDR_REMOVABLE	Reader supports removable tokens.
CSSM_CSP_RDR_HW	Reader is a hardware device.

ReaderCustomsFlags
Flags defined by the vendor. Consult the individual CSP User's Guide for the list of valid flags.

The fields may not be valid if the CSSM_CSP_RDR_TOKENPRESENT flag is not set in the *ReaderFlags* field. Unknown string and CSSM_DATA fields will be set to NULL, integer and date fields will be set to zero and, flag fields will have all flags set to false.

TokenDescription
A NULL-terminated character string that contains a text description of the token. This value may be NULL or equal to *ReaderDescription* if the token is not removable.

TokenVendor
A NULL-terminated string that contains the name of the token vendor. This value may be NULL or equal to *ReaderVendor* if the token is not removable.

TokenSerialNumber
A NULL-terminated string that contains the serial number of the token. This value may be NULL or equal to *ReaderSerialNumber* if the token is not removable.

TokenHardwareVersion
Hardware version of the token.

TokenFirmwareVersion
Firmware version of the token.

TokenFlags
Bit-mask containing information about the token. The flags specified in the mask are provided in Table 31.

Table 31. PKCS#11 CSP Token Flags

Token Flags	Description
CSSM_CSP_TOK_RNG	Token has random number generator.
CSSM_CSP_TOK_WRITE_PROTECTED	Token is write-protected.
CSSM_CSP_TOK_LOGIN_REQUIRED	User must login to access private objects.
CSSM_CSP_TOK_USER_PIN_INITIALIZED	User's PIN has been initialized.
CSSM_CSP_TOK_EXCLUSIVE_SESSION	An exclusive session currently exists.
CSSM_CSP_TOK_CLOCK_EXISTS	Token has built-in clock.
CSSM_CSP_TOK_ASYNC_SESSION	Token supports asynchronous operations.
CSSM_CSP_TOK_PROT_AUTHENTICATION	Token has protected authentication path.
CSSM_CSP_TOK_DUAL_CRYPTO_OPS	Token supports dual cryptographic operations.

TokenCustomFlags
Flags defined by the vendor. Consult the individual CSP user's guide for the list of valid flags.

TokenMaxSessionCount
Maximum number of CSP handles referencing the token that may exist simultaneously.

TokenOpenedSessionCount
Number of CSP handles referencing the token that currently exist.

TokenTotalPublicMem
Amount of public storage space in the CSP. This value will be set to CSSM_VALUE_NOT_AVAILABLE if the CSP does not want to expose this information.

TokenFreePublicMem
Amount of public storage space available for use in the CSP. This value will be set to CSSM_VALUE_NOT_AVAILABLE (-1) if the CSP does not want to expose this information.

TokenTotalPrivateMem
Amount of private storage space in the CSP. This value will be set to CSSM_VALUE_NOT_AVAILABLE (-1) if the CSP does not want to expose this information.

TokenFreePrivateMem
Amount of private storage space available for use in the CSP. This value will be set to CSSM_VALUE_NOT_AVAILABLE if the CSP does not want to expose this information.

TokenMaxPinLen
Maximum length of passwords that can be used for authentication to the CSP.

TokenMinPinLen
Minimum length of passwords that can be used for authentication to the CSP.

TokenUTCTime
Character array containing the current Coordinated Universal Time (UTC) value in the CSP. The value is valid if the CSSM_CSP_TOK_CLOCK_EXISTS flag is true. The time is represented in the format YYYYMMDDhhmmssxx (4 characters for the year; 2 characters each for month, day, hour, minute, and second; and 2 additional reserved '0' characters).

UserLabel
A NULL-terminated string containing the label of the token.

UserCACertificate
Certificate of the Certificate Authority (CA).

CSSM_HEADERVISION

This data structure represents the version number of a key header structure. This version number is an integer that increments with each format revision of CSSM_KEYHEADER. The current revision number is represented by CSSM_KEYHEADER_VERSION, which equals 2 in this release of OCSF.

```
typedef uint32 CSSM_HEADERVERSION

#define CSSM_KEYHEADER_VERSION (2)
```

CSSM_KEY

This structure is used to represent keys in OCSF.

```
typedef struct cssm_key{
    CSSM_KEYHEADER KeyHeader;
    CSSM_DATA KeyData;
} CSSM_KEY, *CSSM_KEY_PTR;

typedef CSSM_KEY CSSM_WRAP_KEY, *CSSM_WRAP_KEY_PTR;
```

Definitions:
KeyHeader
Header describing the key, fixed length.
KeyData
Data representation of the key, variable length.

CSSM_KEYHEADER

The key header contains meta-data about a key. It contains information used by a CSP or application when using the associated key data. The service provider module is responsible for setting the appropriate values.

```
typedef struct cssm_keyheader {
    CSSM_HEADERVERSION HeaderVersion;
    CSSM_GUID CspId;
    uint32 BlobType;
    uint32 Format;
    uint32 AlgorithmId;
    uint32 KeyClass;
    uint32 KeySizeInBits;
    uint32 KeyAttr;
    uint32 KeyUsage;
    CSSM_DATE StartDate;
    CSSM_DATE EndDate;
    uint32 WrapAlgorithmId;
    uint32 WrapMode;
    uint32 Reserved;
} CSSM_KEYHEADER, *CSSM_KEYHEADER_PTR;
```

Definitions:

HeaderVersion
This is the version of the key header structure.

CspId
If known, the GUID of the CSP that generated the key. This value will not be known if a key is received from a third party, or extracted from a certificate.

BlobType
Describes the basic format of the key data. It can be any one of the values in Table 32.

Table 32. Keyblob Type Identifiers

Identifier	Description
CSSM_KEYBLOB_RAW	The blob is a clear, raw key.
CSSM_KEYBLOB_RAW_BERDER	The blob is a clear key, DER-encoded.
CSSM_KEYBLOB_REFERENCE	The blob is a reference to a key.
CSSM_KEYBLOB_WRAPPED	The blob is a wrapped RAW key.
CSSM_KEYBLOB_WRAPPED_BERDER	The blob is a wrapped DER-encoded key.
CSSM_KEYBLOB_OTHER	Other keyblob type.

Format
Describes the detailed format of the key data based on the value of the *BlobType* field. If the blob type has a non-reference basic type, then a CSSM_KEYBLOB_RAW_FORMAT identifier must be used, otherwise a CSSM_KEYBLOB_REF_FORMAT identifier is used. Any of the values are valid as format identifiers in Table 33.

Table 33. Keyblob Format Identifiers

Keyblob Format Identifier	Description
CSSM_KEYBLOB_RAW_FORMAT_NONE	No further conversion needs to be done.
CSSM_KEYBLOB_RAW_FORMAT_PKCS1	RSA PKCS1 V1.5
CSSM_KEYBLOB_RAW_FORMAT_PKCS3	RSA PKCS3 V1.5
CSSM_KEYBLOB_RAW_FORMAT_MSCAPI	Microsoft CAPI V2.0
CSSM_KEYBLOB_RAW_FORMAT_PGP	PGP

Table 33. Keyblob Format Identifiers (continued)

Keyblob Format Identifier	Description
CSSM_KEYBLOB_RAW_FORMAT_FIPS186	U.S. Gov. FIPS 186 - DSS V
CSSM_KEYBLOB_RAW_FORMAT_BSAFE	RSA BSAFE V3.0
CSSM_KEYBLOB_RAW_FORMAT_PKCS11	RSA PKCS11 V2.0
CSSM_KEYBLOB_RAW_FORMAT_CDSA	Intel CDSA
CSSM_KEYBLOB_RAW_FORMAT_OTHER	Other, CSP defined.
CSSM_KEYBLOB_REF_FORMAT_INTEGER	Reference is a number or handle.
CSSM_KEYBLOB_REF_FORMAT_STRING	Reference is a string or name.
CSSM_KEYBLOB_REF_FORMAT_OTHER	Other, CSP defined.

AlgorithmId
The algorithm for which the key was generated. This value does not change when the key is wrapped. Any of the defined OCSF algorithm IDs may be used.

KeyClass
Class of key contained in the keyblob. Valid key classes are as follows in Table 34.

Table 34. Key Class Identifiers

Key Class Identifiers	Description
CSSM_KEYCLASS_PUBLIC_KEY	Key is a public key.
CSSM_KEYCLASS_PRIVATE_KEY	Key is a private key.
CSSM_KEYCLASS_SESSION_KEY	Key is a session or symmetric key.
CSSM_KEYCLASS_SECRET_PART	Key is part of secret key.
CSSM_KEYCLASS_OTHER	Other

KeySizeInBits
This is the logical size of the key in bits. The logical size is the value referred to when describing the length of the key. For instance, an RSA key would be described by the size of its modulus and a DSA key would be represented by the size of its prime. Symmetric key sizes describe the actual number of bits in the key. For example, DES keys would be 64 bits and an RC4 key could range from 1 to 128 bits.

KeyAttr
Attributes of the key represented by the data. These attributes are used by CSPs to convey information about stored or referenced keys. The attributes are represented as a bit-mask (see Table 35).

KeyUsage
A bit-mask representing the valid uses of the key. Any of the values are valid in Table 36 on page 125.

Table 35. Key Attribute Flags

Attribute	Description
CSSM_KEYATTR_PERMANENT	Key is stored persistently in the CSP, e.g., PKCS#11 token object.
CSSM_KEYATTR_PRIVATE	Key is a private object and protected by either a user login, a password, or both.

Table 35. Key Attribute Flags (continued)

Attribute	Description
CSSM_KEYATTR_MODIFIABLE	The key or its attributes can be modified.
CSSM_KEYATTR_SENSITIVE	Key is sensitive. It may only be extracted from the CSP in a wrapped state. It will always be false for raw keys.
CSSM_KEYATTR_ALWAYS_SENSITIVE	Key has always been sensitive. It will always be false for raw keys.
CSSM_KEYATTR_EXTRACTABLE	Key is extractable from the CSP. If this bit is not set, the key is either not stored in the CSP or cannot be extracted from the CSP under any circumstances. It will always be false for raw keys.
CSSM_KEYATTR_NEVER_EXTRACTABLE	Key has never been extractable. It will always be false for raw keys.

Table 36. Key Usage Flags

Usage Mask	Description
CSSM_KEYUSE_ANY	Key may be used for any purpose supported by the algorithm.
CSSM_KEYUSE_ENCRYPT	Key may be used for encryption.
CSSM_KEYUSE_DECRYPT	Key may be used for decryption.
CSSM_KEYUSE_SIGN	Key can be used to generate signatures. For symmetric keys this represents the ability to generate MACs.
CSSM_KEYUSE_VERIFY	Key can be used to verify signatures. For symmetric keys this represents the ability to verify MACs.
CSSM_KEYUSE_SIGN_RECOVER	Key can be used to perform signatures with message recovery. This form of a signature is generated using the CSSM_EncryptData API with the algorithm mode set to CSSM_ALGMODE_PRIVATE_KEY. This attribute is only valid for asymmetric algorithms.
CSSM_KEYUSE_VERIFY_RECOVER	Key can be used to verify signatures with message recovery. This form of a signature is verified using the CSSM_DecryptData API with the algorithm mode set to CSSM_ALGMODE_PRIVATE_KEY. This attribute is only valid for asymmetric algorithms.
CSSM_KEYUSE_WRAP	Key can be used to wrap another key.
CSSM_KEYUSE_UNWRAP	Key can be used to unwrap a key.
CSSM_KEYUSE_DERIVE	Key can be used as the source for deriving other keys.

StartDate
Date from which the corresponding key is valid. All fields of the CSSM_DATA structure will be set to zero if the date is unspecified or unknown. This date is not enforced by the CSP.

EndDate
Data that the key expires and can no longer be used. All fields of the

CSSM_DATA structure will be set to zero is the date if unspecified or unknown. This date is not enforced by the CSP.

WrapAlgorithmId
If the key data contains a wrapped key, this field contains the algorithm used to create the wrapped blob. This field will be set to CSSM_ALGID_NONE if the key is not wrapped.

WrapMode
If the wrapping algorithm supports multiple wrapping modes, this field contains the mode used to wrap the key. This field is ignored if the *WrapAlgorithmId* is CSSM_ALGID_NONE.

Reserved
This field is reserved for future use. It should always be set to zero.

CSSM_KEY_SIZE

This structure holds the physical key size and the effective key size for a given key. The metric used is bits. The number of effective bits is the number of key bits that can be used in a cryptographic operation compared with the number of bits that may be present in the key. When the number of effective bits is less than the number of actual bits, this is known as "dumbing down."

```
typedef struct cssm_key_size {
    uint32 KeySizeInBits;/* Key size in bits */
    uint32 EffectiveKeySizeInBits; /* Effective key size in bits */
} CSSM_KEYSIZE, *CSSM_KEYSIZE_PTR
```

Definitions:

KeySizeInBits
The actual number of bits in a key.

EffectiveKeySizeInBits
The number of key bits that can be used for cryptographic operations.

CSSM_KEY_TYPE

```
typedef uint32 CSSM_KEY_TYPE, *CSSM_KEY_TYPE_PTR;
```

CSSM_NOTIFY_CALLBACK

This data structure defines a pointer to a function that applications can use to invoke an application-supplied function.

```
typedef CSSM_RETURN (CSSMAPI *CSSM_NOTIFY_CALLBACK)
    (CSSM_MODULE_HANDLE ModuleHandle,
        uint32 Application,
        uint32 Reason,
        void * Param)
```

Definitions:

ModuleHandle
Handle of the module to which the notification applies.

Application
Application-specific context indicator. This value is specified when a service provider module is attached.

Reason
One of the values is specified in Table 37 on page 127.

Param
Used by the module that triggers the notification to pass relevant information about the notification to the application.

Table 37. Reasons

Reason	Value
CSSM_NOTIFY_SURRENDER	0
CSSM_NOTIFY_COMPLETE	1
CSSM_NOTIFY_DEVICE_REMOVED	2
CSSM_NOTIFY_DEVICE_INSERTED	3

CSSM_PADDING

```
typedef enum cssm_padding {
    CSSM_PADDING_NONE           = 0,
    CSSM_PADDING_CUSTOM         = CSSM_PADDING_NONE+1,
    CSSM_PADDING_ZERO           = CSSM_PADDING_NONE+2,
    CSSM_PADDING_ONE            = CSSM_PADDING_NONE+3,
    CSSM_PADDING_ALTERNATE      = CSSM_PADDING_NONE+4,
    CSSM_PADDING_FF             = CSSM_PADDING_NONE+5,
    CSSM_PADDING_PKCS5          = CSSM_PADDING_NONE+6,
    CSSM_PADDING_PKCS7          = CSSM_PADDING_NONE+7,
    CSSM_PADDING_CipherStealing = CSSM_PADDING_NONE+8,
    CSSM_PADDING_RANDOM         = CSSM_PADDING_NONE+9
} CSSM_PADDING;
```

CSSM_QUERY_SIZE_DATA

```
typedef struct cssm_query_size_data {
    uint32 SizeInputBlock;
    uint32 SizeOutputBlock;
} CSSM_QUERY_SIZE_DATA, *CSSM_QUERY_SIZE_DATA_PTR
```

Definitions:

SizeInputBlock
The size of the input block in bytes.

SizeOutputBlock
The size of the output block in bytes.

CSSM_RANGE

```
typedef struct cssm_range {
uint32 Min;/* inclusive minimium value */
uint32 Max;/* inclusive maximium value */
} CSSM_RANGE, *CSSM_RANGE_PTR
```

Definitions:

Min
Minimum value in the range.

Max
Maximum value in the range.

CSSM_SOFTWARECSPSUBSERVICEINFO

```
typedef struct cssm_softwarecspsubserviceinfo {
    uint32 NumberOfCapabilities;
    CSSM_CSP_CAPABILITY_PTR CapabilityList;
    void* Reserved;
} CSSM_SOFTWARE_CSPSUBSERVICE_INFO, *CSSM_SOFTWARE_CSPSUBSERVICE_INFO_PTR;
```

Definitions:

NumberOfCapabilities
Number of capabilities availabilities available from the CSP.

CapabilityList
Pointer to an array of CSSM_CSP_CAPABILITY structures that represent the capabilities available from the CSP.

Reserved
Reserved for future use.

Cryptographic Context Operations

This describes the interfaces for the cryptographic context operations.

CSSM_CSP_CreateAsymmetricContext

On z/OS, when any CSSM_CSP_CreateAsymmetricContext operation is invoked, a copy of the context is created. The pointer to the copy is returned on all CSSM_GetContext calls.

Purpose

This function creates an asymmetric encryption cryptographic context and returns the cryptographic context handle. The handle can be used to call asymmetric encryption functions and cryptographic wrap/unwrap functions.

Format

```
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateAsymmetricContext
                (CSSM_CSP_HANDLE CSPHandle,
                uint32 AlgorithmID,
                const CSSM_CRYPTO_DATA_PTR PassPhrase,
                const CSSM_KEY_PTR Key,
                uint32 Padding)
```

Parameters

Input

CSPHandle
The handle that describes the CSP module used to perform this function. If a NULL handle is specified, OCSF returns an error.

AlgorithmID
The algorithm identification number for the algorithm used for asymmetric encryption.

PassPhrase
The passphrase is required to unlock the private key. The passphrase structure accepts an immediate value for the passphrase or the caller can specify a callback function the CSP can use to obtain the passphrase. The passphrase is needed only for signature operations, not verify operations. When the context is used for a wrap or unwrap operation, the passphrase can be used to generate a symmetric key for wrapping or unwrapping.

Key
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, the public key and passphrase are 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, the public key and the passphrase can be used to access the private key used to perform the unwrapping.

Input/optional

Padding
The method for padding. Typically specified for ciphers that pad.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_EncryptData
CSSM_QuerySize
CSSM_EncryptDataInit
CSSM_EncryptDataUpdate
CSSM_EncryptDataFinal
CSSM_DecryptData
CSSM_DecryptDataInit
CSSM_DecryptDataUpdate
CSSM_DecryptDataFinal
CSSM_GetContext
CSSM_SetContext
CSSM_DeleteContext
CSSM_GetContextAttribute
CSSM_UpdateContextAttributes

CSSM_CSP_CreateDeriveKeyContext

Purpose

On z/OS, when any CSSM_CSP_CreateDeriveKeyContext operation is invoked, a copy of the context is created. The pointer to the copy is returned on all CSSM_GetContext calls.

This function creates a cryptographic context to derive either a symmetric key or an asymmetric key, and returns a handle to the context. The cryptographic context handle can be used for calling the cryptographic derive key function.

Format

```
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateDeriveKeyContext
                 (CSSM_CSP_HANDLE CSPHandle,
                 uint32 AlgorithmID,
                 CSSM_KEY_TYPE DeriveKeyType,
                 uint32 DeriveKeyLength,
                 uint32 IterationCount,
                 const CSSM_DATA_PTR Salt,
                 const CSSM_CRYPTO_DATA_PTR Seed,
                 const CSSM_CRYPTO_DATA_PTR PassPhrase)
```

Parameters

Input

CSPHandle
The handle that describes the CSP module used to perform this function. If a NULL handle is specified, OCSF returns an error.

AlgorithmID
The algorithm identification number for a derived key algorithm.

DeriveKeyType
The type of key to derive.

DeriveKeyLength
The length of key to derive.

Input/optional

IterationCount
The number of iterations to be performed during the derivation process. Used heavily by password-based derivation methods.

Salt
A salt used to generate the key.

Seed
A seed used to generate a random number. The caller can both pass a seed and seed length in bytes or pass in a callback function. If NULL is passed, the CSP will use its default seed handling mechanism.

PassPhrase
The passphrase is required to unlock the private key. The passphrase structure accepts an immediate value for the passphrase or the caller can specify a callback function the CSP can use to obtain the passphrase. The passphrase is needed only for signature operations, not verify operations.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DeriveKey

CSSM_CSP_CreateDigestContext

Purpose

On z/OS, when any CSSM_CSP_CreateDigestContext operation is invoked, a copy of the context is created. The pointer to the copy is returned on all CSSM_GetContext calls.

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.

Format

```
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateDigestContext (CSSM_CSP_HANDLE CSPHandle, uint32 AlgorithmID)
```

Parameters

Input

CSPHandle
The handle that describes the CSP module used to perform this function. If a NULL handle is specified, OCSF returns an error.

AlgorithmID
The algorithm identification number for message digests.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DigestData
CSSM_DigestDataInit
CSSM_DigestDataUpdate
CSSM_DigestDataFinal
CSSM_GetContext
CSSM_SetContext
CSSM_DeleteContext
CSSM_GetContextAttributes
CSSM_UpdateContextAttributes

CSSM_CSP_CreateKeyGenContext

Purpose

On z/OS, when any CSSM_CSP_CreateKeyGenContext operation is invoked, a copy of the context is created. The pointer to the copy is returned on all CSSM_GetContext calls.

This function creates a key generation cryptographic context and returns a handle to the context. The cryptographic context handle can be used to call key/keypair generation functions.

Format

```
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateKeyGenContext
                 (CSSM_CSP_HANDLE CSPHandle,
                 uint32 AlgorithmID,
                 const CSSM_CRYPTO_DATA_PTR PassPhrase,
                 uint32 KeySizeInBits,
                 const CSSM_CRYPTO_DATA_PTR Seed,
                 const CSSM_DATA_PTR Salt,
                 const CSSM_DATE_PTR StartDate,
                 const CSSM_DATE_PTR EndDate,
                 const CSSM_DATA_PTR Params)
```

Parameters

Input

CSPHandle
The handle that describes the CSP module used to perform this function. If a NULL handle is specified, OCSF returns an error.

AlgorithmID
The algorithm identification number of the algorithm used for key generation.

PassPhrase
The passphrase is required to unlock the private key. The passphrase structure accepts an immediate value for the passphrase or the caller can specify a callback function the CSP can use to obtain the passphrase. The passphrase is needed only for signature operations, not verify operations. Once the new key is created, the passphrase or nickname must be provided in all future references to access the private or symmetric key.

KeySizeInBits
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). This is the effective key size.

Input/optional

Seed
A seed used to generate the key. The caller can either pass a seed or seed length in bytes or pass in a callback function. If NULL is passed, the CSP will use its default seed handling mechanism.

Salt
A Salt used to generate the key.

StartDate
Date from which the corresponding key is valid. All fields of the CSSM_DATE structure will be set to zero if the date is unspecified or unknown. The CSP module does not enforce this date.

EndDate
Data that the key expires and can no longer be used. All fields of the CSSM_DATE structure will be set to zero if the date is unspecified or unknown. The CSP module does not enforce this date.

Params
A data buffer containing parameters required to generate a key pair for a specific algorithm.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred and OCSF was unable to create the context. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_GenerateKey
CSSM_GenerateKeyPair
CSSM_GetContext
CSSM_SetContext
CSSM_DeleteContext
CSSM_GetContextAttribute
CSSM_UpdateContextAttributes

CSSM_CSP_CreateMacContext

Purpose

On z/OS, when any CSSM_CSP_CreateMacContext operation is invoked, a copy of the context is created. The pointer to the copy is returned on all CSSM_GetContext calls.

This function creates a Message Authentication Code (MAC) cryptographic context and returns a handle to the context. The cryptographic context handle can be used to call MAC functions. Note that MAC contexts that use RC2 require an effective key size in bits attribute. To add this attribute, use CSSM_UpdateContextAttributes.

Format

```
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateMacContext(CSSM_CSP_HANDLE CSPHandle,
                                                 uint32 AlgorithmID,
                                                 const CSSM_KEY_PTR Key)
```

Parameters

Input

CSPHandle
The handle that describes the CSP module used to perform this function. If a NULL handle is specified, OCSF returns an error.

AlgorithmID
The algorithm identification number for the MAC algorithm.

Key
The key used to generate a MAC. The caller passes in a pointer to a CSSM_KEY structure containing the key.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

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

Purpose

On z/OS, when any CSSM_CSP_CreatePassThroughContext operation is invoked, a copy of the context is created. The pointer to the copy is returned on all CSSM_GetContext calls.

This function creates a custom cryptographic context and returns a handle to the context. The cryptographic context handle can be used to call the CSSM_CSP_PassThrough function for the CSP.

Format

```
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreatePassThroughContext
                (CSSM_CSP_HANDLE CSPHandle,
                const CSSM_KEY_PTR Key,
                const CSSM_DATA_PTR ParamBufs,
                uint32 ParamBufCount)
```

Parameters

Input

CSPHandle
The handle that describes the CSP module used to perform this function. If a NULL handle is specified, OCSF returns an error.

Key
The key to be used for the context. The caller passes in a pointer to a CSSM_KEY structure containing the key.

ParamBufs
Array of input buffers to the passthrough call.

ParamBufCount
The number of input buffers pointed to by ParamBufs.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

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 along with the function ID to call the desired custom function.

Related Information

CSSM_CSP_PassThrough
CSSM_GetContext
CSSM_SetContext
CSSM_DeleteContext
CSSM_GetContextAttribute
CSSM_UpdateContextAttributes

CSSM_CSP_CreateRandomGenContext

Purpose

On z/OS, when any CSSM_CSP_CreateRandomGenContext operation is invoked, a copy of the context is created. The pointer to the copy is returned on all CSSM_GetContext calls.

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

Format

```
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateRandomGenContext
                (CSSM_CSP_HANDLE CSPHandle,
                uint32 AlgorithmID,
                const CSSM_CRYPTO_DATA_PTR Seed,
                uint32 Length))
```

Parameters

Input

CSPHandle
The handle that describes the CSP module used to perform this function. If a NULL handle is specified, OCSF returns an error.

AlgorithmID
The algorithm identification number for random number generation.

Length
The length of the random number to be generated.

Input/optional

Seed
A seed used to generate a random number. The caller can either pass a seed or seed length in bytes or pass in a callback function. If NULL is passed, the CSP will use its default seed handling mechanism.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_GenerateRandom
CSSM_GetContext
CSSM_SetContext
CSSM_DeleteContext
CSSM_GetContextAttribute
CSSM_UpdateContextAttributes

CSSM_CSP_CreateSignatureContext

Purpose

On z/OS, when any CSSM_CSP_CreateSignatureContext operation is invoked, a copy of the context is created. The pointer to the copy is returned on all CSSM_GetContext calls.

This function creates a signature cryptographic context for sign and verify operations given a handle of a CSP, an algorithm identification number, a passphrase structure, and a key. The passphrase 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.

Format

```
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateSignatureContext
                  (CSSM_CSP_HANDLE CSPHandle,
                  uint32 AlgorithmID,
                  const CSSM_CRYPTO_DATA_PTR PassPhrase,
                  const CSSM_KEY_PTR Key)
```

Parameters

Input

CSPHandle
The handle that describes the CSP module used to perform this function. If a NULL handle is specified, OCSF returns an error.

AlgorithmID
The algorithm identification number for a signature/verification algorithm.

PassPhrase
The passphrase is required to unlock the private key. The passphrase structure accepts an immediate value for the passphrase or the caller can specify a callback function the CSP can use to obtain the passphrase. The passphrase is needed only for signature operations, not verify operations.

Key
The key used to sign. The caller passes in a pointer to a CSSM_KEY structure containing the key and the key length.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

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

Purpose

On z/OS, when any CSSM_CSP_CreateSymmetricContext operation is invoked, a copy of the context is created. The pointer to the copy is returned on all CSSM_GetContext calls.

This function creates a symmetric encryption cryptographic context and returns a handle to the context. The cryptographic context handle can be used to call symmetric encryption functions and the cryptographic wrap/unwrap functions.

Format

```
CSSM_CC_HANDLE CSSMAPI CSSM_CSP_CreateSymmetricContext
                 (CSSM_CSP_HANDLE CSPHandle,
                  uint32 AlgorithmID,
                  uint32 Mode,
                  const CSSM_KEY_PTR Key,
                  const CSSM_DATA_PTR InitVector,
                  uint32 Padding,
                  uint32 Params)
```

Parameters

Input

CSPHandle
The handle that describes the CSP module used to perform this function. If a NULL handle is specified, OCSF returns an error.

AlgorithmID
The algorithm identification number for symmetric encryption.

Mode
The mode of the specified algorithm ID.

Key
The key used for symmetric encryption. The caller passes in a pointer to a CSSM_KEY structure containing the key. This key can be used directly for wrap and unwrap operations.

Input/optional

InitVector
The initial vector for symmetric encryption; typically specified for block ciphers.

Padding
The method for padding; typically specified for ciphers that pad.

Params
Specifies the number of rounds of encryption; used for ciphers with variable number of rounds, such as RC5. For ciphers such as RC2, this parameter specifies the effective key size in bits.

Return Value

Returns a cryptographic context handle. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_EncryptData
CSSM_QuerySize
CSSM_EncryptDataInit
CSSM_EncryptDataUpdate
CSSM_EncryptDataFinal
CSSM_DecryptData
CSSM_DecryptDataInit
CSSM_DecryptDataUpdate
CSSM_DecryptDataFinal
CSSM_GetContext
CSSM_SetContext
CSSM_DeleteContext
CSSM_GetContextAttribute
CSSM_UpdateContextAttributes

CSSM_DeleteContext

Purpose

This function frees the context structure allocated by any of the create context functions. On z/OS, this also deletes the context copy that is returned by a CSSM_GetContext call.

Format

CSSM_RETURN CSSMAPI CSSM_DeleteContext (CSSM_CC_HANDLE *CCHandle*)

Parameters

Input

CCHandle
The handle associated with the context to be deleted.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CSP_CreateKeyGenContext
CSSM_CSP_CreateDigestContext
CSSM_CSP_CreateSymmetricContext
CSSM_CSP_CreateAsymmetricContext
CSSM_CSP_CreateSignatureContext

CSSM_FreeContext

Purpose

On z/OS this API should be issued, but no processing is done. On z/OS, a copy of the context is created during CSSM_Create...Context calls. The memory for the context copy is freed during CSSM_DeleteContext.

Format

CSSM_RETURN CSSMAPI CSSM_FreeContext (CSSM_CONTEXT_PTR *Context*)

Parameters

Input

Context
The pointer to the memory that describes the context structure.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_GetContext

CSSM_GetContext

Purpose

This function retrieves the context information when provided with a context handle. A pointer to the context copy, created during one of the CSSM_CSP_Create...Context calls, is returned on all calls.

Format

```
CSSM_CONTEXT_PTR CSSMAPI CSSM_GetContext (CSSM_CC_HANDLE CCHandle)
```

Parameters

Input

CCHandle
The handle to the context information.

Return Value

The pointer to the CSSM_CONTEXT structure that describes the context associated with the handle CCHandle. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code. Call CSSM_FreeContext to free the memory allocated by OCSF.

Related Information

CSSM_SetContext
CSSM_FreeContext

CSSM_GetContextAttribute

Purpose

This function retrieves the context attributes information for the given context and attribute type. Note that not all context attributes can be queried using this function. For example, key size cannot be queried. To determine the key size, query the key. The key size data is contained in the header of the key. These attribute types can be retrieved using CSSM_GetContextAttribute:

Format

```
CSSM_CONTEXT_ATTRIBUTE_PTR CSSMAPI CSSM_GetContextAttribute (const CSSM_CONTEXT_PTR Context,
                                                             uint32 AttributeType)
```

• custom	• key	• output size	• &tab;start date&tab;	• padding
• CSP handle	• key length&tab;	• seed	• end date	• random
• passphrase	• keytype	• rounds	• remote KR profile	• mode
• effective bits	• key attributes	• salt	• local KR profile	• &tab;algorithm parameters
• initialization vector				

Parameters

Input

Context
A pointer to the context.

AttributeType
The attribute type of the specified context.

Return Value

The pointer to the CSSM_ATTRIBUTE structure that describes the context attributes associated with the context and the attribute type. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_GetContext

CSSM_UpdateContextAttribute

Purpose

This function updates the security context. When an attribute is already present in the context, this update operation replaces the previously defined attribute with the current attribute. On z/OS, this call can be made only when no other thread is using the original context or the copy returned by CSSM_GetContext.

Format

```
CSSM_RETURN CSSMAPI CSSM_UpdateContextAttributes
              (CSSM_CC_HANDLE CCHandle,
              uint32 NumberAttributes,
              const CSSM_CONTEXT_ATTRIBUTE_PTR ContextAttributes)
```

Parameters

Input

CCHandle
The handle to the context.

NumberAttributes
The number of CSSM_CONTEXT_ATTRIBUTE &tab;structures to allocate.

ContextAttributes
Pointer to data that describes the attributes to be associated with this context.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_GetContextAttribute

Cryptographic Sessions and Login

The interfaces discussed here support a password based login sequence.

CSSM_CSP_ChangeLoginPassword

Purpose

Changes the login password of the current login session from the old password to the new password. The requesting user must have a login session in process.

Format

```
CSSM_RETURN CSSMAPI CSSM_CSP_ChangeLoginPassword
               (CSSM_CSP_HANDLE CSPHandle,
                const CSSM_CRYPTO_DATA_PTR OldPassword,
                const CSSM_CRYPTO_DATA_PTR NewPassword)
```

Parameters

CSPHandle
Handle of the CSP supporting the current login session.

OldPassword
Current password used to log into the token.

NewPassword
New password to be used for future logins by this user to this token.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_CSP_Login
CSSM_CSP_Logout

CSSM_CSP_Login

Purpose

Logs the user into the CSP, allowing for multiple login types and parallel operation notification.

Format

```
CSSM_RETURN CSSMAPI CSSM_CSP_Login
               (CSSM_CSP_HANDLE CSPHandle,
               const CSSM_CRYPTO_DATA_PTR Password,
               const CSSM_DATA_PTR pReserved)
```

Parameters

Input

CSPHandle
Handle of the CSP to log into.

Password
Password used to log into the token.

pReserved
This field is reserved for future use. The value NULL should always be given.

Return Value

CSSM_OK if login is successful, CSSM_FAIL is login fails. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_CSP_ChangeLoginPassword
CSSM_CSP_Logout

CSSM_CSP_Logout

Purpose

Terminates the login session associated with the specified CSP Handle.

Format

```
CSSM_RETURN CSSMAPI CSSM_CSP_Logout (CSSM_CSP_HANDLE CSPHandle)
```

Parameters

Input

CSPHandle
Handle for the target CSP.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_CSP_Login
CSSM_CSP_ChangeLoginPassword

Cryptrographic Operations

The interfaces discussed here provide for cryptographic operations including encryption, decryption, digital signaturing, key and key pair generation, random number generation, message digest, key wrapping, key unwrapping, and key exchange.

CSSM_DecryptData

Purpose

This function decrypts the supplied encrypted data. The CSSM_QuerySize function can be used to estimate the output buffer size required. When working with U.S. exportable versions of the OCSF, the caller may be required to possess specific exemptions or privileges in order to allow this call to complete successfully.

Format

```
CSSM_RETURN CSSMAPI CSSM_DecryptData
              (const CSSM_CC_HANDLE CCHandle,
              const CSSM_DATA_PTR CipherBufs,
              uint32 CipherBufCount,
              CSSM_DATA_PTR ClearBufs,
              uint32 ClearBufCount,
              uint32 *bytesDecrypted,
              CSSM_DATA_PTR RemData)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

CipherBufs
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

CipherBufCount
The number of *CipherBufs*.

ClearBufCount
The number of *ClearBufs*.

Output

ClearBufs
A pointer to a vector of CSSM_DATA structures that contain the decrypted data resulting from the decryption operation.

BytesDecrypted
The size of the decrypted data in bytes.

RemData
A pointer to the CSSM_DATA structure for the last decrypted block.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space; the application has to free the memory in this case. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned. In-place decryption can be done by supplying the same input and output buffer.

Related Information

CSSM_QuerySize
CSSM_EncryptData
CSSM_DecryptDataInit
CSSM_DecryptDataUpdate
CSSM_DecryptDataFinal
CSSM_RequestCssmExemption
CSSM_DecryptDataFinal

CSSM_DecryptDataFinal

Purpose

This function finalizes the staged decrypt function. When working with U.S. exportable versions of the OCSF, the caller may be required to possess specific exemptions or privileges in order to allow this call to complete successfully.

Format

```
CSSM_RETURN CSSMAPI CSSM_DecryptDataFinal (CSSM_CC_HANDLE CCHandle, CSSM_DATA_PTR RemData)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Output

RemData
A pointer to the CSSM_DATA structure for the last decrypted block.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space, which the application must later free . If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned. In-place decryption can be done by supplying the same input and output buffers.

Related Information

CSSM_DecryptData
CSSM_DecryptDataInit
CSSM_DecryptDataUpdate
CSSM_Request_CssmExemption

CSSM_DecryptDataInit

Purpose

This function initializes the staged decrypt function. When working with U.S. exportable versions of the OCSF, the caller may be required to possess specific exemptions or privileges in order to allow this call to complete successfully.

Format

```
CSSM_RETURN CSSMAPI CSSM_DecryptDataInit (CSSM_CC_HANDLE CCHandle)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_DecryptData
CSSM_DecryptDataUpdate
CSSM_DecryptDataFinal
CSSM_RequestCssmExemption

CSSM_DecryptDataUpdate

Purpose

This function updates the staged decrypt function. The CSSM_QuerySize function can be used to estimate the output buffer size required for each update call. There may be algorithm-specific and token-specific rules restricting the lengths of data in CSSM_DecryptDataUpdate calls. When working with U.S. exportable versions of the OCSF, the caller may be required to possess specific exemptions or privileges in order to allow this call to complete successfully.

Format

```
CSSM_RETURN CSSMAPI CSSM_DecryptDataUpdate
             (CSSM_CC_HANDLE CCHandle,
             const CSSM_DATA_PTR CipherBufs,
             uint32 CipherBufCount,
             CSSM_DATA_PTR ClearBufs,
             uint32 ClearBufCount,
             uint32 *bytesDecrypted)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

CipherBufs
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

CipherBufCount
The number of *CipherBufs*.

ClearBufCount
The number of *ClearBufs*.

Output

bytesDecrypted
A pointer to uint32 for the size of the decrypted data in bytes.

ClearBufs
A pointer to a vector of CSSM_DATA structures that contain the decrypted data resulting from the decryption operation.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space, which the application must later free. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned. In-place decryption can be done by supplying the same input and output buffers.

Related Information

CSSM_DecryptData
CSSM_DecryptDataInit
CSSM_DecryptDataFinal
CSSM_QuerySize
CSSM_RequestCssmExemption

CSSM_DeriveKey

Purpose

This function derives a new asymmetric key using the context and information from the base key.

Format

```
CSSM_RETURN CSSMAPI CSSM_DeriveKey
              (CSSM_CC_HANDLE CCHandle,
              const CSSM_KEY_PTR BaseKey,
              void *Param,
              uint32 KeyUsage,
              uint32 KeyAttr,
              const CSSM_DATA_PTR KeyLabel,
              CSSM_KEY_PTR DerivedKey)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation.

BaseKey
The base key used to derive the new key. The base key may be a public key, a private key, or an asymmetric key.

KeyUsage
A bit-mask representing the valid uses of the key. See Table 36 on page 125 for a list of valid values.

KeyAttr
&tab;A bit-mask representing the attributes of the key represented by the data. These attributes are used by CSP service providers to convey information about stored or referenced keys.

Output

DerivedKey
A pointer to a CSSM_KEY structure that returns the derived key.

Input/optional

KeyLabel
Pointer to a byte string that will be used as the label for the derived key.

Input/Output

Param
The use of this parameter varies depending on the derivation algorithms. Specific algorithms use Params to pass custom data to algorithms.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The KeyData field of the CSSM_KEY structure is not required to be allocated. In this case, the memory required to represent the key is allocated by the CSP. The application is required to free this memory. The CSP will only allocate memory if the Data field of KeyData is NULL and the Length field is zero.

Related Information

CSSM_CSP_CreateDeriveKeyContext

CSSM_DigestData

Purpose

This function computes a message digest for the supplied data.

Format

```
CSSM_RETURN CSSMAPI CSSM_DigestData
              (CSSM_CC_HANDLE CCHandle,
              const CSSM_DATA_PTR DataBufs,
              uint32 DataBufCount,
              CSSM_DATA_PTR Digest)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount
The number of *DataBufs*.

Output

Digest
A pointer to the CSSM_DATA structure for the message digest.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space, which the application must later free. If the output buffer pointer this is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

Related Information

CSSM_DigestDataInit
CSSM_DigestDataUpdate
CSSM_DigestDataFinal

CSSM_DigestDataClone

Purpose

This function clones a given staged message digest context with its cryptographic attributes and intermediate result.

Format

```
CSSM_CC_HANDLE CSSMAPI CSSM_DigestDataClone (CSSM_CC_HANDLE CCHandle)
```

Parameters

Input

CCHandle
The handle that describes the context of a staged message digest operation.

Return Value

The handle of cloned context. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

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.

Related Information

CSSM_DigestData
CSSM_DigestDataInit
CSSM_DigestDataUpdate
CSSM_DigestDataFinal

CSSM_DigestDataFinal

Purpose

This function finalizes the staged message digest function.

Format

```
CSSM_RETURN CSSMAPI CSSM_DigestDataFinal (CSSM_CC_HANDLE CCHandle, CSSM_DATA_PTR Digest)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Output

Digest
A pointer to the CSSM_DATA structure for the message digest.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space, which the application must later free. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

Related Information

CSSM_DigestData
CSSM_DigestDataInit
CSSM_DigestDataUpdate
CSSM_DigestDataClone

CSSM_DigestDataInit

Purpose

This function initializes the staged message digest operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_DigestDataInit (CSSM_CC_HANDLE CCHandle)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_DigestData
CSSM_DigestDataUpdate
CSSM_DigestDataClone
CSSM_DigestDataFinal

CSSM_DigestDataUpdate

Purpose

This function updates the staged message digest operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_DigestDataUpdate (CSSM_CC_HANDLE CCHandle,const CSSM_DATA_PTR DataBufs,
                                           uint32 DataBufCount)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount
The number of *DataBufs*.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_DigestData
CSSM_DigestDataInit
CSSM_DigestDataClone
CSSM_DigestDataFinal

CSSM_EncryptData

Purpose

This function encrypts the supplied data using information in the context. The CSSM_QuerySize function can be used to estimate the output buffer size required. When working with U.S. exportable versions of the OCSF, the caller may be required to possess specific exemptions or privileges in order to allow this call to complete successfully.

Format

```
CSSM_RETURN CSSMAPI CSSM_EncryptData
              (CSSM_CC_HANDLE CCHandle,
              const CSSM_DATA_PTR ClearBufs,
              uint32 ClearBufCount,
              CSSM_DATA_PTR CipherBufs,
              uint32 CipherBufCount,
              uint32 *bytesEncrypted,
              CSSM_DATA_PTR RemData)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

ClearBufs
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

ClearBufCount
The number of *ClearBufs*.

CipherBufCount
The number of *CipherBufs*.

Output

CipherBufs
A pointer to a vector of CSSM_DATA structures that contain the results of the operation on the data.

bytesEncrypted
The size of the encrypted data in bytes.

RemData
A pointer to the CSSM_DATA structure for the last encrypted block containing padded data.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space, which the application must later free. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned. In-place encryption can be done by supplying the same input and output buffers.

Related Information

CSSM_QuerySize
CSSM_DecryptData
CSSM_EncryptDataInit

CSSM_EncryptDataUpdate
CSSM_EncryptDataFinal
CSSM_RequestCssmExemption

CSSM_EncryptDataFinal

Purpose

This function finalizes the staged encrypt operation. When working with U.S. exportable versions of the OCSF, the caller may be required to possess specific exemptions or privileges in order to allow this call to complete successfully.

Format

```
CSSM_RETURN CSSMAPI CSSM_EncryptDataFinal(CSSM_CC_HANDLE CCHandle, CSSM_DATA_PTR RemData)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Output

RemData
A pointer to the CSSM_DATA structure for the last encrypted block containing padded data.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space, which the application must later free. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned. In-place encryption can be done by supplying the same input and output buffers.

Related Information

CSSM_EncryptData
CSSM_EncryptDataInit
CSSM_EncryptDataUpdate
CSSM_RequestCssmExemption

CSSM_EncryptDataInit

Purpose

This function initializes the staged encrypt operation. There may be algorithm-specific and token-specific rules restricting the lengths of data in the CSSM_EncryptDataUpdate calls that make use of these parameters. When working with U.S. exportable versions of the OCSF, the caller may be required to possess

specific exemptions or privileges in order to allow this call to complete successfully.

Format

```
CSSM_RETURN CSSMAPI CSSM_EncryptDataInit (CSSM_CC_HANDLE CCHandle)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_EncryptData
CSSM_EncryptDataUpdate
CSSM_EncryptDataFinal
CSSM_RequestCssmExemption

CSSM_EncryptDataUpdate

Purpose

This function updates the staged encrypt operation. The CSSM_QuerySize function can be used to estimate the output buffer size required for each update call. There may be algorithm-specific and token-specific rules restricting the lengths of data in CSSM_EncryptDataUpdate calls. When working with U.S. exportable versions of the OCSF, the caller may be required to possess specific exemptions or privileges in order to allow this call to complete successfully.

Format

```
CSSM_RETURN CSSMAPI CSSM_EncryptDataUpdate
                    (CSSM_CC_HANDLE CCHandle,
                    const CSSM_DATA_PTR ClearBufs,
                    uint32 ClearBufCount,
                    CSSM_DATA_PTR CipherBufs,
                    uint32 CipherBufCount,
                    uint32 *bytesEncrypted)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

ClearBufs
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

ClearBufCount
The number of *ClearBufs*.

CipherBufCount
The number of *CipherBufs*.

Output

CipherBufs
A pointer to a vector of CSSM_DATA structures that contain the encrypted data resulting from the encryption operation.

bytesEncrypted
The size of the encrypted data in bytes.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space, which the application must later free. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned. In-place encryption can be done by supplying the same input and output buffers.

Related Information

CSSM_EncryptData
CSSM_EncryptDataInit
CSSM_EncryptDataFinal
CSSM_QuerySize
CSSMRequestCssmExemption

CSSM_GenerateAlgorithmParams

Purpose

This function generates algorithm parameters for the specified context. These parameters include Diffie-Hellman key agreement parameters and DSA key generation parameters.

Format

```
CSSM_RETURN CSSMAPI CSSM_GenerateAlgorithmParams
              (CSSM_CC_HANDLE CCHandle,
               uint32 ParamBits,
               CSSM_DATA_PTR Param)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

ParamBits
Used to generate parameters for the algorithm (for example, Diffie-Hellman).

Output

Param
Pointer to CSSM_DATA structure used to obtain the key exchange parameter and the size of the key exchange parameter in bytes.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space, which the application must later free. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

CSSM_GenerateKey

Purpose

This function generates a symmetric key.

Format

```
CSSM_RETURN CSSMAPI CSSM_GenerateKey
              (CSSM_CC_HANDLE CCHandle,
              uint32 KeyUsage,
              uint32 KeyAttr,
              const CSSM_DATA_PTR KeyLabel,
              CSSM_KEY_PTR Key)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

KeyUsage
A bit-mask representing the valid uses of the key. See Table 36 on page 125 for a list of valid values.

KeyAttr
&tab;A bit-mask representing the attributes of the key represented by the data. These attributes are used by CSP service providers to convey information about stored or referenced keys.

Output

Key
Pointer to CSSM_KEY structure containing the key.

Input/optional

KeyLabel
Pointer to a byte string that will be used as a label/identifier for the derived key. If a key label is not used, this field should be set to NULL.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The *KeyData* field of the CSSM_KEY structure is not required to be allocated. In this case, the memory required to represent the key is allocated by the CSP. The application is required to free this memory. The CSP will only allocate memory if the *Data* field of *KeyData* is NULL and the *Length* field is zero.

Related Information

CSSM_GenerateRandom
CSSM_GenerateKeyPair

CSSM_GenerateKeyPair

Purpose

This function generates an asymmetric key pair.

Format

```
CSSM_RETURN CSSMAPI CSSM_GenerateKeyPair
              (CSSM_CC_HANDLE CCHandle,
               uint32 PublicKeyUsage
               uint32 PublicKeyAttr,
               const CSSM_DATA_PTR PublicKeyLabel,
               CSSM_KEY_PTR PublicKey,
               uint32 PrivateKeyUsage,
               uint32 PrivateKeyAttr,
               const CSSM_DATA_PTR PrivateKeyLabel,
               CSSM_KEY_PTR PrivateKey)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Output

PublicKey
Pointer to CSSM_KEY structure used to obtain the public key.

PrivateKey
Pointer to CSSM_KEY structure used to obtain the private key.

Input/optional

PublicKeyUsage
A bit-mask representing the valid uses of the public key. This field may be required by some CSP modules. Refer to the information provided with the CSP for more information. See Table 36 on page 125 for a list of valid key usage values.

PublicKeyAttr
A bit-mask representing the attributes of the public key represented by the data. These attributes are used by CSP service providers to convey information

about stored or referenced keys. This field may be required by some CSP modules. Refer to the information provided with the CSP for more information.

PublicKeyLabel
Pointer to a byte string that will be used as a label/identifier for the derived public key. If a key label is not used, this field should be set to NULL.

PrivateKeyUsage
A bit-mask representing the valid uses of the private key. This field may be required by some CSP modules. For more information, see the information provided with the CSP from the module vendor. See Table 36 on page 125 for a list of valid key usage values.

PrivateKeyAttr
A bit-mask representing the attributes of the private key represented by the data. These attributes are used by CSP service providers to convey information about stored or referenced keys. This field may be required by some CSP modules. Refer to the information provided with the CSP for more information.

PrivateKeyLabel
Pointer to a byte string that will be used as a label/identifier for the derived private key. If a key label is not used, this field should be set to NULL.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The *KeyData* field of the CSSM_KEY structures are not required to be allocated. In this case, the memory required to represent the key is allocated by the CSP. The application is required to free this memory. The CSP will only allocate memory if the *Data* field of *KeyData* is NULL and the *Length* field is zero.

Related Information

CSSM_GenerateRandom

CSSM_GenerateMac

Purpose

This function generates a message authentication code for the supplied data.

Format

```
CSSM_RETURN CSSMAPI CSSM_GenerateMac (CSSM_CC_HANDLE CCHandle,
                                      const CSSM_DATA_PTR DataBufs,
                                      uint32 DataBufCount,
                                      CSSM_DATA_PTR Mac)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount
The number of *DataBufs*.

Output

Mac
A pointer to the CSSM_DATA structure containing the message authentication code.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space, which the application must later free. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

Related Information

CSSM_GenerateMacInit
CSSM_GenerateMacUpdate
CSSM_Generate MacFinal

CSSM_GenerateMacFinal

Purpose

This function finalizes the staged message authentication code operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_GenerateMacFinal (CSSM_CC_HANDLE CCHandle, CSSM_DATA_PTR Mac)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Mac
A pointer to the CSSM_DATA structure containing the message authentication code.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space, which the application must later free. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

Related Information

CSSM_GenerateMac
CSSM_GenerateMacInit
CSSM_GenerateMacUpdate

CSSM_GenerateMacInit

Purpose

This function initializes the staged message authentication code operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_GenerateMacInit (CSSM_CC_HANDLE CCHandle)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_GenerateMac
CSSM_GenerateMacUpdate
CSSM_GenerateMacFinal

CSSM_GenerateMacUpdate

Purpose

This function updates the staged message authentication code operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_GenerateMacUpdate (CSSM_CC_HANDLE CCHandle,
                                            const CSSM_DATA_PTR DataBufs,
                                            uint32 DataBufCount)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount
The number of *DataBufs*.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_GenerateMac
CSSM_GenerateMacInit
CSSM_GenerateMacFinal

CSSM_GenerateRandom

Purpose

This function generates random data.

Format

```
CSSM_RETURN CSSMAPI CSSM_GenerateRandom (CSSM_CC_HANDLE CCHandle,CSSM_DATA_PTR RandomNumber)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Output

RandomNumber
Pointer to CSSM_DATA structure used to obtain the random number and the size of the random number in bytes.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space, which the application must later free. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

CSSM_QueryKeySizeInBits

Purpose

This function queries a CSP for the effective and real size of a key in bits.

Format

```
CSSM_RETURN CSSMAPI CSSM_QueryKeySizeInBits
              (CSSM_CSP_HANDLE CSPHandle,
              CSSM_CC_HANDLE CCHandle,
              CSSM_KEY_SIZE_PTR KeySize)
```

Parameters

Input

CSPHandle
The handle that describes the CSP module used to perform this function.

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Output

KeySize
Pointer to a CSSM_KEY_SIZE data structure returns the actual size and the effective size of the key in bits.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_GenerateRandom
CSSM_GenerateKeyPair

CSSM_QuerySize

Purpose

This function queries for the size of the output data for Signature, Message Digest, and Message Authentication Code context types and queries for the algorithm block size, or the size of the output data for encryption and decryption context types. This function also can be used to query the output size requirements for the intermediate steps of a staged cryptographic operation (for example, CSSM_EncryptDataUpdate and CSSM_DecryptDataUpdate). There may be algorithm-specific and token-specific rules restricting the lengths of data in these data update calls.

Format

```
CSSM_RETURN CSSMAPI CSSM_QuerySize
             (CSSM_CC_HANDLE CCHandle,
             CSSM_BOOL Encrypt,
             uint32 QuerySizeCount,
             CSSM_QUERY_SIZE_DATA_PTR DataBlock)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Encrypt
When asymmetric and symmetric contexts are being used, *Encrypt* indicates whether an encryption (CSSM_TRUE) or a decryption (CSSM_FALSE) operation will be performed. For all other operations and context types, *Encrypt* should be set to CSSM_FALSE.

QuerySizeCount
An integer that indicates the number of data blocks that are in DataBlock.

Input/Output

DataBlock
&tab;A pointer to a CSSM_QUERY_SIZE_DATA structure that contains the size of the input and the output data blocks, in bytes.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_EncryptData
CSSM_EncryptDataUpdate
CSSM_DecryptData
CSSM_DecryptDataUpdate
CSSM_SignData
CSSM_VerifyData
CSSM_DigestData
CSSM_GenerateMac

CSSM_SignData

Purpose

This function signs data using the private key associated with the public key specified in the context.

Format

```
CSSM_RETURN CSSMAPI CSSM_SignData
              (CSSM_CC_HANDLE CCHandle,
              const CSSM_DATA_PTR DataBufs,
              uint32 DataBufCount,
              CSSM_DATA_PTR Signature)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount
The number of DataBufs to be signed.

Output

Signature
A pointer to the CSSM_DATA structure containing the signature.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space, which the application must later free. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

Related Information

CSSM_VerifyData
CSSM_SignDataInit
CSSM_SignDataUpdate
CSSM_SignDataFinal

CSSM_SignDataFinal

Purpose

This function completes the final stage of the sign data operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_SignDataFinal (CSSM_CC_HANDLE CCHandle, CSSM_DATA_PTR Signature)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Output

Signature
A pointer to the CSSM_DATA structure for the signature.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The output can be obtained either by filling the caller-supplied buffer or using the application's memory allocation functions to allocate space, which the application must later free. If the output buffer pointer is NULL, an error code CSSM_CSP_INVALID_DATA_POINTER is returned.

Related Information

CSSM_SignData
CSSM_SignDataInit
CSSM_SignDataUpdate

CSSM_SignDataInit

Purpose

This function initializes the staged sign data operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_SignDataInit (CSSM_CC_HANDLE CCHandle)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_SignData
CSSM_SignDataUpdate
CSSM_SignDataFinal

CSSM_SignDataUpdate

Purpose

This function updates the data for the staged sign data operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_SignDataUpdate
             (CSSM_CC_HANDLE CCHandle,
             const CSSM_DATA_PTR DataBufs,
             uint32 DataBufCount)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount
The number of *DataBufs* to be signed.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_SignData
CSSM_SignDataInit
CSSM_SignDataFinal

CSSM_UnwrapKey

Purpose

This function unwraps the data using the context. When working with U.S. exportable versions of the OCSF, the caller may be required to possess specific exemptions or privileges in order to allow this call to complete successfully.

Format

```
CSSM_RETURN CSSMAPI CSSM_UnwrapKey
              (CSSM_CC_HANDLE CCHandle,
              const CSSM_CRYPTO_DATA_PTR NewPassPhrase,
              const CSSM_WRAP_KEY_PTR WrappedKey,
              uint32 KeyAttr,
              const CSSM_DATA_PTR KeyLabel,
              CSSM_KEY_PTR UnwrappedKey)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation.

NewPassPhrase
The passphrase or a callback function to be used to obtain the passphrase. If the unwrapped key is a private key and the persistent object mode is true, then the private key is unwrapped and securely stored by the CSP. The NewPassPhrase is used to secure the private key after it is unwrapped. It is assumed that a known public key is associated with the private key.

WrappedKey
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.

KeyAttr
Attribute the unwrapped key will assume.

Output

UnwrappedKey
A pointer to a CSSM_KEY structure that returns the unwrapped key.

Input/optional

KeyLabel
Pointer to a byte string that will be used as the label for the unwrapped key.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Notes

The *KeyData* field of the CSSM_KEY structure is not required to be allocated. In this case, the memory required to represent the key is allocated by the CSP. The application is required to free this memory. The CSP will only allocate memory if the *Data* field of *KeyData* is NULL and the *Length* field is zero.

Related Information

CSSM_WrapKey
CSSM_RequestCssmExemption

CSSM_VerifyData

Purpose

This function verifies the input data against the provided signature.

Format

```
CSSM_BOOL CSSMAPI CSSM_VerifyData
             (CSSM_CC_HANDLE CCHandle,
             const CSSM_DATA_PTR DataBufs,
             uint32 DataBufCount,
             const CSSM_DATA_PTR Signature)
```

Parameters

Input

CCHandle
&tab;The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount
The number of *DataBufs* to be verified.

Signature
A pointer to a CSSM_DATA structure which contains the signature and the size of the signature.

Return Value

A CSSM_TRUE return value signifies the signature was successfully verified. When CSSM_FALSE is returned, either the signature was not successfully verified or an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_SignData
CSSM_VerifyDataInit
CSSM_VerifyDataUpdate
CSSM_VerifyDataFinal

CSSM_VerifyDataFinal

Purpose

This function finalizes the staged verify data function.

Format

```
CSSM_BOOL CSSMAPI CSSM_VerifyDataFinal (CSSM_CC_HANDLE CCHandle,const CSSM_DATA_PTR Signature)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Signature
A pointer to a CSSM_DATA structure that contains the starting address for the signature to verify against and the length of the signature in bytes.

Return Value

A CSSM_TRUE return value signifies the signature was successfully verified. When CSSM_FALSE is returned, either the signature was not successfully verified or an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_VerifyData
CSSM_VerifyDataInit
CSSM_VerifyDataUpdate

CSSM_VerifyDataInit

Purpose

This function initializes the staged verify data operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_VerifyDataInit (CSSM_CC_HANDLE CCHandle)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_VerifyDataUpdate
CSSM_VerifyDataFinal
CSSM_VerifyData

CSSM_VerifyDataUpdate

Purpose

This function updates the data to the staged verify data operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_VerifyDataUpdate
              (CSSM_CC_HANDLE CCHandle,
              const CSSM_DATA_PTR DataBufs,
              uint32 DataBufCount)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount
The number of *DataBufs* to be verified.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_VerifyData
CSSM_VerifyDataInit
CSSM_VerifyDataFinal

CSSM_VerifyMac

Purpose

This function verifies a message authentication code for the supplied data.

Format

```
CSSM_RETURN CSSMAPI CSSM_VerifyMac
               (CSSM_CC_HANDLE CCHandle,
               const CSSM_DATA_PTR DataBufs,
               uint32 DataBufCount,
               CSSM_DATA_PTR Mac)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount
The number of DataBufs.

Mac
A pointer to the CSSM_DATA structure containing the MAC to verify.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_VerifyMacInit
CSSM_VerifyMacUpdate
CSSM_VerifyMacFinal

CSSM_VerifyMacFinal

Purpose

This function finalizes the staged message authentication code verification operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_VerifyMacFinal (CSSM_CC_HANDLE CCHandle, CSSM_DATA_PTR Mac)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Mac
A pointer to the CSSM_DATA structure containing the MAC to verify.

Return Value

CSSM_OK if the MAC verifies correctly, CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_VerifyMac
CSSM_VerifyMacInit
CSSM_VerifyMacUpdate

CSSM_VerifyMacInit

Purpose

This function initializes the staged message authentication code verification operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_VerifyMacInit (CSSM_CC_HANDLE CCHandle)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_VerifyMac
CSSM_VerifyMacUpdate
CSSM_VerifyMacFinal

CSSM_VerifyMacUpdate

Purpose

This function updates the staged message authentication code verification operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_VerifyMacUpdate
             (CSSM_CC_HANDLE CCHandle,
             const CSSM_DATA_PTR DataBufs,
             uint32 DataBufCount)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation used to link to the CSP-managed information.

DataBufs
A pointer to a vector of CSSM_DATA structures that contain the data to be operated on.

DataBufCount
The number of *DataBufs*.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_VerifyMac
CSSM_VerifyMacInit
CSSM_VerifyMacFinal

CSSM_WrapKey

Purpose

This function wraps the supplied key using the context. The key may be a symmetric key or the public key of a public/private key pair. If a symmetric key is specified, it is wrapped. If a public key is specified, the passphrase is used to unlock the corresponding private key, which is then wrapped. When working with U.S. exportable versions of the OCSF, the caller may be required to possess specific exemptions or privileges in order to allow this call to complete successfully.

Format

```
CSSM_RETURN CSSMAPI CSSM_WrapKey
             (CSSM_CC_HANDLE CCHandle,
             const CSSM_CRYPTO_DATA_PTR PassPhrase,
             const CSSM_KEY_PTR Key,
             CSSM_WRAP_KEY_PTR WrappedKey)
```

Parameters

Input

CCHandle
The handle to the context that describes this cryptographic operation.

PassPhrase
The passphrase or a callback function to be used to obtain the passphrase that can be used by the CSP to unlock the private key before it is wrapped. This input is ignored when wrapping a symmetric, secret key.

Key
A pointer to the target key to be wrapped. If a private key is to be wrapped, the target key is the public key associated with the private key. If a symmetric key is to be wrapped, the target key is that symmetric key.

Output

WrappedKey
A pointer to a CSSM_WRAP_KEY_PTR structure that returns the wrapped key.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_UnwrapKey
CSSM_RequestCssmExemption

Extensibility Functions

The CSSM_CSP_PassThrough function allows CSP developers to extend the cryptographic functionality of the OCSF API. Because it is only exposed to OCSF as a function pointer, its name internal to the CSP can be assigned at the discretion of the CSP module developer. However, its parameter list and return value must match what is shown in this function.

CSSM_CSP_PassThrough

Format

```
void * CSSMAPI CSSM_CSP_PassThrough
          (CSSM_CC_HANDLE CCHandle,
          uint32 PassThroughId,
          const void *InData)
```

Parameters

Input

CCHandle
The handle that describes the context of this cryptographic operation.

PassThroughId
An identifier specifying the custom function to be performed.

InData
A pointer to a module-specific structure containing the input data.

Return Value

A pointer to a module-specific structure containing the output data. If successful, this function returns a non-NULL value. A NULL value indicates that an error has occurred. Use CSSM_GetError to obtain a specific error code.

Chapter 13. Key Recovery Services API

The Key Recovery interfaces are **not** supported in the OCSF, with the exception of the CSSM_KR_QueryPolicyInfo. The interfaces can be compiled into an application for compatibility purposes with other implementations. However, the functions are not available. Key Recovery contexts can be created, but are of no use.

Data Structures

This discusses the Key Recovery data structures.

Note: Some application interfaces use data structures defined by other OCSF services. Those data structures are defined with those particular OCSF services.

CSSM_CERTGROUP

This structure contains a set of certificates. It is assumed that the certificates are related based on cosignaturing. The certificate group is a syntactic representation of a trust model. All certificates in the group must be of the same type. Typically, the certificates are related in some manner, but this is not required.

```
typedef struct cssm_certgroup {
    uint32 NumCerts;
    CSSM_DATA_PTR CertList;
    void *reserved;
} CSSM_CERTGROUP, *CSSM_CERTGROUP_PTR;
```

Definitions:

NumCerts
: Number of certificates in the group.

CertList
: List of certificates.

Reserved
: Reserved for future use.

CSSM_CONTEXT_ATTRIBUTE Extensions

The key recovery, context creation operations return key recovery context handles that are represented as cryptographic context handles. The CSSM_CONTEXT data structure has been extended to include the new types of attributes, as shown in the example:

```
typedef struct cssm_context_attribute {
    uint32 AttributeType;/* one of the defined CSSM_ATTRIBUTE_TYPEs */
    uint32 AttributeLength;/* length of attribute */
    union {
        uint8 *Description; uint32 *Length;
        void *Pointer;
        CSSM_CRYPTO_DATA_PTR SeedPassPhrase;
        CSSM_KEY_PTR Key;
        CSSM_DATA_PTR Data;
        CSSM_KR_PROFILE_PTR KRProfile;/*new attribute to hold KR profile*/
    } Attribute; /* data that describes attribute */
} CSSM_CONTEXT_ATTRIBUTE, *CSSM_CONTEXT_ATTRIBUTE_PTR;
```

Several new attribute types were defined to support the key recovery context attributes. The CSSM_ATTRIBUTE_TYPE enum is extended as follows:

```
CSSM_ATTRIBUTE_KRPROFILE_LOCAL = CSSM_ATTRIBUTE_LAST + 1,
CSSM_ATTRIBUTE_KRPROFILE_REMOTE= CSSM_ATTRIBUTE_LAST + 2
```

CSSM_KR_LIST_ITEM

The data structure contains the context of one of the entries in a policy module.

```
typedef struct kr_policy_list_item {
    struct kr_policy_list_item  *next;
    uint32              AlgorithmId;
    uint32              Mode;
    uint32              MaxKeyLength;
    uint32              MaxRounds;
    uint8               WorkFactor;
    uint8               PolicyFlags; /* to indicate which jurisdiction
                        required the policy */
    uint32              AlgClass; /* SYMMETRIC versus ASYMMETRIC */
} CSSM_KR_POLICY_LIST_ITEM;
```

CSSM_KR_NAME

This data structure contains a typed name. The namespace type specifies what kind of name is contained in the third parameter.

```
typedef struct cssm_kr_name {
    uint8 Type; /* namespace type */
    uint8 Length; /* name string length */
    char *Name; /* name string */
} CSSM_KR_NAME, *CSSM_KR_NAME_PTR;
```

CSSM_KR_PROFILE

This data structure encapsulates the key recovery profile for a given user and a given key recovery mechanism.

```
typedef struct cssm_kr_profile {
    CSSM_KR_NAME UserName;
    CSSM_DATA_PTR UserCertificate;
    CSSM_CERTGROUP_PTR KRSCertChain;
    uint8 LE_KRANum;
    CSSM_CERTGROUP_PTR LE_KRACertChainList;
    uint8 ENT_KRANum;
    CSSM_CERTGROUP_PTR ENT_KRACertChainList;
    uint8 INDIV_KRANum;
    CSSM_CERTGROUP_PTR INDIV_KRACertChainList;
    CSSM_DATA_PTR INDIV_AuthenticationInfo;
    uint32KRSPFlags;
    CSSM_DATA_PTR KRSPExtensions;
} CSSM_KR_PROFILE, *CSSM_KR_PROFILE_PTR;
```

Definitions:

UserName
: Name of user this profile profiles.

UserCertificate
: PK certificate of user.

KRSCertChain
: Reserved for future use.

LE_KRANum
: &tab;Number of law enforcement cert chains in *LE_KRACertChainList*.

LE_KRACertChainList
: List of certificate chains for law enforcement.

ENT_KRANum
: Number of enterprise cert chain in *ENT_KRACertChainList*.

ENT_KRACertChainList
: List of certificate chains for enterprise.

INDIV_KRANum
: Number of individual cert chains in *INDIV_KRACertChainList*.

INDIV_KRACertChainList
: List of certificate chains for individual key recovery.

INDIV_AuthenticationInfo
: Authentication Information (AI) for user key recovery.

KRSPFlags
Flag values interpreted by Key Recovery Service Provider (KRSP).
KRSPExtensions
Reserved for extensions specific to a key recovery module

CSSM_KRSP_HANDLE

```
typedef uint32 CSSM_KRSP_HANDLE /* Key Recovery Service Provider Handle */
```

CSSM_KRSPSUBSERVICE

Three structures are used to contain all of the static information that describes a KRSP module: cssm_moduleinfo, cssm_serviceinfo, and cssm_krspsubservice. This descriptive information is securely stored in the OCSF registry when the key recovery module is installed with OCSF. A KRSP module may implement multiple types of services and organize them as subservices.

The descriptive information stored in these structures can be queried using the function CSSM_GetModuleInfo and specifying the KRSP module Globally Unique ID (GUID).

```
typedef struct cssm_krspsubservice {
    uint32 SubServiceId;
    CSSM_STRING Description;/* Description of this sub service */
    CSSM_STRING Jurisdiction;
} CSSM_KRSPSUBSERVICE, *CSSM_KRSPSUBSERVICE_PTR;
```

CSSM_KR_WRAPPEDPRODUCTINFO

```
typedef struct cssm_kr_wrappedproductinfo {
    CSSM_VERSION StandardVersion;
    CSSM_STRING StandardDescription;
    CSSM_VERSION ProductVersion;
    CSSM_STRING ProductDescription;
    CSSM_STRING ProductVendor;
    uint32 ProductFlags;
} CSSM_KR_WRAPPEDPRODUCT_INFO, *CSSM_KR_WRAPPEDPRODUCT_INFO_PTR;
```

Definitions:
StandardVersion
Version of standard to which this product conforms.
StandardDescription
Description of standard to which this product conforms.
ProductVersion
Version of wrapped product/library.
ProductDescription
Description of wrapped product/library
ProductVendor
Vendor of wrapped product/library.
ProductFlags
Specifies product flags.

CSSM_POLICY_INFO

This data structure encapsulates policy module information.

```
typedef struct policy_info {
 CSSM_BOOL  krbNotAllowed;
 uint32   numberOfEntries;
 CSSM_KR_POLICY_LIST_ITEM *policyEntry;
} CSSM_POLICY_INFO, *CSSM_POLICY_INFO_PTR;
```

Key Recovery Module Management Operations

This describes the interfaces for the key recovery module management operations.

CSSM_KR_SetEnterpriseRecoveryPolicy

Purpose

This call establishes the identity of the file that contains the enterprise key recovery policy function. It allows the use of a passphrase for access control to the update of the enterprise policy module. The first time this function is invoked, the old passphrase should be "default" in the Param field of the CSSM_CRYPTO_DATA_PTR. A passphrase can be established at this time for subsequent access control to this function by entering it in the NewPassphrase parameter. If the passphrase is to be changed, both the old and new passphrases have to be supplied.

The policy function module is operating system platform-specific (for Win95 and NT, it may be a Dynamic Link Library (DLL); for UNIX-based platforms, it may be a separate executable that gets launched by the OCSF). It is expected that the policy function file will be protected using the available protection mechanisms of the operating system platform. The policy function is expected to conform to this interface:

```
CSSM_BOOL EnterpriseRecoveryPolicy (CSSM_CONTEXT_PTR CryptoContext);
```

The CSSM_BOOL return value of this policy function will determine whether enterprise-based key recovery is mandated for the given cryptographic operation. CSSM_TRUE means that key recovery enablement is required for the given Context, and CSSM_FALSE means it is not.

Format

```
CSSM_RETURN CSSMAPI CSSM_KR_SetEnterpriseRecoveryPolicy
              (char * RecoveryPolicyFileName,
              const CSSM_CRYPTO_DATA_PTR OldPassPhrase,
              const CSSSM_CRYPTO_DATA_PTR NewPassphrase)
```

Parameters

Input

RecoveryPolicyFileName
: A pointer to a character string that specifies the filename of the module that contains the enterprise key recovery policy function. The filename may be a fully qualified pathname or a partial pathname.

OldPassPhrase
: The passphrase used to control access to this operation. This should be "default" when this function is invoked for the first time.

NewPassPhrase
: The value of the passphrase to be established for subsequent access to this function. It should be identical to the OldPassPhrase if the passphrase does not need to be updated.

Return Value

A CSSM return value. This function returns CSSM_OK if successful, and returns CSSM_FAIL if an error has occurred. Use CSSM_GetError to determine the error code.

Key Recovery Context Operations

Key recovery contexts are essentially cryptographic contexts. These API functions deal with the creation of these special types of cryptographic contexts. Once these contexts are created, the regular OCSF context API functions may be used to manipulate these key recovery contexts.

CSSM_KR_CreateRecoveryEnablementContext

Purpose

This call creates a key recovery enablement context based on a KRSP handle (which determines the key recovery mechanism that is in use) and key recovery profiles for the local and remote parties involved in a cryptographic exchange. It is expected that the LocalProfile will contain sufficient information to perform law enforcement, enterprise, and individual key recovery enablement, whereas, the RemoteProfile will contain information to perform law enforcement and enterprise key recovery enablement only. However, any and all of the fields within the profiles may be set to NULL–in this case, default values for these fields are to be used when performing the recovery enablement operations.

Format

```
CSSM_CC_HANDLE CSSMAPI CSSM_KR_CreateRecoveryEnablementContext
                  (CSSM_KRSP_HANDLE KRSPHandle,
                  const CSSM_KR_PROFILE LocalProfile,
                  const CSSM_KR_PROFILE RemoteProfile)
```

Parameters

Input

KRSPHandle
The handle to the KRSP that will be used.

LocalProfile
The key recovery profile for the local client.

RemoteProfile
The key recovery profile for the remote client.

Return Value

A handle to the key recovery enablement context is returned. If the handle is NULL, that signifies that an error has occurred.

CSSM_KR_CreateRecoveryRegistrationContext

Purpose

This call creates a key recovery registration context based on a KRSP handle (which determines the key recovery mechanism that is in use). This context may be used for performing registration with Key Recovery Servers (KRSs) and/or Key Recovery Agents (KRAs).

Format

```
CSSM_CC_HANDLE CSSMAPI CSSM_KR_CreateRecoveryRegistrationContext (CSSM_KRSP_HANDLE KRSPHandle)
```

Parameters

Input

KRSPHandle
The handle to the KRSP that is used.

Return Value

A handle to the key recovery registration context is returned. If the handle is NULL, that signifies that an error has occurred.

CSSM_KR_CreateRecoveryRequestContext

Purpose

This call creates a key recovery request context based on a KRSP handle (which determines the key recovery mechanism that is in use). The profile for the local party and flag values to signify what kind of key recovery is desired. A handle to the key recovery request context is returned.

Format

```
CSSM_CC_HANDLE CSSMAPI CSSM_KR_CreateRecoveryRequestContext
                (CSSM_KRSP_HANDLE KRSPHandle,
                const CSSM_KR_PROFILE_PTR LocalProfile)
```

Parameters

Input

KRSPHandle
The handle to the KRSP that is used.

LocalProfile
The key recovery profile for the local client. This parameter is relevant only when KRFlags is set to KR_INDIV.

Return Value

A handle to the key recovery context is returned. If the handle is NULL, that signifies that an error has occurred.

CSSM_KR_GetPolicyInfo

Purpose

This call is supported in the OCSF. This call should be used to determine the strength and type of cryptographic algorithm allowed.

Format

```
CSSM_RETURN CSSM_KR_GetPolicyInfo
               (CSSM_CC_HANDLE CCHandle,
               uint32 EncryptionProhibited,
               uint32 *WorkFactor)
```

Parameters

Input

CCHandle
The handle to the cryptographic context that will be used.

EncryptionProhibited
The usability field for law enforcement policy. Possible value is:

- KR_LE -Signifies that either the strength or algorithm specified in the cryptographic context is outside the allowable values after a policy enforcement check was done.

Output

Workfactor
The maximum permissible workfactor value that may be used for law enforcement key recovery.

Return Value

CSSM_OK if successful, CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Key Recovery Registration Operations

This describes the interfaces for the key recovery registration operations.

CSSM_KR_RegistrationRequest

Purpose

This function performs a key recovery registration operation. The KRInData parameter contains known input parameters for the recovery registration operation. A UserCallback function may be supplied to allow the registration operation to interact with the user interface, if necessary. When this operation is successful, a ReferenceHandle and an EstimatedTime parameter are returned; the ReferenceHandle will be used to invoke the CSSM_KR_RegistrationRetrieve function, after the EstimatedTime in seconds.

Format

```
CSSM_RETURN CSSMAPI CSSM_KR_RegistrationRequest
              (CSSM_CC_HANDLE RecoveryRegistrationContext,
              CSSM_DATA_PTR KRInData,
              CSSM_CRYPTO_DATA_PTR UserCallback,
              uint8 KRFlags,
              unit32 *EstimatedTime,
              CSSM_HANDLE_PTR ReferenceHandle)
```

Parameters

Input

RecoveryRegistrationContext
The handle to the key recovery registration context.

KRInData
Input data for key recovery registration.

UserCallBack
A callback function that may be used to collect further information from the user interface.

KRFlags
Flag values for recovery registration. Defined values are:

- KR_INDIVIDUAL - signifies that the registration is for the IND scenario
- KR_ENT - signifies that the registration is for the ENT scenario
- KR_LE - signifies that the registration is for the LE scenario

Output

EstimatedTime
The estimated time after which the CSSM_KR_RegistrationRetrieve call should be invoked to obtain registration results.

ReferenceHandle
The handle to use to invoke the CSSM_KR_RegistrationRetrieve function.

Return Value

CSSM_OK if successful, CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

CSSM_KR_RegistrationRetrieve

Purpose

This function completes a key recovery registration operation. The results of a successful registration operation are returned through the *KRProfile* parameter, which may be used with the profile management API functions.

If the results are not available when this function is invoked, the *KRProfile* parameter is set to NULL, and the *EstimatedTime* parameter indicates when this operation should be repeated with the same *ReferenceHandle*.

Format

```
CSSM_RETURN CSSMAPI CSSM_KR_RegistrationRetrieve
               (CSSM_KRSP_HANDLE hKRSP,
               CSSM_HANDLE ReferenceHandle,
               unit32 *EstimatedTime,
               CSSM_KR_PROFILE_PTR KRProfile)
```

Parameters

Input

hKRSP
The handle to the KRSP that will be used.

ReferenceHandle
The handle to the key recovery registration request that will be completed.

Output

EstimatedTime
The estimated time after which this call should be repeated to obtain registration results. This is set to a non-zero value only when the KRProfile parameter is NULL.

Input/Output

KRProfile
Key recovery profile that is filled in by the registration operation.

Return Value

CSSM_OK if successful, CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Key Recovery Enablement Operations

This describes the interfaces for the key recovery enablement operations.

CSSM_KR_GenerateRecoveryFields

Purpose

This function generates the Key Recovery Fields (KRFs) for a cryptographic association given the key recovery context, the session specific key recovery attributes, and the handle to the cryptographic context containing the key that will be made recoverable. The session attributes and the flags are not interpreted at the OCSF layer. If this call returns successfully, and the caller possesses the CSSM_STRONG_CRYPTO_WITH_KR privilege, the EncryptionProhibited flags within the CryptoContext may be modified, allowing the CryptoContext handle to be used for the OCSF encrypt APIs. The generated KRFs are returned as an output parameter. The *KRFlags* parameter may be used to fine tune the contents of the *KRFields* produced by this operation.

Format

```
CSSM_KR_GenerateRecoveryFields CSSM_RETURN CSSMAPI
                 (CSSM_CC_HANDLE hKRContext,
                 CSSM_CC_HANDLE CryptoContext,
                 CSSM_DATA_PTR KRSPOptions,
                 uint32 KRFlags,
                 CSSM_DATA_PTR KRFields)
```

Parameters

Input

hKRContext
The handle to the key recovery context for the cryptographic association.

CryptoContext
The cryptographic context handle that points to the session key.

KRSPOptions
The Key Recovery Service Provider (KRSP) specific options. These options are uninterpreted by the OCSF Framework, but passed on to the KRSP.

KRFlags
Flag values for KRFs generation. Defined values are:

- KR_INDL - Signifies that only the individual KRFs should be generated.
- KR_ENT - Signifies that only the enterprise KRFs should be generated.
- KR_LE - Signifies that only the law enforcement KRFs should be generated.
- KR_ALL - Signifies that law enforcement, enterprise, and individual KRFs should be generated.
- KR_OPTIMIZE - Signifies that performance optimization options are to be adopted by a KRSP while implementing this operation.
- KR_DROP_WORKFACTOR - Signifies that the law enforcement portion of the KRFs should be generated without using the key size workfactor.

Output

KRFields
The KRFs in the form of an uninterpreted data blob.

Return Value

CSSM_OK if successful, CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_RequestCssmExemption

CSSM_KR_ProcessRecoveryFields

Purpose

This function processes a set of KRFs given the key recovery context and the cryptographic context for the decryption operation. If the call is successful, and the caller possesses the CSSM_STRONG_CRYPTO_WITH_KR privilege, the EncryptionProhibited flags within the CryptoContextmay be modified, allowing the CryptoContext handle to be used for the OCSF decrypt API calls.

Format

```
CSSM_RETURN CSSMAPI CSSM_KR_ProcessRecoveryFields
               (CSSM_CC_HANDLE KeyRecoveryContext,
               CSSM_CC_HANDLE CryptoContext,
               CSSM_DATA_PTR KRSPOptions,
               uint32 KRFlags,
               CSSM_DATA_PTR KRFields)
```

Parameters

Input

KeyRecoveryContext
The handle to the key recovery context.

CryptoContext
A handle to the cryptographic context for which the KRFs are to be processed.

KRSPOptions
The KRSP specific options. These options are uninterpreted by the OCSF Framework, but passed on to the KRSP.

KRFlags
Flag values for KRFs processing. Defined values are:

- KR_ENT - Signifies that only the enterprise KRFs should be processed.
- KR_LE - Signifies that only the law enforcement KRFs should be processed.
- KR_ALL - Signifies that only the enterprise KRFs should be processed.
- KR_OPTIMIZE - Signifies that performance optimization options will be adopted by a KRSP while implementing this operation.

KRFields
The KRFs to be processed.

Return Value

CSSM_OK if successful, CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

Related Information

CSSM_RequestCssmExemption

Key Recovery Request Operations

This describes the interfaces for the key recovery request operations.

CSSM_KR_GetRecoveredObject

Purpose

This function is used to step through the results of a recovery request operation in order to retrieve a single recovered key at a time along with its associated meta-information. The cache handle returned from a successful CSSM_KR_RecoveryRetrieve operation is used. When multiple keys are recovered by a single recovery request operation, the *IndexInResults* parameter indicates which item to retrieve through this function.

The *RecoveredKey* parameter serves as an input template for the key to be returned. If a private key is to be returned by this operation, the *PassPhrase* parameter is used to inject the private key into the CSP indicated by the *RecoveredKey* template; the corresponding public key is returned in the *RecoveredKey* parameter. Subsequently, the *PassPhrase* and the public key may be used to reference the private key when operations using the private key are required. The *OtherInfo* parameter may be used to return other meta-data associated with the recovered key.

Format

```
CSSM_RETURN CSSMAPI CSSM_KR_GetRecoveredObject
               (CSSM_KRSP_HANDLE KRSPHandle,
               CSSM_HANDLE_PTR CacheHandle,
               unit32 IndexInResults,
               CSSM_CSP_HANDLE CSPHandle,
               const CSSM_CRYPTO_DATA_PTR PassPhrase,
               CSSM_KEY_PTR RecoveredKey,
               unit32 Flags,
               CSSM_DATA_PTR OtherInfo )
```

Parameters

Input

KRSPHandle
The handle to the KRSP that is to be used.

CacheHandle
Pointer to the handle returned from a successful CSSM_KR_RecoveryRequest operation.

IndexInResults
The index into the results that are referenced by the *ResultsHandle* parameter.

PassPhrase
This parameter is only relevant if the recovered key is a private key. It is used to protect the private key when it is inserted into the CSP specified by the *RecoveredKey* template.

Flags
Flag values relevant for recovery of a key. Possible values are:

- CERT_RETRIEVE - If the recovered key is a private key, return the corresponding public key certificate in the *OtherInfo* parameter.

Output

RecoveredKey
This parameter returns the recovered key.

OtherInfo
This parameter is used if there are additional information associated with the recovered key (such as the public key certificate when recovering a private key) that will be returned.

Input/optional

CSPHandle
This parameter identifies the CSP that the recovered key should be injected into. It may be set to NULL if the key is to be returned in raw form to the caller.

Return Value

CSSM_OK if successful, CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

CSSM_KR_RecoveryRequest

Purpose

This function performs a key recovery request operation. The *KRInData* contains known input parameters for the recovery request operation. A *UserCallback* function may be supplied to allow the recovery operation to interact with the user interface, if necessary. If the recovery request operation is successful, a *ReferenceHandle* and an *EstimatedTime* parameter are returned; the *ReferenceHandle* will be used to invoke the CSSM_KR_RecoveryRetrieve function, after the *EstimatedTime* in seconds.

Format

```
CSSM_RETURN CSSMAPI CSSM_KR_RecoveryRequest
              (CSSM_CC_HANDLE RecoveryRequestContext,
              const CSSM_DATA_PTR KRInData,
              const CSSM_CRYPTO_DATA_PTR UserCallback,
              unit32 *EstimatedTime,
              const CSSM_HANDLE_PTR ReferenceHandle)
```

Parameters

Input

RecoveryRequestContext
The handle to the key recovery request context.

KRInData
Input data for key recovery requests. For encapsulation schemes, the KRFs are included in this parameter.

UserCallBack
A callback function that may be used to collect further information from the user interface.

Output

ReferenceHandle
Handle returned when recovery request is successful. This handle may be used to invoke the CSSM_KR_RecoveryRetrieve function.

EstimatedTime
The estimated time after which the CSSM_KR_RecoveryRetrieve call should be invoked to obtain recovery results.

Return Value

CSSM_OK if successful, CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

CSSM_KR_RecoveryRequestAbort

Purpose

This function terminates a recovery request operation and releases any state information related to the recovery request.

Format

```
CSSM_RETURN CSSMAPI CSSM_KR_RecoveryRequestAbort (CSSM_KR_HANDLE KRSPHandle,CSSM_HANDLE CacheHandle)
```

Parameters

Input

KRSPHandle
The handle to the KRSP that is to be used.

CacheHandle
The handle returned from a successful CSSM_KR_RecoveryRequest operation.

Return Value

CSSM_OK if successful, CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

CSSM_KR_RecoveryRetrieve

Purpose

This function completes a key recovery request operation. The *ReferenceHandle* parameter indicates which outstanding recovery request is to be completed. The results of a successful recovery operation are referenced by the *ResultsHandle* parameter, which may be used with the CSSM_KR_GetRecoveredObject function to retrieve the recovered keys.

If the results are not available at the time this function is invoked, the *CacheHandle* is NULL, and the *EstimatedTime* parameter indicates when this operation should be repeated with the same *ReferenceHandle*.

Format

```
CSSM_RETURN CSSMAPI CSSM_KR_RecoveryRetrieve
               (CSSM_KRSP_HANDLE KRSPHandle,
               CSSM_HANDLE_PTR ReferenceHandle,
               unit32 *EstimatedTime,
               CSSM_HANDLE_PTR CacheHandle,
               unit32 *NumberOfRecoveredKeys)
```

Parameters

Input

KRSPHandle
The handle to the KRSP that is to be used.

ReferenceHandle
Handle that indicates which key recovery request operation is to be completed.

Output

EstimatedTime
The estimated time after which this call should be repeated to obtain recovery results. This is set to a non-zero value only when the ResultsHandle parameter is NULL.

CacheHandle
Handle returned when recovery operation is successful. This handle may be used to get individual keys using the CSSM_KR_GetRecoveredObject function. This handle is NULL if the EstimatedTime parameter is not zero.

NumberOfRecoveredKeys
The number of recovered key objects that may be obtained using the ResultsHandle.

Return Value

CSSM_OK if successful, CSSM_FAIL if an error occurred. Use CSSM_GetError to determine the exact error.

CSSM_KR_QueryPolicyInfo

Purpose

This function queries the law enforcement CSSM policy in effect and returns relevant information for application use. No privilege is required to invoke this function.

The policy information reports the maximum key length that can be generated, per cipher algorithm type and mode, without a need to generate key recovery blocks. It also specifies whether it is the jurisdiction of manufacturing or the jurisdiction of use to enforce the given policy. For special situations where the jurisdiction of use prohibits generation of key recovery fields, that information will also be provided.

Applications can request policy information relative to a specific algorithm, by providing the CSSM algorithm identifier in the first parameter to the call. If a CSSM_ALGID_NONE is provided in this field, the PolicyInfoData will contain

information pertaining to the entire set of algorithms controlled for the law enforcement jurisdiction. The *mode* parameter can be specified exactly, or set to CSSM_ALGMODE_NONE. In the latter case, all matching algorithm ids regardless of the actual mode are returned. The *class* parameter should be set correctly to symmetric or asymmetric, otherwise the results will not be accurate.

If the API cannot find a matching entry in the configured policies, the *numberOfEntries* field in *PolicyInfoData* is set to 0, and the return code is set to CSSM_OK. If the return code is set to CSSM_FAIL, there was an internal error that can be retrieved using CSSM_GetError API function.

Applications have the responsibility to free the memory associated with the policy information data when no longer needed.

Format

```
CSSM_RETURN CSSMAPI CSSM_KR_QueryPolicyInfo
               (const uint32 AlgorithmID,
               const uint32 Mode,
               const uint32 Class,
               CSSM_POLICY_INFO_PTR *PolicyInfoData)
```

Parameters

Input

Class
: The class of the desired algorithm. The allowed values are CSSM_ALGCLASS_ASYMMETRIC and CSSM_ALGCLASS_SYMMETRIC.

Mode
: The desired algorithm mode. This parameter can be set to CSSM_ALGMODE_NONE to get all applicable modes.

AlgorithmID
: CSSM defined algorithm identifier for which policy information is requested. This Parameter must be CSSM_ALGID_NONE if global policy information is desired.

Input/Output

PolicyInfoData
: Pointer to a CSSM policy information data structure to receive the query results.

Return Value

This function returns CSSM_OK if a privilege set was successfully retrieved. On error CSSM_FAIL is returned. Use CSSM_GetError to obtain the error code.

Chapter 14. Trust Policy Services API

The primary purpose of a Trust Policy (TP) module is to answer the question, "Is this certificate authorized for this action in this trust domain?" Applications are executed within some trust domain. For example, executing an installation program at the office takes place within the corporate information technology trust domain. Executing an installation program on a system at home takes place within the user's personal system trust domain. The TP that allows or blocks the installation action is different for the two domains. The corporate domain may require extensive credentials and accept only credentials signed by selected parties. The personal system domain may require only a credential that establishes the bearer as a known user on the local system.

The general OCSF trust model defines a set of basic trust objects that most (if not all) TPs use to model their trust domain and the policies over that domain. These basic trust objects include:

- Policies
- Certificates
- Defined sources of trust
- Certificate Revocation Lists
- Application-specific actions
- Evidence.

Policies define the credentials required for authorization to perform an action on another object. For example, a system administrator policy controls creating new user accounts on a computer system. Certificates are the basic credentials representing a trust relationship among a set of two or more parties. When an organization issues certificates, it defines its issuing procedure in a Certification Practice Statement (CPS). The statement identifies existing policies with which it is consistent. The statement also can be the source of new policy definitions if the action and target object domains are not covered by an existing, published policy. An application domain can recognize multiple policies. A given policy can be recognized by multiple application domains.

Evaluation of trust depends on relationships among certificates. For example, certificate chains represent hierarchical trust, where a root authority is the source of trust. Entities attain a level of trust based on their relationship to the root authority. Certificate graphs represent an introducer model of trust, where the number and strength of endorsers (i.e., immediate links in the graph) increases the level of trust attained by an entity. In both models, the trust domain can define accepted sources of trust. These may be mandated by fiat or can be computed by some other means. In contrast to the sources of trust, Certificate Revocation Lists (CRLs) represent sources of distrust. TPs may consult these lists during the verification process.

Trust evaluation can be performed with respect to a specific action the bearer wishes to perform, with respect to a policy, or with respect to the application domain in general. In the latter case, the action is understood to be either one specific action, or all actions in the domain.

When verifying trust, a TP module processes a group of certificates. The result of verification is a list of evidence, which forms an audit trail of the process. The evidence may be a list of verified attribute values that were contained in the certificates, or the entire set of verified certificates, or some other information that

serves as evidence of the verification. In the end, the trust and authorizations asserted are based on the authority implied by a set of assumed or otherwise specified public keys.

Many applications know a priori the TP modules it must use. The OCSF registry and query mechanism provides applications access to TP module descriptions. This information is provided by the TP module during installation and can assist the application in selecting the appropriate TP module for a given application domain.

Data Structures

This describes the Trust Policy data structures.

Note: Some application interfaces use data structures defined by other OCSF services. Those data structures are defined with those particular OCSF services.

CSSM_REVOKE_REASON

This data structure represents the reason a certificate is being revoked.

```
typedef enum cssm_revoke_reason {
    CSSM_REVOKE_CUSTOM = 0,
    CSSM_REVOKE_UNSPECIFIC = 1,
    CSSM_REVOKE_KEYCOMPROMISE = 2,
    CSSM_REVOKE_CACOMPROMISE = 3,
    CSSM_REVOKE_AFFILIATIONCHANGED = 4,
    CSSM_REVOKE_SUPERCEDED = 5,
    CSSM_REVOKE_CESSATIONOFOPERATION = 6,
    CSSM_REVOKE_CERTIFICATEHOLD = 7,
    CSSM_REVOKE_CERTIFICATEHOLDRELEASE = 8,
    CSSM_REVOKE_REMOVEFROMCRL = 9
} CSSM_REVOKE_REASON;
```

CSSM_TP_ACTION

This data structure represents a descriptive value defined by the TP module. A TP can define application-specific actions for the application domains over which the TP applies. Given a set of credentials, the TP module verifies authorizations to perform these actions.

```
typedef uint32 CSSM_TP_ACTION
```

CSSM_TP_HANDLE

This data structure represents the TP module handle. The handle value is a unique pairing between a TP module and an application that has attached that module. TP handles can be returned to an application as a result of the CSSM_ModuleAttach function.

```
typedef uint32 CSSM_TP_HANDLE/* Trust Policy Handle */
```

CSSM_TP_STOP_ON

This enumerated list defines the conditions controlling termination of the verification process by the TP module when a set of policies/conditions must be tested.

```
typedef enum cssm_tp_stop_on {
    CSSM_TP_STOP_ON_POLICY = 0,&tab;/* use the pre-defined stopping criteria */
    CSSM_TP_STOP_ON_NONE = 1,&tab;/* evaluate all condition whether T or F */
    CSSM_TP_STOP_ON_FIRST_PASS = 2, /* stop evaluation at first TRUE */
    CSSM_TP_STOP_ON_FIRST_FAIL = 3/* stop evaluation at first FALSE */
} CSSM_TP_STOP_ON;
```

CSSM_TPSUBSERVICE

Three structures are used to contain all of the static information that describes a TP module: cssm_moduleinfo, cssm_serviceinfo, and cssm_tpsubservice. This descriptive information is securely stored in the OCSF registry when the TP module is installed with OCSF. A TP module may implement multiple types of services and organize them as subservices. For example, a TP module supporting electronic transaction applications may organize its implementation into three subservices: one for micro-cash payments from an electronic wallet, a second for payments by credit card, and a third for payments by bank debit card. Most TP modules will implement exactly one subservice.

The descriptive information stored in these structures can be queried using the function CSSM_GetModuleInfo and specifying the trust policy module GUID.

```
typedef struct cssm_tpsubservice {
    uint32 SubServiceId;
    CSSM_STRING Description;
    CSSM_CERT_TYPE CertType;
    CSSM_USER_AUTHENTICATION_MECHANISM AuthenticationMechanism;
    uint32 NumberOfPolicyIdentifiers;
    CSSM_FIELD_PTR PolicyIdentifiers;
    CSSM_TP_WRAPPEDPRODUCT_INFO WrappedProduct;
} CSSM_TPSUBSERVICE, *CSSM_TPSUBSERVICE_PTR;
```

Definitions:

SubServiceId
: A unique, identifying number for the subservice described in this structure.

Description
: A string containing a descriptive name or title for this subservice.

CertType
: Type of certificate accepted by the TP module.

AuthenticationMechanism
: An enumerated value defining the credential format accepted by the TP module. An authentication credential is required for some TP functions. Presented credentials must be of the required format.

NumberOfPolicyIdentifiers
: The number of policies supported by this TP module.

PolicyIdentifiers
: &tab;A list of the policies (represented by their identifiers) supported by this TP module. There must be NumberOfPolicyIdentifiers entries in this list.

WrappedProduct
: Pointer to wrapped product information.

CSSM_TP_WRAPPEDPRODUCTINFO

```
typedef struct cssm_tp_wrappedproductinfo {
    CSSM_VERSION StandardVersion;
    CSSM_STRING StandardDescription;
    CSSM_STRING ProductVendor;
    uint32 ProductFlags;
} CSSM_TP_WRAPPEDPRODUCT_INFO, *CSSM_TP_WRAPPEDPRODUCT_INFO_PTR;
```

Definitions:

StandardVersion
: Version of standard to which this product conforms.

StandardDescription
: Description of standard to which this product conforms.

ProductVendor
: Vendor of wrapped product/library.

ProductFlags
: ProductFlags.

Trust Policy Operations

This describes the interfaces for the trust policy operations.

CSSM_TP_ApplyCrlToDb

Purpose

This function updates persistent storage to reflect entries in the CRL. 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.

Format

```
CSSM_RETURN CSSMAPI CSSM_TP_ApplyCrlToDb
               (CSSM_TP_HANDLE TPHandle,
               CSSM_CL_HANDLE CLHandle,
               CSSM_CSP_HANDLE CSPHandle,
               const CSSM_DL_DB_LIST_PTR DBList,
               const CSSM_DATA_PTR Crl)
```

Parameters

Input

TPHandle
The handle that describes the TP module used to perform this function.

Crl
A pointer to the CSSM_DATA structure containing a CRL to be applied to the data store.

Input/optional

CLHandle
The handle that describes the Certificate Library (CL) module that can be used to manipulate the CRL as it is applied to the data store and to manipulate the certificates affected by the CRL, if required. If no CL module is specified, the TP module uses an assumed CL module, if required.

CSPHandle
The handle referencing a Cryptographic Service Provider (CSP) 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.

DBList
A list of handle pairs specifying a Data Storage Library (DL) module and a data store managed by that module. These data stores can contain certificates that might be affected by the CRL, they may contain CRLs, or both. If no DL and database (DB) handle pairs are specified, the TP module must use an assumed DL module and an assumed data store for this operation.

Return Value

A CSSM_OK return value signifies that the revocations contained in the CRL have been appropriately applied to the specified database. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CrlGetFirstItem
CSSM_CL_CrlGetNextItem
CSSM_DL_CertRevoke

CSSM_TP_CertRevoke

Purpose

This function updates a CRL. The TP module determines whether the revoking certificate can revoke the target certificates. If authorized, a CRL record is added to the CRL and returned to the caller.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_TP_CertRevoke
                (CSSM_TP_HANDLE TPHandle,
                CSSM_CL_HANDLE CLHandle,
                CSSM_CC_HANDLE CCHandle,
                const CSSM_DL_DB_LIST_PTR DBList,
                const CSSM_DATA_PTR OldCrl,
                const CSSM_CERTGROUP_PTR CertGroupToBeRevoked,
                const CSSM_CERTGROUP_PTR RevokerCertGroup,
                CSSM_REVOKE_REASON Reason)
```

Parameters

Input

TPHandle
The handle that describes the TP module used to perform this function.

CCHandle
The handle that describes the context for a cryptographic operation. The cryptographic context specifies the handle of the CSP that must be used to perform the operation.

CertGroupToBeRevoked
A pointer to the CSSM_CERTGROUP structure containing one or more related certificates to be revoked.

RevokerCertGroup
A pointer to the CSSM_CERTGROUP structure containing the certificate used to revoke the target certificates.

Reason
The reason for revoking the target certificates.

Input/optional

CLHandle
The handle that describes the CL module that can be used to manipulate the certificates targeted for revocation and the revoker's certificates. If no CL module is specified, the TP module uses an assumed CL module, if required.

DBList
A list of handle pairs specifying a DL module and a data store managed by that module. These data stores can be used to store or retrieve objects (such as certificate and CRLs) related to the subject certificate and revoker's certificate. If no DL and DB handle pairs are specified, the TP module can use an assumed DL module and an assumed data store, if required.

OldCrl
A pointer to the CSSM_DATA structure containing an existing CRL. If this input is NULL, a new list is created.

Return Value

A pointer to the CSSM_DATA structure containing the updated CRL. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

CSSM_TP_CertSign

Purpose

This function signs a certificate and enforces a specific signing policy, such as X.509, or another standard that the TP module supports.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_TP_CertSign
                (CSSM_TP_HANDLE TPHandle,
                CSSM_CL_HANDLE CLHandle,
                CSSM_CC_HANDLE CCHandle,
                const CSSM_DL_DB_LIST_PTR DBList,
                const CSSM_DATA_PTR CertToBeSigned,
                const CSSM_CERTGROUP_PTR SignerCertGroup,
                const CSSM_FIELD_PTR SignScope,
                uint32 ScopeSize)
```

Parameters

Input

TPHandle
The handle that describes the TP module used to perform this function.

CCHandle
The handle that describes the context for a cryptographic operation. The cryptographic context specifies the handle of the CSP that must be used to perform the operation.

CertToBeSigned
A pointer to the CSSM_DATA structure containing a certificate to be signed.

SignerCertGroup
A pointer to the CSSM_CERTGROUP structure containing one or more related certificates used to sign the certificate.

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

Input/optional

CLHandle
The handle that describes the CL module that can be used to manipulate the certificate to be signed. If no CL module is specified, the TP module uses an assumed CL module, if required.

DBList
&tab;A list of handle pairs specifying a DL module and a data store managed by that module. These data stores can be used to store, retrieve objects (such as certificate and CRLs) related to the signer's certificate, or a data store for

storing a resulting signed CRL. If no DL and DB handle pairs are specified, the TP module can use an assumed DL module and an assumed data store, if required.

SignScope
A pointer to the CSSM_FIELD array containing the tags of the certificate fields to be included in the signing process. If the signing scope is null, the TP Module must assume a default scope (portions of the certificate to be hashed) when performing the signing process.

Return Value

A pointer to a CSSM_DATA structure containing the signed certificate. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

CSSM_TP_CrlSign

Purpose

This function signs a CRL. The TP module determines whether the signer's certificate is trusted to sign the CRL. If trust is satisfied, then the TP module has the option to carry out the service or to return a permission status without performing the service. This allows the library to support external as well as internal CRL service models. In either model, once a CRL is signed, revocation records can no longer be added to that CRL. To do so, would break the integrity of the signature resulting in a non-verifiable, rejected CRL.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_TP_CrlSign
                (CSSM_TP_HANDLE TPHandle,
                CSSM_CL_HANDLE CLHandle,
                CSSM_CC_HANDLE CCHandle,
                const CSSM_DL_DB_LIST_PTR DBList,
                const CSSM_DATA_PTR CrlToBeSigned,
                const CSSM_CERTGROUP_PTR SignerCertGroup,
                const CSSM_FIELD_PTR SignScope,
                uint32 ScopeSize)
```

Parameters

Input

TPHandle
The handle that describes the TP module used to perform this function.

CCHandle
The handle that describes the context for a cryptographic operation. The cryptographic context specifies the handle of the CSP and other cryptographic parameters that must be used to perform the operation.

CrlToBeSigned
A pointer to the CSSM_DATA structure containing a CRL to be signed.

SignerCertGroup
A pointer to the CSSM_CERTGROUP structure containing one or more related certificates used to sign the CRL.

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

Input/optional

CLHandle
The handle that describes the CL module that can be used to manipulate the certificates to be signed. If no CL module is specified, the TP module uses an assumed CL module, if required.

DBList
A list of handle pairs specifying a DL module and a data store managed by that module. These data stores can be used to store and retrieve objects (such as certificate and CRLs) related to the signer's certificate, or be a data store for storing a resulting signed CRL. If no DL and DB handle pairs are specified, the TP module can use an assumed DL module and an assumed data store, if required.

SignScope
A pointer to the CSSM_FIELD array containing the tags of the CRL fields to be included in the signing process. If the signing scope is null, the TP Module must assume a default scope (portions of the CRL to be hashed) when performing the signing process.

Return Value

A pointer to the CSSM_DATA structure containing the signed CRL. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

CSSM_TP_CrlVerify

Purpose

This function determines whether the CRL is trusted. The conditions for trust are part of the TP 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, the date the CRL was issued, the effective dates on the CRL, etc.

Format

```
CSSM_BOOL CSSMAPI CSSM_TP_CrlVerify
            (CSSM_TP_HANDLE TPHandle,
            CSSM_CL_HANDLE CLHandle,
            CSSM_CSP_HANDLE CSPHandle,
            const CSSM_DL_DB_LIST_PTR DBList,
            const CSSM_DATA_PTR CrlToBeVerified,
            const CSSM_CERTGROUP_PTR SignerCertGroup,
            const CSSM_FIELD_PTR VerifyScope,
            uint32 ScopeSize)
```

Parameters

Input

TPHandle
The handle that describes the TP module used to perform this function.

CrlToBeVerified
A pointer to the CSSM_DATA structure containing a signed CRL to be verified.

SignerCertGroup
A pointer to the CSSM_CERTGROUP structure containing one or more related certificates used to sign the CRL.

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

Input/optional

CLHandle
The handle that describes the CL module that can be used to manipulate the certificates to be verified. If no CL module is specified, the TP module uses an assumed CL module, if required.

CSPHandle
The handle referencing a CSP 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.

DBList
&tab;A list of handle pairs specifying a DL module and a data store managed by that module. These data stores can be used to store or retrieve objects (such as certificate and CRLs) related to the signer's certificate. If no DL and DB handle pairs are specified, the TP module can use an assumed DL module and an assumed data store, if required.

VerifyScope
A pointer to the CSSM_FIELD array indicating the CRL fields to be included in the CRL signature verification process. A null input verifies the signature assuming the module's default sets of fields were used in the signing process (this can include all fields in the CRL).

Return Value

A CSSM_TRUE return value signifies that the CRL can be trusted. When CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Group Functions

This describes the interfaces for the group functions.

CSSM_TP_CertGoupConstruct

Purpose

This function constructs an ordered certificate group using the certificates in *CertGroupFrag* as a starting point. There is no implied ordering for the certificates in *CertGroupFrag* except that the certificate in position 0 of the certificate group is assumed to the starting point for constructing the remaining certificate group. An ordering relationship may be defined and recorded in the certificates themselves or assumed by the TP model.

The certificate group is augmented by adding semantically related certificates obtained by searching the certificate data stores specified in *DBList*. For example, if the certificate model is a hierarchical model of certificate chains, the leaf certificate in the chain is a CertGroup fragment and the complete certificate chain, including the root certificate, is the anticipated result of the construction operation.

Format

```
CSSM_CERTGROUP_PTR CSSMAPI CSSM_TP_CertGroupConstruct
                    (CSSM_TP_HANDLE TPHandle,
                    CSSM_CL_HANDLE CLHandle,
                    CSSM_CSP_HANDLE CSPHandle,
                    CSSM_CERTGROUP_PTR CertGroupFrag,
                    CSSM_DL_DB_LIST_PTR DBList)
```

Parameters

Input

TPHandle
The handle to the TP module to perform this operation.

CSPHandle
The handle to the CSP that can be used for verification of certificate chains while constructing the certificate group.

CertGroupFrag
A list of certificates that form a possibly incomplete set of certificates. This set is used as the base set for constructing a complete certificate group.

DBList
A list of handle pairs specifying a DL module and a data store managed by that module. These data stores should contain certificates (and possibly other security objects). The data stores should be searched to complete construction of a semantically related certificate group.

Input/optional

CLHandle
The handle to the CL module that can be used to manipulate and parse values stored in the certgroup certificates. If no CL module is specified, the TP module uses an assumed CL module.

Return Value

A list of certificates that form a complete certificate group based on the original subset of certificates and the certificate data stores. A NULL list indicates an error.

Related Information

CSSM_TP_CertGroupPrune
CSSM_TP_CertGroupVerify

CSSM_TP_CertGroupPrune

Purpose

This function removes certificates from a 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 also can remove additional certificates that can be added to the certificate group again using the CSSM_CertGroupConstruct function. The pruned certificate group should be suitable for transmission to external hosts, 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.

Format

```
CSSM_CERTGROUP_PTRCSSMAPI CSSM_TP_CertGroupPrune
                (CSSM_TP_HANDLE TPHandle,
                CSSM_CL_HANDLE CLHandle,
                CSSM_CERTGROUP_PTR OrderedCertGroup,
                CSSM_DL_DB_LIST_PTR DBList)
```

Parameters

Input

TPHandle
The handle to the TP module to perform this operation.

OrderedCertGroup
The initial, complete set of certificates from which certificates will be selectively removed.

DBList
A list of handle pairs specifying a DL module and a data store managed by that module. These data stores should contain certificates (and possibly other security objects also). The data stores are searched for certificates semantically related to those in the certificate group to determine whether they should be removed from the certificate group.

Input/optional

CLHandle
The handle to the CL module that can be used to manipulate and parse the certgroup certificates and the certificates in the specified data stores. If no CL module is specified, the TP module uses an assumed CL module.

Return Value

Returns a certificate group containing those certificates which are verifiable credentials outside of the local system. If the list is NULL, an error has occurred.

Related Information

CSSM_TP_CertGroupConstruct
CSSM_TP_CertGroupVerify

CSSM_TP_CertGroupVerify

Purpose

This function verifies the signatures on each certificate in the group. Each certificate in the group has an associated signing certificate that was used to sign the subject certificate. Determination of the associated signing certificate is implied by the certificate model. For example, when verifying an X.509 certificate chain, the signing certificate for a certificate C is known to be the certificate of the issuers of certificate C. In a multisignature, web of trust model, the signing certificates can be any certificates in the CertGroup or unknown certificates.

Signature verification is performed on the *VerifyScope* fields for all certificates in the *CertGroup*. Additional validation tests can be performed on the certificates in the group depending on the certificate model supported by the TP. For example,

certificate expiration dates can be checked and appropriate CRLs can be searched as part of the verification process.

Format

```
CSSM_BOOL CSSMAPI CSSM_TP_CertGroupVerify
                        (CSSM_TP_HANDLE TPHandle,
                        CSSM_CL_HANDLE CLHandle,
                        CSSM_DL_DB_LIST_PTR DBList,
                        CSSM_CSP_HANDLE CSPHandle,
                        const CSSM_FIELD_PTR PolicyIdentifiers,
                        uint32 NumberofPolicyIdentifiers,
                        CSSM_TP_STOP_ON VerificationAbortOn,
                        const CSSM_CERTGROUP_PTR CertToBeVerified,
                        const CSSM_DATA_PTR AnchorCerts,
                        uint32 NumberofAnchorCerts,
                        const CSSM_FIELD_PTR VerifyScope,
                        uint32 ScopeSize,
                        CSSM_TP_ACTION Action,
                        const CSSM_DATA_PTR Data,
                        CSSM_DATA_PTR *Evidence,
                        uint32 *EvidenceSize)
```

Parameters

Input

TPHandle
The handle to the TP module to perform this operation.

NumberofPolicyIdentifiers
The number of policy identifiers provided in the *PolicyIdentifiers* parameter.

NumberofAnchorCerts
The number of anchor certificates provided in the *AnchorCerts* parameter.

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

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

Output

EvidenceSize
The number of entries in the *Evidence* list. The returned value is zero if no evidence is produced. *Evidence* may be produced even when verification fails. This evidence can describe why and how the operation failed to verify the subject certificate.

Input/optional

CLHandle
The handle to the CL module that can be used to manipulate and parse the certgroup certificates and the certificates in the specified data stores. If no CL module is specified, the TP module uses an assumed CL module.

DBList
A list of handle pairs specifying a DL module and a data store managed by that module. These data stores should contain zero or more trusted certificates. If no data stores are specified, the TP module can assume a default data store, if required.

CSPHandle
The handle of a CSP that can be used for verification of the certificate chain.

PolicyIdentifiers
The policy identifier is an object identifier (OID)/value pair. The CSSM_OID structure contains the name of the policy and the value is an optional caller-specified input value for the TP module to use when applying the policy.

VerificationAbortOn
When a TP module verifies multiple conditions or multiple policies, the TP module can allow the caller to specify when to abort the verification process. If supported by the TP module, this selection can effect the evidence returned by the TP module to the caller. The default stopping condition is to stop evaluation according to the policy defined in the TP Module. The specifiable stopping conditions and their meaning are defined in Table 38.

Table 38. Specifiable Stopping Conditions

CSSM_TP_STOP_ON	Definitions
CSSM_STOP_ON_POLICY	Stop verification whenever the policy dictates it.
CSSM_STOP_ON_NONE	Stop verification only after all conditions have been tested (ignoring the pass-fail status of each condition).
CSSM_STOP_ON_FIRST_PASS	Stop verification on the first condition that passes.
CSSM_STOP_ON_FIRST_FAL	Stop verification on the first condition that fails.

The TP module may ignore the caller's specified stopping condition and revert to the default of stopping according to the policy embedded in the module.

AnchorCerts
A pointer to the CSSM_DATA structure containing one or more certificates to be used in order to validate the subject certificate. These certificates can be root certificates, cross-certified certificates, and certificates belonging to locally designated sources of trust.

VerifyScope
A pointer to the CSSM_FIELD array containing the OID indicators specifying the certificate fields to be used in the verification process. If VerifyScope is not specified, the TP Module must assume a default scope (portions of each certificate) when performing the verification process.

Action
An application-specific and application-defined action to be performed under the authority of the input certificate. If no action is specified, the TP module defines a default action and performs verification assuming that action is being requested. Note that it is possible that a TP module verifies certificates for only one action.

Data
&tab;A pointer to the CSSM_DATA structure containing the application-specific data or a reference to the application-specific data upon which the requested action should be performed. If no data is specified, the TP module defines one or more default data objects upon which the action or default action would be performed.

Evidence
A pointer to a list of CSSM_DATA objects containing an audit trail of evidence constructed by the TP module during the verification process. Typically, this is a list of certificates and CRLs that were used to establish the validity of the CertToBeVerified, but other objects may be appropriate for other types of TPs.

Return Value

CSSM_TRUE if the certificate group verified. CSSM_FALSE if the certificate did not verify or an error condition occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_TP_CertGroupConstruct
CSSM_TP_CertGroupPrune

Extensibility Functions

This describes the trust policy extensibility functions.

CSSM_TP_PassThrough

Purpose

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

Format

```
void * CSSMAPI CSSM_TP_PassThrough
                    (CSSM_TP_HANDLE TPHandle,
                    CSSM_CL_HANDLE CLHandle,
                    CSSM_DL_HANDLE DLHandle,
                    CSSM_DB_HANDLE DBHandle,
                    CSSM_CC_HANDLE CCHandle,
                    uint32 PassThroughId,
                    const void *InputParams)
```

Parameters

Input

TPHandle
The handle that describes the TP module used to perform this function.

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

Output

InputParms
A pointer to a module, implementation-specific structure containing parameters to be interpreted in a function-specific manner by the requested TP module.

Input/optional

CLHandle
The handle that describes the DL module that can be used to store or retrieve objects (such as certificate and CRLs) related to the subject certificate and anchor certificates. If no DL module is specified, the TP module uses an assumed DL module, if required.

DLHandle
The handle that describes the DL module that can be used to store or retrieve objects (such as certificate and CRLs) related to the subject certificate and anchor certificates. If no DL module is specified, the TP module uses an assumed DL module, if required.

DBHandle
The handle that describes the data store that can be accessed to store or retrieve objects (such as certificate and CRLs) related to the subject certificate and anchor certificates. If no data store is specified, the TP module uses an assumed data store, if required.

CCHandle
The handle that describes the context for a cryptographic operation. The cryptographic context specifies the handle of the CSP that must be used to perform the operation. If no cryptographic context is specified, the TP module uses an assumed context, if required.

Return Value

A pointer to a module, implementation-specific structure containing the output from the passthrough function. The output data must be interpreted by the calling application based on externally available information. If the pointer is NULL, an error has occurred.

Chapter 15. Certificate Library Services API

The primary purpose of a Certificate Library (CL) module is to perform syntactic operations on a specific certificate format and its associated Certificate Revocation List (CRL) format. This encapsulation allows applications and TP modules to focus on the usage of certificates rather than the mechanics of format manipulation.

The syntactic operations on certificates include field management operations and cryptographic operations. Field management operations allow an application to input fields into a certificate and retrieve fields from a certificate without knowledge of the certificate's content organization or encoding format. Cryptographic operations on certificates encode the proper fields of a certificate in the proper order prior to executing certificate signing and verification.

The syntactic operations on CRLs mirror the operations on their corresponding certificate format. CRL field management operations allow the insertion and retrieval of CRL fields, including addition and removal of certificates from the revocation list. The CL module manages the translation from the certificate to be revoked to its representation in the CRL. The CL module also properly encodes the necessary fields of a CRL prior to signing and verification.

Each CL module may implement some or all of these functions on certificates and CRLs. The available functions are registered with OCSF when the module is attached. Each CL module should be accompanied with information specifying supported functions, nonsupported functions, and module-specific passthrough functions. It is the responsibility of the application developer to obtain and use this information when developing applications using a selected CL module.

A CL module's functionality may be partitioned, as appropriate, between the local client and a remote server. For example, a CL module may redirect the *CSSM_CL_CertSign* function to a Certificate Authority (CA) server application, but perform the *CSSM_CL_CertGetKeyInfo* function as a local operation.

CL modules manipulate memory-based objects only. The persistence of certificates, CRLs, and other security-related objects is an independent property of these objects. It is the responsibility of the application and/or the TP module to use data storage modules to make objects persistent (if appropriate).

Data Structures

This describes the data structures that may be passed to or returned from a CL function. They will be used by applications to prepare data to be passed as input parameters into OCSF API function calls that will be passed without modification to the appropriate CL module. The CL module is then responsible for interpreting them and returning the appropriate data structure to the calling application via OCSF. These data structures are defined in the header file, cssmtype.h, which is distributed with OCSF.

Note: Some application interfaces use data structures defined by other OCSF services. Those data structures are defined with those particular OCSF services.

CSSM_CA_SERVICES

This bit-mask defines the additional certificate-creation-related services that an issuing CA can offer. Such services include (but are not limited to) archiving the certificate and keypair, publishing the certificate to one or more certificate directory services, and sending automatic, out-of-band notifications of the need to renew a certificate. A CA may offer any subset of these services. Additional services can be defined over time.

```
typedef uint32 CSSM_CA_SERVICES;

#define CSSM_CA_KEY_ARCHIVE       0x0001&tab;  /* archive cert & keys */
#define CSSM_CA_CERT_PUBLISH      0x0002&tab;  /* cert in directory service */
#define CSSM_CA_CERT_NOTIFY_RENEW&tab; 0x0004  /* notify at renewal time */
#define CSSM_CA_CRL_DISTRIBUTE    0x0008&tab;  /* push CRL to everyone */
```

CSSM_CERT_ENCODING

This variable specifies the certificate-encoding format supported by a CL.

```
typedef enum cssm_cert_encoding {
    CSSM_CERT_ENCODING_UNKNOWN  =  0x00,
    CSSM_CERT_ENCODING_CUSTOM   =  0x01,
    CSSM_CERT_ENCODING_BER      =  0x02,
    CSSM_CERT_ENCODING_DER      =  0x03,
    CSSM_CERT_ENCODING_NDR      =  0x04,
} CSSM_CERT_ENCODING, *CSSM_CERT_ENCODING_PTR;
```

CSSM_CERTGROUP

This structure contains a set of certificates. It is assumed that the certificates are related based on cosigning. The certificate group is a syntactic representation of a trust model. All certificates in the group must be of the same type. Typically, the certificates are related in some manner, but this is not required.

```
typedef struct cssm_certgroup {
    uint32 NumCerts;
    CSSM_DATA_PTR CertList;
    void *reserved;
} CSSM_CERTGROUP, *CSSM_CERTGROUP_PTR;
```

Definitions:

NumCerts

Number of certificates in the group.

CertList

List of certificates.

Reserved

Reserved for future use.

CSSM_CERT_TYPE

This variable specifies the type of certificate format supported by a CL and the types of certificates understood for import and export. They are expected to define such well-known certificate formats as X.509 Version 3 and Simple Distributed Security Infrastructure (SDSI) as well as custom certificate formats. The list of enumerated values can be extended for new types by defining a label with an associated value greater than CSSM_CL_CUSTOM_CERT_TYPE.

```
typedef uint32 CSSM_CERT_TYPE,*CSSM_CERT_TYPE_PTR;
/* bit masks for supported cert types */
#define     CSSM_CERT_UNKNOWN        0x00000000
#define     CSSM_CERT_X_509v1        0x00000001
#define     CSSM_CERT_X_509v2        0x00000002
#define     CSSM_CERT_X_509v3&tab;       0x00000004
#define     CSSM_CERT_Fortezza&tab;      0x00000008
#define     CSSM_CERT_PGP            0x00000010
#define     CSSM_CERT_SPKI           0x00000020
#define     CSSM_CERT_SDSIv1         0x00000040
#define     CSSM_CERT_Intel          0x00000080
```

```
#define     CSSM_CERT_ATTRIBUTE_BER  0x00000100
#define     CSSM_CERT_X509_CRL       0x00000200
#define     CSSM_CERT_LAST           0x00007fff

/* Applications wishing to define their own custom certificate
 * type should create a random uint32 whose value is greater than
 * the CSSM_CL_CUSTOM_CERT_TYPE */
#define     CSSM_CL_CUSTOM_CERT_TYPE   0x08000
```

CSSM_CL_CA_CERT_CLASSINFO

```
typedef struct cssm_cl_ca_cert_classinfo {
    CSSM_STRING CertClassName;
    CSSM_DATA CACert;
} CSSM_CL_CA_CERT_CLASSINFO, *CSSM_CL_CA_CERT_CLASSINFO_PTR;
```

Definitions:

CertClassName
: Name of a certificate class issued by this certificate authority.

CACert
: CA certificate for this cert class.

CSSM_CL_CA_PRODUCTINFO

This structure holds product information about a backend CA that is accessible to the CL module. The CL module vendor is not required to provide this information, but may choose to do so. For example, a CL module that implements upstream protocols to a particular type of commercial CA can record information about that CA service in this structure.

```
typedef struct cssm_cl_ca_productinfo {
    CSSM_VERSION StandardVersion;
    CSSM_STRING StandardDescription;
    CSSM_VERSION ProductVersion;
    CSSM_STRING ProductDescription;
    CSSM_STRING ProductVendor;
    CSSM_CERT_TYPE CertType;
    CSSM_CA_SERVICES AdditionalServiceFlags;
    uint32 NumberOfCertClasses;
    CSSM_CL_CA_CERT_CLASSINFO_PTR CertClasses;
} CSSM_CL_CA_PRODUCTINFO, *CSSM_CL_CA_PRODUCTINFO_PTR;
```

Definitions:

StandardVersion
: If this product conforms to an industry standard, this is the version number of that standard.

StandardDescription
: If this product conforms to an industry standard, this is a description of that standard.

ProductVersion
: Version number information for the actual product version used in this version of the CL module.

ProductDescription
: A string describing the product.

ProductVendor
: The name of the product vendor.

CertType
: An enumerated value specifying the certificate and CRL type that the CA manages.

AdditionalServiceFlags
: A bit-mask indicating the additional services a caller can request from a CA (as side effects and in conjunction with other service requests).

NumberOfCertClasses
: The number of classes or levels of certificates managed by this CA.

CertClasses
: Names of the certificate classes issued by this CA

CSSM_CL_ENCODER_PRODUCTINFO

This structure holds product information about embedded products that a CL module uses to provide its services. The CL module vendor is not required to provide this information, but may choose to do so. For example, a CL module that manipulates X.509 certificates may embed a third-party tool that parses, encodes, and decodes those certificates. The CL module vendor can describe such embedded products using this structure.

```
typedef struct cssm_cl_encoder_productinfo {
    CSSM_VERSION StandardVersion;
    CSSM_STRING StandardDescription;
    CSSM_VERSION ProductVersion;
    CSSM_STRING ProductDescription;
    CSSM_STRING ProductVendor;
    CSSM_CERT_TYPE CertType;
    uint32 ProductFlags;
} CSSM_CL_ENCODER_PRODUCTINFO, *CSSM_CL_ENCODER_PRODUCTINFO_PTR;
```

Definitions:

StandardVersion
: If this product conforms to an industry standard, this is the version number of that standard.

StandardDescription
: If this product conforms to an industry standard, this is a description of that standard.

ProductVersion
: Version number information for the actual product version used in this version of the CL module.

ProductDescription
: A string describing the product.

ProductVendor
: The name of the product vendor.

CertType
: An enumerated value specifying the certificate and CRL type that the CA manages.

ProductFlags
: A bit-mask indicating any selectable features of the embedded product that the CL module selected for use.

CSSM_CL_HANDLE

The CSSM_CL_HANDLE is used to identify the association between an application thread and an instance of a CL module. It is assigned when an application causes OCSF to attach to a CL. It is freed when an application causes OCSF to detach from a CL. The application uses the CSSM_CL_HANDLE with every CL function call to identify the targeted CL. The CL module uses the CSSM_CL_HANDLE to identify the appropriate application's memory management routines when allocating memory on the application's behalf.

```
typedef uint32 CSSM_CL_HANDLE
```

CSSM_CLSUBSERVICE

Three structures are used to contain all of the static information that describes a CL module: cssm_moduleinfo, cssm_serviceinfo, and cssm_clsubservice. This descriptive information is securely stored in the OCSF registry when the CL module is installed with OCSF. A CL module may implement multiple types of

services and organize them as subservices. For example, a CL module supporting X.509 encoded certificates may organize its implementation into three subservices: one for X.509 Version 1, a second for X.509 Version 2, and a third for X.509 Version 3. Most CL modules will implement exactly one subservice.

The descriptive information stored in these structures can be queried using the function CSSM_GetModuleInfo and specifying the CL module Globally Unique ID (GUID).

```
typedef struct cssm_clsubservice {
   uint32 SubServiceId;
   CSSM_STRING Description;
   CSSM_CERT_TYPE CertType;
   CSSM_CERT_ENCODING CertEncoding;
   CSSM_USER_AUTHENTICATION_MECHANISM AuthenticationMechanism;
   uint32 NumberOfTemplateFields;
   CSSM_OID_PTR CertTemplates;
   uint32 NumberOfTranslationTypes;
   CSSM_CERT_TYPE_PTR CertTranslationTypes;
   CSSM_CL_WRAPPEDPRODUCT_INFO WrappedProduct;
} CSSM_CLSUBSERVICE, *CSSM_CLSUBSERVICE_PTR;
```

Definitions:

SubServiceId
: A unique, identifying number for the subservice described in this structure.

Description
: A string containing a description name or title for this subservice.

CertType
: An identifier for the type of certificate. This parameter is also used to determine the certificate data format.

CertEncoding
: An identifier for the certificate encoding format.

AuthenticationMechanism
: An enumerated value defining the credential format accepted by the CL module. Authentication credential may be required when requesting certificate creation or other CL functions. Presented credentials must be of the required format.

NumberOfTemplateFields
: The number of certificate fields. This number also indicates the length of the CertTemplate array.

CertTemplates
: A pointer to an array of tag/value pairs which identify the field values of a certificate.

NumberOfTranslationTypes
: The number of certificate types that this CL module can import and export. This number also indicates the length of the *CertTranslationTypes* array.

CertTranslationTypes
: &tab;A pointer to an array of certificate types. This array indicates the certificate types that can be imported into and exported from this CL module's native certificate type.

WrappedProduct
: A data structure describing the embedded products and CA service used by the CL module.

CSSM_CL_WRAPPEDPRODUCTINFO

This structure lists the set of embedded products and the CA service used by the CL module to implement its services. The CL module is not required to provide any of this information, but may choose to do so.

```
typedef struct cssm_cl_wrappedproductinfo {
    CSSM_CL_ENCODER_PRODUCTINFO_PTR EmbeddedEncoderProducts;
    uint32 NumberOfEncoderProducts;
    CSSM_CL_CA_PRODUCTINFO_PTR AccessibleCAProducts;
    uint32 NumberOfCAProducts;
} CSSM_CL_WRAPPEDPRODUCTINFO, *CSSM_CL_WRAPPEDPRODUCTINFO_PTR;
```

Definitions:

EmbeddedEncoderProducts
: An array of structures that describe each embedded encoder product used in this CL module implementation.

NumberOfEncoderProducts
: A count of the number of distinct embedded certificate encoder products used in the CL module implementation.

AccessibleCAProducts
: An array of structures that describe each type of CA accessible through this CL module implementation.

NumberOfCAProducts
: A count of the number of distinct CA products described in the array *AccessibleCAProducts*.

CSSM_FIELD

This structure contains the object identifier (OID)/value pair for any item that can be identified by an OID. A CL module uses this structure to hold an OID/value pair for a field in a certificate or CRL.

```
typedef struct cssm_field {
    CSSM_OID FieldOid;
    CSSM_DATA FieldValue;
}CSSM_FIELD, *CSSM_FIELD_PTR
```

Definitions:

FieldOid
: The OID that identifies the certificate or CRL data type or data structure.

FieldValue
: A CSSM_DATA type which contains the value of the specified OID in a contiguous block of memory.

CSSM_OID

The OID is used to hold an identifier for the data types and data structures that comprise the fields of a certificate or CRL. The underlying representation and meaning of the identifier is defined by the CL module. For example, a CL module can choose to represent its identifiers in any of these forms:

- A character string in a character set native to the platform
- A DER-encoded X.509 OID that must be parsed
- An S-expression that must be evaluated
- An enumerated value that is defined in header files supplied by the CL module.

```
typedef CSSM_DATA CSSM_OID, *CSSM_OID_PTR
```

Certificate Operations

This describes the certificate operations interfaces for CL.

CSSM_CL_CertAbortQuery

Purpose

This function terminates the get operation initiated by CSSM_CL_CertGetFirstFieldValue or CSSM_CL_GetNextFieldValue, and allows the CL to release all intermediate state information associated with the query. This function should be called even if all values retrieved by the call to CSSM_CL_CertGetFirstFieldValue are obtained by repeated calls to CSSM_CL_CertGetNextFieldValue.

Format

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

Parameters

Input

ResultsHandle
The handle that identifies the results of a get field value request.

CLHandle
The handle that describes the CL module used to perform this function.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CertGetFirstFieldValue
CSSM_CL_CertGetNextFieldValue

CSSM_CL_CertCreateTemplate

Purpose

This function allocates and initializes memory for a certificate based on the input OID/value pairs specified in the *CertTemplate*. The initialization process includes encoding all certificate field values according to the format required by the certificate representation. The function returns the initialized template containing encoded values. The memory is allocated using the calling application's memory management routines.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_CL_CertCreateTemplate
                (CSSM_CL_HANDLE CLHandle,
                const CSSM_FIELD_PTR CertTemplate,
                uint32 NumberOfFields)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

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

NumberOfFields
The number of certificate field values specified in the CertTemplate.

Return Value

A pointer to the CSSM_DATA structure containing the unsigned certificate template. If the return pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CertRequest
CSSM_CL_CertGetFirstFieldValue

CSSM_CL_CertDescribeFormat

Purpose

This function returns a list of the object identifiers used to describe the certificate format supported by the specified CL.

Format

```
CSSM_OID_PTR CSSMAPI CSSM_CL_CertDescribeFormat (CSSM_CL_HANDLE CLHandle, uint32 *NumberOfFields)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

Output

NumberOfFields
The length of the returned array of OIDs.

Return Value

A pointer to the array of CSSM_OIDs that represent the supported certificate format. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CertGetAllFields
CSSM_CL_CertGetFirstFieldValue
CSSM_CL_CertGetNextFieldValue
CSSM_CL_CertAbortQuery
CSSM_CL_CertGetKeyInfo

CSSM_CL_CertExport

Purpose

This function exports a certificate from the native format of the specified CL into the specified target certificate format. The set of *TargetCertTypes* supported for export varies with the CL module. See the information provided by the module vendor for a list of target certificate formats.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_CL_CertExport
          (CSSM_CL_HANDLE CLHandle,
          CSSM_CERT_TYPE TargetCertType,
          const CSSM_DATA_PTR NativeCert)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

TargetCertType
A unique value which identifies the target type of the certificate being exported.

NativeCert
A pointer to the CSSM_DATA structure containing the certificate to be exported.

Return Value

A pointer to the CSSM_DATA structure containing the target-type certificate exported from the native certificate. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CertImport

CSSM_CL_CertGetAllFields

Purpose

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

Format

```
CSSM_FIELD_PTR CSSMAPI CSSM_CL_CertGetAllFields
                (CSSM_CL_HANDLE CLHandle,
                const CSSM_DATA_PTR Cert,
                uint32 *NumberOfFields)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

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

Output

NumberOfFields
The length of the returned array of fields.

Return Value

A pointer to an array of CSSM_FIELD structures that contain the values of all of the fields of the input certificate. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CertGetFirstFieldValue
CSSM_CL_CertDescribeFormat
CSSM_CL_CertView

CSSM_CL_CertGetFirstFieldValue

Purpose

This function returns the value of the designated certificate field. 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.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_CL_CertGetFirstFieldValue
                (CSSM_CL_HANDLE CLHandle,
                const CSSM_DATA_PTR Cert,
                const CSSM_OID_PTR CertField,
                CSSM_HANDLE_PTR ResultsHandle,
                uint32 *NumberOfMatchedFields)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

Cert
A pointer to the CSSM_DATA structure containing the certificate.

CertField
A pointer to an OID that identifies the field value to be extracted from the Cert.

Output

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

NumberOfMatchedFields
The number of fields that match the CertField OID.

Return Value

A pointer to the CSSM_DATA structure containing the value of the requested field. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CertGetNextFieldValue
CSSM_CL_CertAbortQuery
CSSM_CL_CertGetAllFields

CSSM_CL_CertGetKeyInfo

Purpose

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.

Format

```
CSSM_KEY_PTR CSSMAPI CSSM_CL_CertGetKeyInfo (CSSM_CL_HANDLE CLHandle,const CSSM_DATA_PTR Cert )
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

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

Return Value

A pointer to the CSSM_KEY structure containing the public key and possibly other key information. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CertGetFirstFieldValue

CSSM_CL_CertGetNextFieldValue

Purpose

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 initiates the process and returns a results handle identifying the certificate from which values are being obtained and the OID corresponding to those values. The CSSM_CL_CertGetNextFieldValue function can be called repeatedly to obtain these values one at a time.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_CL_CertGetNextFieldValue (CSSM_CL_HANDLE CLHandle, CSSM_HANDLE ResultsHandle)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

ResultsHandle
The handle that identifies the results of a certificate query.

Return Value

A pointer to the CSSM_DATA structure containing the value of the requested field. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CertGetFirstFieldValue
CSSM_CL_CertAbortQuery

CSSM_CL_CertImport

Purpose

This function imports a certificate from the specified foreign format into the native format of the specified CL. The set of ForeignCertTypes supported for import varies with the CL module. See the information provided by the module vendor for a list of supported foreign certificate formats.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_CL_CertImport
                (CSSM_CL_HANDLE CLHandle,
                CSSM_CERT_TYPE ForeignCertType,
                const CSSM_DATA_PTR ForeignCert)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

ForeignCertType
A unique value that identifies the type of the certificate being imported.

ForeignCert
A pointer to the CSSM_DATA structure containing the certificate to be imported into the CL modules native certificate type.

Return Value

A pointer to the CSSM_DATA structure containing the native-type certificate imported from the foreign certificate. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CertExport

CSSM_CL_CertSign

Purpose

This function creates a signed certificate by signing the fields of the input certificate as indicated by the *SignScope* array.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_CL_CertSign
        (CSSM_CL_HANDLE CLHandle,
        CSSM_CC_HANDLE CCHandle,
        const CSSM_DATA_PTR CertToBeSigned,
        const CSSM_DATA_PTR SignerCert,
        const CSSM_FIELD_PTR SignScope,
        uint32 ScopeSize)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

CCHandle
The handle that describes the context of this cryptographic operation.

CertToBeSigned
The DER-encoded certificate to be signed.

SignerCert
A pointer to the CSSM_DATA structure containing the certificate to be used to sign the subject certificate.

SignScope
A pointer to the CSSM_FIELD array containing the tag/value pairs of the fields to be signed. A null input signs all the fields in the certificate.

ScopeSize
The number of entries in the sign scope list.

Return Value

A pointer to the CSSM_DATA structure containing the signed certificate. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CertVerify

CSSM_CL_CertVerify

Purpose

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.

Format

```
CSSM_BOOL CSSMAPI CSSM_CL_CertVerify
           (CSSM_CL_HANDLE CLHandle,
           CSSM_CC_HANDLE CCHandle,
           const CSSM_DATA_PTR CertToBeVerified,
           const CSSM_DATA_PTR SignerCert,
           const CSSM_FIELD_PTR VerifyScope,
           uint32 ScopeSize)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

CCHandle
The handle that describes the context of this cryptographic operation.

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

SignerSize
A pointer to the CSSM_DATA structure containing the certificate used to sign the subject certificate.

ScopeSize
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.

Input/optional

VerifyScope
A pointer to the CSSM_FIELD array containing the tag/value pairs of the fields to be used in verifying the signature (i.e., the fields that were used to calculate the signature). If the verify scope is null, the CL 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.

Return Value

CSSM_TRUE if the certificate signature verified. CSSM_FALSE if the certificate signature did not verify or an error condition occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CertSign

Certificate Revocation List Operations

This describes the certification revocation list operation interfaces for CL.

CSSM_CL_CRLAbortQuery

Purpose

This function terminates the query initiated by CSSM_CL_CrlGetFirstFieldValue or CSSM_CL_CrlGetNextFieldValue and allows the CL to release all intermediate state

information associated with the get operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_CL_CrlAbortQuery (CSSM_CL_HANDLE CLHandle, CSSM_HANDLE ResultsHandle)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

ResultsHandle
The handle that identifies the results of a CRL query.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CrlGetFirstFieldValue
CSSM_CL_CrlGetNextFieldValue

CSSM_CL_CrlAddCert

Purpose

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 revoker's certificate is used to sign the new CRL entry. The operation is valid only if the CRL has not been closed by the process of signing the CRL (i.e., execution of the function CSSM_CL_CrlSign). Once the CRL has been signed, entries can not be added or removed.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlAddCert
                (CSSM_CL_HANDLE CLHandle,
                CSSM_CC_HANDLE CCHandle,
                const CSSM_DATA_PTR Cert,
                const CSSM_DATA_PTR RevokerCert,
                const CSSM_FIELD_PTR CrlEntryFields,
                uint32 NumberOfFields,
                const CSSM_DATA_PTR OldCrl)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

CCHandle
The handle that describes the context of this cryptographic operation.

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

RevokerCert
&tab;A pointer to the CSSM_DATA structure containing the revoker's certificate.

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

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

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

Return Value

A pointer to the CSSM_DATA structure containing the updated CRL. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CrlRemoveCert

CSSM_CL_CrlCreateTemplate

Purpose

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 may be set using the CSSM_CL_CrlSetFieldValues operation. The new CRL contains no revocation records.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlCreateTemplate
                (CSSM_CL_HANDLE CLHandle,
                const CSSM_FIELD_PTR CrlTemplate,
                uint32 NumberOfFields)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

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

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

Return Value

A pointer to the CSSM_DATA structure containing the new CRL. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

CSSM_CL_CrlDescribeFormat

Purpose

This function returns a list of the object identifiers used to describe the CRL format supported by the specified CL.

Format

```
CSSM_OID_PTR CSSMAPI CSSM_CL_CrlDescribeFormat (CSSM_CL_HANDLE CLHandle,uint32 *NumberOfFields)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

Output

NumberOfFields
The length of the returned array of OIDs.

Return Value

A pointer to the array of CSSM_OIDs which represent the supported CRL format. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

CSSM_CL_CrlGetFirstFieldValue

Purpose

This function returns the value of the designated CRL field. If more than one field matches the *CrlField* OID, the first matching field will be returned. The number of matching fields, *NumberOfMatchedFields*, is an output parameter, as is the *ResultsHandle* to be used to retrieve the remaining matching fields.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlGetFirstFieldValue
                (CSSM_CL_HANDLE CLHandle,
                const CSSM_DATA_PTR Crl,
                const CSSM_OID_PTR CrlField,
                CSSM_HANDLE_PTR ResultsHandle,
                uint32 *NumberOfMatchedFields)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

Crl
&tab;A pointer to the CSSM_DATA structure which contains the CRL from which the first revocation record is to be retrieved.

CrlField
An OID that identifies the field value to be extracted from the Crl.

Output

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

NumberOfMatchedFields
The number of fields that match the CrlField OID.

Return Value

Returns a pointer to a CSSM_DATA structure containing the first field that matched the CrlField. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CrlGetNextFieldValue
CSSM_CL_CrlAbortQuery

CSSM_CL_CrlGetNextFieldValue

Purpose

This function returns the value of a CRL field, when that field occurs multiple times in a CRL. A CRL with repeated fields has multiple field values corresponding to a single OID. A call to the function CSSM_CL_CrlGetFirstFieldValue initiates the process and returns a results handle identifying the CRL from which values are being obtained and the OID corresponding to those values. The CSSM_CL_CrlGetNextFieldValue function can be called repeatedly to obtain these values one at a time.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlGetNextFieldValue (CSSM_CL_HANDLE CLHandle, CSSM_HANDLE ResultsHandle)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

ResultsHandle
The handle that identifies the results of a CRL query.

Return Value

Returns a pointer to a CSSM_DATA structure containing the next field in the CRL that matched the *CrlField* specified in the CL_CrlGetFirstFieldValue function. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CrlGetFirstFieldValue
CSSM_CL_CrlAbortQuery

CSSM_CL_CrlRemoveCert

Purpose

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 using the function CSSM_CL_CrlSign. Once the CRL has been signed, entries can not be added or removed.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlRemoveCert
                      (CSSM_CL_HANDLE CLHandle,
                      const CSSM_DATA_PTR Cert,
                      const CSSM_DATA_PTR OldCrl)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

Cert
A pointer to the CSSM_DATA structure containing the certificate to be reinstated.

OldCrl
A pointer to the CSSM_DATA structure containing the CRL from which the certificate is to be removed.

Return Value

A pointer to the CSSM_DATA structure containing the updated CRL. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CrlAddCert

CSSM_CL_CrlSetFields

Purpose

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 (e.g., as in an extension or CRL record), then a NEW field with the specified value is added to the CRL.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlSetFields
            (CSSM_CL_HANDLE CLHandle,
            const CSSM_FIELD_PTR CrlTemplate,
            uint32 NumberOfFields,
            const CSSM_DATA_PTR OldCrl)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

CrlHandle
Any array of field OID/value pairs containing the values to initialize the CRL attribute fields.

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

OldCrl
The CRL to be updated with the new attribute values. The CRL must be unsigned and available for update.

Return Value

A pointer to the modified, unsigned CRL. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

CSSM_CL_CrlSign

Purpose

This function signs the fields of the CRL indicated in the *SignScope* parameter, in accordance with the specified cryptographic context. Once the CRL has been signed it may not be modified. This means that entries cannot be added or removed from the CRL through application of the CSSM_CL_CrlAddCert or CSSM_CL_CrlRemoveCert operations. A signed CRL can be verified, applied to a data store, and searched for values.

Format

```
CSSM_DATA_PTR CSSMAPI CSSM_CL_CrlSign
                (CSSM_CL_HANDLE CLHandle,
                CSSM_CC_HANDLE CCHandle,
                const CSSM_DATA_PTR UnsignedCrl,
                const CSSM_DATA_PTR SignerCert,
                const CSSM_FIELD_PTR SignScope,
                uint32 ScopeSize)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

CCHandle
The handle that describes the context of this cryptographic operation.

UnsignedCrl
A pointer to the CSSM_DATA structure containing the CRL to be signed.

SignerCert
A pointer to the CSSM_DATA structure containing the certificate to be used to sign the CRL.

ScopeSize
The number of entries in the sign scope list. If the signing scope is not specified, the input scope size must be zero.

Input/optional

SignScope
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 CL module includes a default set of CRL fields in the signing process.

Return Value

A pointer to the CSSM_DATA structure containing the signed CRL. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CrlVerify

CSSM_CL_CrlVerify

Purpose

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.

Format

```
CSSM_BOOL CSSMAPI CSSM_CL_CrlVerify
            (CSSM_CL_HANDLE CLHandle,
            CSSM_CC_HANDLE CCHandle,
            const CSSM_DATA_PTR CrlToBeVerified,
            const CSSM_DATA_PTR SignerCert,
            const CSSM_FIELD_PTR VerifyScope,
            uint32 ScopeSize)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

CCHandle
The handle that describes the context of this cryptographic operation.

CrlToBeVerified
A pointer to the CSSM_DATA structure containing the CRL to be verified.

SignerCert
A pointer to the CSSM_DATA structure containing the certificate used to sign the CRL.

ScopeSize
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.

Input/optional

VerifyScope
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 CL module assumes that a default set of fields were used in the signing process, and those same fields are used in the verification process.

Return Value

A CSSM_TRUE return value signifies that the CRL verifies successfully. When CSSM_FALSE is returned, either the CRL verified unsuccessfully or an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_CL_CrlSign

CSSM_CL_IsCertInCrl

Purpose

This function searches the CRL for a record corresponding to the certificate. The CRL itself may be signed or unsigned. Each entry within the CRL is signed by the revoker's certificate, hence an unsigned list can be validly searched for individually signed CRL entries.

Format

```
CSSM_BOOL CSSMAPI CSSM_CL_IsCertInCrl (CSSM_CL_HANDLE CLHandle,const CSSM_DATA_PTR Cert,
                                       const CSSM_DATA_PTR Crl)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

Cert
A pointer to the CSSM_DATA structure containing the certificate to be located.

Crl
A pointer to the CSSM_DATA structure containing the CRL to be searched.

Return Value

A CSSM_TRUE return value signifies that the certificate is in the CRL. When CSSM_FALSE is returned, either the certificate is not in the CRL or an error has occurred. Use CSSM_GetError to obtain the error code.

Extensibility Functions

This describes the extensibility function interface for CL.

CSSM_CL_PassThrough

Purpose

This function allows applications to call CL module-specific operations. Such operations may include queries or services that are specific to the domain represented by the CL module.

Format

```
void * CSSMAPI CSSM_CL_PassThrough
                 (CSSM_CL_HANDLE CLHandle,
                 CSSM_CC_HANDLE CCHandle,
                 uint32 PassThroughId,
                 const void *InputParams)
```

Parameters

Input

CLHandle
The handle that describes the CL module used to perform this function.

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

InputParams
A pointer to a module, implementation-specific structures containing parameters to be interpreted in a function-specific manner by the requested CL module. This parameter can be used as a pointer to an array of CSSM_DATA structures.

Input/optional

CCHandle
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.

Return Value

A pointer to a module, implementation-specific structure containing the output from the passthrough function. The output data must be interpreted by the calling application based on externally available information. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Chapter 16. Data Storage Library Services API

The primary purpose of a Data Storage Library (DL) module is to provide persistent storage of security-related objects including certificates, Certificate Revocation Lists (CRLs), keys, and policy objects. A DL module is responsible for the creation and accessibility of one or more data stores. A single DL module can be tightly tied to a Certificate Library (CL) and/or Trust Policy (TP) module, or can be independent of all other module types. A single data store can contain a single object type in one format, a single object type in multiple formats, or multiple object types. The persistent repository can be local or remote; for example, a DL can provide client access to a remote directory/storage service.

OCSF stores and manages meta-information about a DL in the OCSF registry. This information describes the storage and retrieval capabilities of a DL. Applications can query the OCSF registry to obtain information about the available DLs and attach to a DL that provides the needed services. Some DL services can acquire and store meta-information about each of the data stores it manages. When this information is available it is stored in the OCSF registry. Not all DL service providers can supply this information.

The DL APIs define a data storage model that can be implemented using a custom storage device, a traditional local or remote file system service, a Database Management System (DBMS) package, or a complete (local or remote) information management system. The abstract data model defined by the DL APIs partitions all values stored in a data record into two categories: one or more mutable attributes and one opaque data object. The attribute values can be directly manipulated by the application and the DL module. Values stored within the opaque data object must be accessed using parsing functions. For example, a DL that stores certificates cannot interpret the format of those certificates. A set of parsing functions such as those defined in a CL module can be used to parse the opaque certificate object. The DL module defines a default set of parsing functions. An application can define a OCSF module to be used for parsing or can define its own set of parsing functions to be used during a data storage session.

Data Structures

This describes the DL data structures.

Note: Some application interfaces use data structures defined by other OCSF services. Those data structures are defined with those particular OCSF services.

CSSM_DB_ACCESS_TYPE

This structure indicates a user's desired level of access to a data store.

```
typedef struct cssm_db_access_type {
    CSSM_BOOL ReadAccess;
    CSSM_BOOL WriteAccess;
    CSSM_BOOL PrivilegedMode; /* versus user mode */
    CSSM_BOOL Asynchronous; /* versus synchronous */
} CSSM_DB_ACCESS_TYPE, *CSSM_DB_ACCESS_TYPE_PTR;
```

Definitions:

ReadAccess

A Boolean indicating that the user requests read access.

WriteAccess
A Boolean indicating that the user requests write access.
PrivilegedMode
A Boolean indicating that the user requests privileged operations.
Asynchronous
A Boolean indicating that the user requests asynchronous access.

CSSM_DB_ATTRIBUTE_DATA

This data structure holds an attribute value that can be stored in an attribute field of a persistent record. The structure contains a value for the data item and a reference to the meta-information (typing information and schema information) associated with the attribute.

```
typedef struct cssm_db_attribute_data {
    CSSM_DB_ATTRIBUTE_INFO Info;
    CSSM_DATA Value;
} CSSM_DB_ATTRIBUTE_DATA, *CSSM_DB_ATTRIBUTE_DATA_PTR;
```

Definitions:

Info A reference to the meta-information/schema describing this attribute in relationship to the data store at large.

Value The data-present value assigned to the attribute.

CSSM_DB_ATTRIBUTE_INFO

This data structure describes an attribute of a persistent record. The description is part of the schema information describing the structure of records in a data store. The description includes the format of the attribute name and the attribute name itself. The attribute name implies the underlying data type of a value that may be assigned to that attribute.

```
typedef struct cssm_db_attribute_info {
    CSSM_DB_ATTRIBUTE_NAME_FORMAT AttributeNameFormat;
    union {
        char * AttributeName; /* eg. "record label" */
        CSSM_OID AttributeID; /* eg. CSSMOID_RECORDLABEL */
        uint32 AttributeNumber;
#ifndef _MVS_
    };
#else
    /* Use the CDSA Version 2.0 definition instead of the anonymous union of
       the Version 1.x spec which unfortunately is not ANSI-C compatible. */

    } Label;
#endif
} CSSM_DB_ATTRIBUTE_INFO, *CSSM_DB_ATTRIBUTE_INFO_PTR;
```

Definitions:

AttributeNameFormat
Indicates which of the three formats was selected to represent the attribute name.

AttributeName
A character string representation of the attribute name.

AttributeID
A DER-encoded Object Identifier (OID) representation of the attribute name.

AttributeNumber
A numeric representation of the attribute name.

CSSM_DB_ATTRIBUTE_NAME_FORMAT

This enumerated list defines three formats used to represent an attribute name. The name can be represented by a character string in the native string encoding of the platform, by a number, or the name can be represented by an opaque OID structure that is interpreted by the DL module.

```
typedef enum cssm_db_attribute_name_format {
    CSSM_DB_ATTRIBUTE_NAME_AS_STRING = 0,
    CSSM_DB_ATTRIBUTE_NAME_AS_OID = 1,
    CSSM_DB_ATTRIBUTE_NAME_AS_NUMBER = 2
} CSSM_DB_ATTRIBUTE_NAME_FORMAT,*CSSM_DB_ATTRIBUTE_NAME_FORMAT_PTR;
```

CSSM_DB_CERTRECORD_SEMANTICS

These bit-masks define a list of usage semantics for how certificates may be used. It is anticipated that additional sets of bit-masks will be defined listing the usage semantics of how other record types can be used, such as CRL record semantics, key record semantics, policy record semantics, etc.

```
#define CSSM_DB_CERT_USE_ROOT    0x00000001 /* a self-signed root cert */
#define CSSM_DB_CERT_USE_TRUSTED 0x00000002 /* re-issued locally */
#define CSSM_DB_CERT_USE_SYSTEM  0x00000004 /* contains CSSM system cert */
#define CSSM_DB_CERT_USE_OWNER   0x00000008 /* private key is owned by the
system's user */
#define CSSM_DB_CERT_USE_REVOKED 0x00000010 /* revoked cert - used w\ CRL APIs */
#define CSSM_DB_CERT_SIGNING     0x00000011 /* use cert for signing only */
#define CSSM_DB_CERT_PRIVACY     0x00000012 /* use cert for encryption only */
```

CSSM_DB_CONJUNCTIVE

These are the conjunctive operations that can be used when specifying a selection criterion.

```
typedef enum cssm_db_conjunctive{
    CSSM_DB_NONE = 0,
    CSSM_DB_AND = 1,
    CSSM_DB_OR = 2
} CSSM_DB_CONJUNCTIVE, *CSSM_DB_CONJUNCTIVE_PTR;
```

CSSM_DB_HANDLE

A unique identifier for an open data store.

```
typedef uint32 CSSM_DB_HANDLE/* Data Store Handle */
```

CSSM_DB_INDEXED_DATA_LOCATION

This enumerated list defines where within a record the indexed data values reside. Indexes can be constructed on attributes or on fields within the opaque object in the record. CSSM_DB_INDEX_ON_UNKNOWN indicates that the logical location of the index value between these two categories is unknown.

```
typedef enum cssm_db_indexed_data_location {
    CSSM_DB_INDEX_ON_UNKNOWN = 0,
    CSSM_DB_INDEX_ON_ATTRIBUTE = 1,
    CSSM_DB_INDEX_ON_RECORD = 2
} CSSM_DB_INDEXED_DATA_LOCATION;
```

CSSM_DB_INDEX_INFO

This structure contains the meta-information or schema description of an index defined on an attribute. The description includes the type of index (e.g., unique key or nonunique key), the logical location of the indexed attribute in the OCSF record (e.g., an attribute, a field within the opaque object in the record, or unknown), and the meta-information on the attribute itself.

```
typedef struct cssm_db_index_info {
    CSSM_DB_INDEX_TYPE IndexType;
    CSSM_DB_INDEXED_DATA_LOCATION IndexedDataLocation;
    CSSM_DB_ATTRIBUTE_INFO Info;
} CSSM_DB_INDEX_INFO, *CSSM_DB_INDEX_INFO_PTR;
```

Definitions:

IndexType
: A CSSM_DB_INDEX_TYPE.

IndexedDataLocation
: A CSSM_DB_INDEXED_DATA_LOCATION.

Info
: The meta-information description of the attribute being indexed.

CSSM_DB_INDEX_TYPE

This enumerated list defines two types of indexes: indexes with unique values (i.e., primary database keys) and indexes with nonunique values. These values are used when creating a new data store and defining the schema for that data store.

```
typedef enum cssm_db_index_type {
    CSSM_DB_INDEX_UNIQUE = 0,
    CSSM_DB_INDEX_NONUNIQUE = 1
} CSSM_DB_INDEX_TYPE;
```

CSSM_DBINFO

This structure contains the meta-information about an entire data store. The description includes the types of records stored in the data store, the attribute schema for each record type, the index schema for all indexes over records in the data store, the type of authentication mechanism used to gain access to the data store, and other miscellaneous information used by the DL module to manage the data store in a secure manner.

```
typedef struct cssm_dbInfo {
    uint32 NumberOfRecordTypes;
    CSSM_DB_PARSING_MODULE_INFO_PTR DefaultParsingModules;
    CSSM_DB_RECORD_ATTRIBUTE_INFO_PTR RecordAttributeNames;
    CSSM_DB_RECORD_INDEX_INFO_PTR RecordIndexes;

    /* access restrictions for opening this data store */
    CSSM_USER_AUTHENTICATION_MECHANISM AuthenticationMechanism;

    /* transparent integrity checking options for this data store */
    CSSM_BOOL RecordSigningImplemented;
    CSSM_DATA SigningCertificate;
    CSSM_GUID SigningCsp;
    /* additional information */
    CSSM_BOOL IsLocal;
    char *AccessPath; /* URL, dir path, etc */
    void *Reserved;
} CSSM_DBINFO, *CSSM_DBINFO_PTR;
```

Definitions:

NumberOfRecordTypes
: The number of distinct record types stored in this data store.

DefaultParsingModules
: A pointer to a list of (record-type, Globally Unique ID (GUID)) pairs which define the default parsing module for each record type.

RecordAttributeNames
: The meta-information (schema) about the attributes associated with each record type that can be stored in this data store.

RecordIndexes
: The meta-information (schema) about the indexes that are defined over each of the record types that can be stored in this data store.

AuthenticationMechanism
: Defines the authentication mechanism required when accessing this data store.

RecordSigningImplemented
: A flag indicating whether or not the DL module provides record integrity service based on digital signaturing of the data store records.

SigningCertificate
: The certificate used to sign data store records when the transparent record integrity option is in effect.

SigningCsp
The GUID for the Cryptographic Service Provider (CSP) to be used to sign data store records when the transparent record integrity option is in effect.

IsLocal Indicates whether the physical data store is local.

AccessPath
A character string describing the access path to the data store, such as a Uniform Resource Locator (URL), a file system pathname, a remote directory service name, etc.

Reserved
Reserved for future use.

CSSM_DB_OPERATOR

These are the logical operators that can be used when specifying a selection predicate.

```
typedef enum cssm_db_operator {
    CSSM_DB_EQUAL = 0,
    CSSM_DB_NOT_EQUAL = 1,
    CSSM_DB_APPROX_EQUAL = 2,
    CSSM_DB_LESS_THAN = 3,
    CSSM_DB_GREATER_THAN = 4,
    CSSM_DB_EQUALS_INITIAL_SUBSTRING = 5,
    CSSM_DB_EQUALS_ANY_SUBSTRING = 6,
    CSSM_DB_EQUALS_FINAL_SUBSTRING = 7,
    CSSM_DB_EXISTS = 8
} CSSM_DB_OPERATOR, *CSSM_DB_OPERATOR_PTR;
```

CSSM_DB_PARSING_MODULE_INFO

This structure aggregates the GUID of a default parsing module with the record type that it parses. A parsing module can parse multiple record types. The same GUID would be repeated with each record type parsed by the module.

```
typedef struct cssm_db_parsing_module_info {
    CSSM_DB_RECORDTYPERecordType;
    CSSM_GUIDModule;
} CSSM_DB_PARSING_MODULE_INFO, *CSSM_DB_PARSING_MODULE_INFO_PTR;
```

Definitions:

RecordType
The type of record parsed by the module specified by GUID.

Module
A GUID identifying the default parsing module for the specified record type.

CSSM_DB_RECORD_ATTRIBUTE_DATA

This structure aggregates the actual data values for all of the attributes in a single record.

```
typedef struct cssm_db_record_attribute_data {
    CSSM_DB_RECORDTYPE DataRecordType;
    uint32 SemanticInformation;
    uint32 NumberOfAttributes;
    CSSM_DB_ATTRIBUTE_DATA_PTR AttributeData;
} CSSM_DB_RECORD_ATTRIBUTE_DATA, *CSSM_DB_RECORD_ATTRIBUTE_DATA_PTR;
```

Definitions:

DataRecordType
A CSSM_DB_RECORDTYPE.

SemanticInformation
A bit-mask of type CSSM_XXXRECORD_SEMANTICS defining how the

record can be used. Currently, these bit-masks are defined only for certificate records (CSSM_CERTRECORD_SEMANTICS). For all other record types, a bit-mask of zero must be used or a set of semantically meaningful masks must be defined.

NumberOfAttributes
The number of attributes in the record of the specified type.

AttributeData
A list of attribute name/value pairs.

CSSM_DB_RECORD_ATTRIBUTE_INFO

This structure contains the meta-information or schema information about all of the attributes in a particular record type. The description specifies the record type, the number of attributes in the record type, and a type information for each attribute.

```
typedef struct cssm_db_record_attribute_info {
    CSSM_DB_RECORDTYPE DataRecordType;
    uint32 NumberOfAttributes;
    CSSM_DB_ATTRIBUTE_INFO_PTR AttributeInfo;
} CSSM_DB_RECORD_ATTRIBUTE_INFO, *CSSM_DB_RECORD_ATTRIBUTE_INFO_PTR;
```

Definitions:

DataRecordType
A CSSM_DB_RECORDTYPE.

NumberOfAttributes
The number of attributes in a record of the specified type.

AttributeInfo
A list of pointers to the type (schema) information for each of the attributes.

CSSM_DB_RECORD_INDEX_INFO

This structure contains the meta-information or schema description of the set of indexes defined on a single record type. The description includes the type of the record, the number of indexes, and the meta-information describing each index.

```
typedef struct cssm_db_record_index_info {
    CSSM_DB_RECORDTYPE DataRecordType;
    uint32 NumberOfIndexes;
    CSSM_DB_INDEX_INFO_PTR IndexInfo;
} CSSM_DB_RECORD_INDEX_INFO, *CSSM_DB_RECORD_INDEX_INFO_PTR;
```

Definitions:

DataRecordType
A CSSM_DB_RECORDTYPE.

NumberOfIndexes
The number of indexes defined on the records of the given type.

IndexInfo
An array of pointers to the meta-description of each index defined over the specified record type.

CSSM_DB_RECORD_PARSING_FNTABLE

This structure defines the three prototypes for functions that can parse the opaque data object stored in a record. It is used in the CSSM_DbSetRecordParsingFunctions function to override the default parsing module for a given record type. The DL module developer designates the default parsing module for each record type stored in the data store.

```
typedef struct cssm_db_record_parsing_fntable {
    CSSM_DATA_PTR (CSSMAPI *RecordGetFirstFieldValue)
        (CSSM_HANDLE Handle,
        CSSM_DB_RECORDTYPE RecordType,
        const CSSM_DATA_PTR Data,
        const CSSM_OID_PTR DataField,
        CSSM_HANDLE_PTR ResultsHandle,
        uint32 *NumberOfMatchedFields);
    CSSM_DATA_PTR (CSSMAPI *RecordGetNextFieldValue)
        (CSSM_HANDLE Handle,
        CSSM_HANDLE ResultsHandle);
    CSSM_RETURN (CSSMAPI *RecordAbortQuery)
        (CSSM_HANDLE Handle,
        CSSM_HANDLE ResultsHandle);
} CSSM_DB_RECORD_PARSING_FNTABLE, *CSSM_DB_RECORD_PARSING_FNTABLE_PTR;
```

Definitions:

******RecordGetFirstFieldValue*

&tab;A function to retrieve the value of a field in the opaque object. The field is specified by attribute name. The results handle holds the state information required to retrieve subsequent values having the same attribute name.

******RecordGetNextFieldValue*

&tab;A function to retrieve subsequent values having the same attribute name from a record parsed by the first function in this table.

******RecordAbortQuery*

Stop subsequent retrieval of values having the same attribute name from within the opaque object.

CSSM_DB_RECORDTYPE

This enumerated list defines the categories of persistent security-related objects that can be managed by a DL module. These categories are in one-to-one correspondence with types of records that can be managed by a DL module.

```
typedef enum cssm_db_recordtype {
    CSSM_DL_DB_RECORD_GENERIC = 0,
    CSSM_DL_DB_RECORD_CERT = 1,
    CSSM_DL_DB_RECORD_CRL = 2,
    CSSM_DL_DB_RECORD_PUBLIC_KEY = 3,
    CSSM_DL_DB_RECORD_PRIVATE_KEY = 4,
CSSM_DL_DB_RECORD_SYMMETRIC_KEY = 5,
    CSSM_DL_DB_RECORD_POLICY = 6,
    CSSM_DS_DB_PKICA = 7,
    CSSM_DL_DB_PKIUSER = 8,
    CSSM_DL_DB_X_CERT_PAIR = 9,
    CSSM_DL_DB_CRL_DISTRIBUTION_POINT = 10,
    CSSM_DL_DB_AUTHORITY_REVOCATION_LIST = 11,
    CSSM_DL_DB_DELTA_REVOCATION_LIST = 12
} CSSM_DB_RECORDTYPE;
```

CSSM_DB_UNIQUE_RECORD

This structure contains an index descriptor and a module-defined value. The index descriptor may be used by the module to enhance the performance when locating the record. The module-defined value must uniquely identify the record. For a DBMS, this may be the record data. For a Public-Key Cryptographic Standard DL, this may be an object handle. Alternately, the DL may have a module-specific scheme for identifying data that has been inserted or retrieved.

```
typedef struct cssm_db_unique_record {
    CSSM_DB_INDEX_INFO RecordLocator;
    CSSM_DATA RecordIdentifier;
} CSSM_DB_UNIQUE_RECORD, *CSSM_DB_UNIQUE_RECORD_PTR;
```

Definitions:

RecordLocator
The information describing how to locate the record efficiently.
RecordIdentifier
A module-specific identifier which will allow the DL to locate this record.

CSSM_DL_DB_HANDLE

This data structure holds a pair of handles, one for a DL, and another for a data store that is opened and being managed by the DL.

```
typedef struct cssm_dl_db_handle {
    CSSM_DL_HANDLE DLHandle;
    CSSM_DB_HANDLE DBHandle;
} CSSM_DL_DB_HANDLE, *CSSM_DL_DB_HANDLE_PTR;
```

Definitions:

DLHandle
Handle of an attached module that provides DL services.

DBHandle
Handle of an open data store that is currently under the management of the DL module specified by the DLHandle.

CSSM_DL_DB_LIST

This data structure defines a list of handle pairs of (DL handle, data store handle).

```
typedef struct cssm_dl_db_list {
    uint32 NumHandles;
    CSSM_DL_DB_HANDLE_PTR DLDBHandle;
} CSSM_DL_DB_LIST, *CSSM_DL_DB_LIST_PTR;
```

Definitions:
NumHandles
Number of (DL handle, data store handle) pairs in the list.
DLDBHandle
List of (DL handle, data store handle) pairs.

CSSM_CUSTOM_ATTRIBUTES

This structure can be used by DL module developers to define a set of attributes for a custom data store format.

```
typedef void *CSSM_DL_CUSTOM_ATTRIBUTES;
```

CSSM_DL_FFS_ATTRIBUTES

This structure can be used by DL module developers to define a set of attributes for a flat file system data store format.

```
typedef void *CSSM_DL_FFS_ATTRIBUTES;
```

CSSM_DL_HANDLE

A unique identifier for an attached module that provides DL services.

```
typedef uint32 CSSM_DL_HANDLE/* Data Storage Library Handle */
```

CSSM_DL_LDAP_ATTRIBUTES

This structure can be used by DL module developers to define a set of attributes for a Lightweight Directory Access Protocol (LDAP) data store format.

```
typedef void *CSSM_DL_LDAP_ATTRIBUTES;
```

CSSM_DL_ODBC_ATTRIBUTES

This structure can be used by DL module developers to define a set of attributes for an Open Database Connectivity (ODBC) data store format.

```
typedef void *CSSM_DL_ODBC_ATTRIBUTES;
```

CSSM_DL_PKCS11_ATTRIBUTES

Each type of DL module can define its own set of type-specific attributes. This structure contains the attributes that are specific to a data storage device.

```
typedef struct cssm_dl_pkcs11_attributes {
    uint32 DeviceAccessFlags;
} *CSSM_DL_PKCS11_ATTRIBUTES;
```

Definitions:

DeviceAccessFlags
: Specifies the access modes applicable for accessing persistent objects in a data store.

CSSM_DLSUBSERVICE

Three structures are used to contain all of the static information that describes a DL module: cssm_moduleinfo, cssm_serviceinfo, and cssm_dlsubservice. This descriptive information is securely stored in the OCSF registry when the DL module is installed with OCSF. A DL module may implement multiple types of services and organize them as subservices. For example, a DL module supporting two types of remote directory services may organize its implementation into two subservices: one for an X.509 certificate directory and a second for custom enterprise policy data store. Most DL modules will implement exactly one subservice.

Not all DL modules can maintain a summary of managed data stores. In this case, the DL module reports its number of data stores as CSSM_DB_DATASTORES_UNKNOWN. Data stores can (and probably do) exist, but the DL module cannot provide a list of them.

```
#define      CSSM_DB_DATASTORES_UNKNOWN     (-1)
```

The descriptive information stored in these structures can be queried using the function CSSM_GetModuleInfo and specifying the DL module GUID.

```
typedef struct cssm_dlsubservice {
    uint32 SubServiceId;
    CSSM_STRING Description;
    CSSM_DLTYPE Type;
    union {
        CSSM_DL_CUSTOM_ATTRIBUTES CustomAttributes;
        CSSM_DL_LDAP_ATTRIBUTES LdapAttributes;
        CSSM_DL_ODBC_ATTRIBUTES OdbcAttributes;
        CSSM_DL_PKCS11_ATTRIBUTES Pkcs11Attributes;
        CSSM_DL_FFS_ATTRIBUTES FfsAttributes;
    } Attributes;

    CSSM_DL_WRAPPEDPRODUCT_INFO WrappedProduct;
    CSSM_USER_AUTHENTICATION_MECHANISM AuthenticationMechanism;

    /* meta-information about the query support provided by the module */
    uint32 NumberOfRelOperatorTypes;
    CSSM_DB_OPERATOR_PTR RelOperatorTypes;
    uint32 NumberOfConjOperatorTypes;
    CSSM_DB_CONJUNCTIVE_PTR ConjOperatorTypes;
    CSSM_BOOL QueryLimitsSupported;
    /* meta-information about the encapsulated data stores (if known) */
    uint32 NumberOfDataStores;
    CSSM_NAME_LIST_PTR DataStoreNames;
    CSSM_DBINFO_PTR DataStoreInfo;
```

```
    /* additional information */
    void *Reserved;
} CSSM_DLSUBSERVICE, *CSSM_DLSUBSERVICE_PTR;
```

Definitions:

SubServiceId
: A unique, identifying number for the subservice described in this structure.

Description
: A string containing a descriptive name or title for this subservice.

Type
: An identifier for the type of underlying data store the DL module uses to provide persistent storage.

Attributes
: A structure containing attributes that define additional parameter values specific to the DL module type.

WrappedProduct
: Pointer to a CSSM_DL_WRAPPEDPRODUCT_INFO structure describing a product that is wrapped by the DL module.

AuthenticationMechanism
: Defines the authentication mechanism required when using this DL module. This authentication mechanism is distinct from the authentication mechanism (specified in a cssm_dbInfo structure) required to access a specific data store.

NumberOfRelOperatorTypes
: The number of distinct relational operators the DL module accepts in selection queries for retrieving records from its managed data stores.

RelOperatorTypes
: The list of specific relational operators that can be used to formulate selection predicates for queries on a data store. The list contains NumberOfRelOperatorTypes operators.

NumberOfConjOperatorTypes
: The number of distinct conjunctive operators the DL module accepts in selection queries for retrieving records from its managed data stores.

ConjOperatorTypes
: A list of specific conjunctive operators that can be used to formulate selection predicates for queries on a data store. The list contains NumberOfConjOperatorTypes operators.

QueryLimitsSupported
: A Boolean indicating whether query limits are effective when the DL module executes a query.

NumberOfDataStores
: The number of data stores managed by the DL module. This information may not be known by the DL module, in which case this value will equal CSSM_DB_DATASTORES_UNKNOWN.

DataStoreNames
: A list of names of the data stores managed by the DL module. This information may not be known by the DL module and hence may not be available. The list contains NumberOfDataStores entries.

DataStoreInfo
: A list of pointers to the meta-information (schema) for each data store

managed by the DL module. This information may not be known in advance by the DL module and hence may not be available through this structure. The list contains NumberOfDataStores entries.

Reserved
Reserved for future use.

CSSM_DLTYPE

This enumerated list defines the types of underlying DBMSs that can be used by the DL module to provide services. It is the option of the DL module to disclose this information.

```
typedef enum cssm_dltype {
    CSSM_DL_UNKNOWN = 0,
    CSSM_DL_CUSTOM = 1,
    CSSM_DL_LDAP = 2,
    CSSM_DL_ODBC = 3,
    CSSM_DL_PKCS11 = 4,
    CSSM_DL_FFS = 5,/* flat file system or fast file system */
    CSSM_DL_MEMORY = 6,
    CSSM_DL_REMOTEDIR = 7
} CSSM_DLTYPE, *CSSM_DLTYPE_PTR;
```

CSSM_DL_WRAPPEDPRODUCTINFO

This structure lists the set of data store services used by the DL module to implement its services. The DL module vendor is not required to provide this information, but may choose to do so. For example, a DL module that uses a commercial DBMS can record information about that product in this structure. Another example is a DL module that supports certificate storage through an X.500 certificate directory server. The DL module can describe the X.500 directory service in this structure.

```
typedef struct cssm_dl_wrappedproductinfo {
    CSSM_VERSION StandardVersion;
    CSSM_STRING StandardDescription;
    CSSM_VERSION ProductVersion;
    CSSM_STRING ProductDescription;
    CSSM_STRING ProductVendor;
    uint32 ProductFlags;
} CSSM_DL_WRAPPEDPRODUCT_INFO, *CSSM_DL_WRAPPEDPRODUCT_INFO_PTR;
```

Definitions:

StandardVersion
If this product conforms to an industry standard, this is the version number of that standard.

StandardDescription
If this product conforms to an industry standard, this is a description of that standard.

ProductVersion
Version number information for the actual product version used in this version of the DL module.

ProductDescription
A string describing the product.

ProductVendor
The name of the product vendor.

ProductFlags
A bit-mask enumerating selectable features of the database service that the DL module uses in its implementation.

CSSM_NAME_LIST

```
typedef struct cssm_name_list {
    uint32 NumStrings;
    char** String;
} CSSM_NAME_LIST, *CSSM_NAME_LIST_PTR;
```

CSSM_QUERY

This structure holds a complete specification of a query to select records from a data store.

```
typedef struct cssm_query {
    CSSM_DB_RECORDTYPE RecordType;
    CSSM_DB_CONJUNCTIVE Conjunctive;
    uint32 NumSelectionPredicates;
    CSSM_SELECTION_PREDICATE_PTR SelectionPredicate;
    CSSM_QUERY_LIMITS QueryLimits;
    CSSM_QUERY_FLAGS QueryFlags;
} CSSM_QUERY, *CSSM_QUERY_PTR;
```

Definitions:

RecordType
: Specifies the type of record to be retrieved from the data store.

Conjunctive
: The conjunctive operator to be used in constructing the selection predicate for the query.

NumSelectionPredicates
: The number of selection predicates to be connected by the specified conjunctive operator to form the query.

SelectionPredicate
: The list of selection predicates to be combined by the conjunctive operator to form the data store query.

QueryLimits
: Defines the time and space limits for processing the selection query. The constant values CSSM_QUERY_TIMELIMIT_NONE and CSM_QUERY_SIZELIMIT_NONE should be used to specify no limit on the resources used in processing the query.

QueryFlags
: An integer that indicates the return format of the key data. This integer is represented by CSSM_QUERY_RETURN_DATA. When CSSM_QUERY_RETURN_DATA is 1, the key record is returned in Common Data Security Architecture (CDSA) format. When CSSM_QUERY_RETURN_DATA is 0, the information is returned in raw format (a format native to the individual CSP, such as BSAFE or PKCS#11).

CSSM_QUERY_LIMITS

This structure defines the time and space limits a caller can set to control early termination of the execution of a data store query. The constant values CSSM_QUERY_TIMELIMIT_NONE and CSM_QUERY_SIZELIMIT_NONE should be used to specify no limit on the resources used in processing the query. These limits are advisory. Not all DL modules recognize and act upon the query limits set by a caller.

```
#define     CSSM_QUERY_TIMELIMIT_NONE     0
#define     CSSM_QUERY_SIZELIMIT_NONE     0
```

```
typedef struct cssm_query_limits {
    uint32 TimeLimit;
    uint32 SizeLimit;
} CSSM_QUERY_LIMITS, *CSSM_QUERY_LIMITS_PTR;
```

Definitions:

TimeLimit
: Defines the maximum number of seconds of resource time that should be expended performing a query operation. The constant value CSSM_QUERY_TIMELIMIT_NONE means no time limit is specified.

SizeLimit
: Defines the maximum number of records that should be retrieved in response to a single query. The constant value CSSM_QUERY_SIZELIMIT_NONE means no space limit is specified.

CSSM_SELECTION_PREDICATE

This structure defines the selection predicate to be used for database queries.

```
typedef struct cssm_selection_predicate {
    CSSM_DB_OPERATOR DbOperator;
    CSSM_DB_ATTRIBUTE_DATA Attribute;
} CSSM_SELECTION_PREDICATE, *CSSM_SELECTION_PREDICATE_PTR;
```

Definitions:

DbOperator
: The relational operator to be used when comparing a value to the values stored in the specified attribute in the data store.

Attribute
: The meta-information about the attribute to be searched and the attribute value to be used for comparison with values in the data store.

Data Storage Functions

This describes the interfaces for the data storage functions.

CSSM_DL_Authenticate

Purpose

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 credentials apply 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 credentials are applied to a specific data store, the handle for that data store must be specified in the *DLDBHandle* pair.

Format

```
CSSM_RETURN CSSMAPI CSSM_DL_Authenticate
               (CSSM_DL_DB_HANDLE DLDBHandle,
               const CSSM_DB_ACCESS_TYPE_PTR AccessRequest,
               const CSSM_USER_AUTHENTICATION_PTR UserAuthentication)
```

Parameters

Input

DLDBHandle
The handle pair that describes the DL 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
An indicator of the requested access mode for the data store or DL module in general.

UserAuthentication
The caller's credential as required for obtaining authorized access to the data store or to the DL module in general.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

CSSM_DL_DbClose

Purpose

This function closes an open data store.

Format

```
CSSM_RETURN CSSMAPI CSSM_DL_DbClose (CSSM_DL_DB_HANDLE DLDBHandle)
```

Parameters

Input

DLDBHandle
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.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DL_DbOpen

CSSM_DL_DbCreate

Purpose

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. 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 the *OpenParameters*.

Format

```
CSSM_DB_HANDLE CSSMAPI CSSM_DL_DbCreate
                (CSSM_DL_HANDLE DLHandle,
                const char *DbName,
                const CSSM_DBINFO_PTR DBInfo,
                const CSSM_DB_ACCESS_TYPE_PTR AccessRequest,
                const CSSM_USER_AUTHENTICATION_PTR UserAuthentication,
                const void *OpenParameters)
```

Parameters

Input

DLHandle
The handle that describes the DL module used to perform this function.

DbName
The general, external name for the new data store.

DBInfo
A pointer to a structure describing the format/schema of each record type that will be stored in the new data store.

AccessRequest
An indicator of the requested access mode for the data store, such as read-only or read/write.

Input/optional

UserAuthentication
&tab;The caller's credential as required for obtaining access to the data store. If no credentials are required for the specified data store, then user authentication must be NULL.

OpenParameters
A pointer to a module-specific set of parameters required to open the data store.

Return Value

The handle the newly created and open data store. When NULL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DL_DbOpen
CSSM_DL_DbClose
CSSM_DL_DbDelete

CSSM_DL_DbDelete

Purpose

This function deletes all records from the specified data store and removes all state information associated with that data store.

Format

```
CSSM_RETURN CSSMAPI CSSM_DL_DbDelete
              (CSSM_DL_HANDLE DLHandle,
              const char *DbName,
              const CSSM_USER_AUTHENTICATION_PTR UserAuthentication)
```

Parameters

Input

DLHandle
The handle that describes the DL module to be used to perform this function.

DbName
A pointer to the string containing the logical name of the data store.

Input/optional

UserAuthentication
The caller's credentials as required for obtaining access (and consequently deletion capability) to the data store. If no credentials are required for the specified data store, then user authentication must be NULL.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DL_DbCreate
CSSM_DL_DbOpen
CSSM_DL_DbClose

CSSM_DL_DbExport

Purpose

This function exports a copy of the data store records from the source data store to a data container that can be used as the input data source for the CSSM_DL_DbImport function. The DL module may require additional user authentication to determine whether the user is authorized to create a copy of an existing data store.

Format

```
CSSM_RETURN CSSMAPI CSSM_DL_DbExport
                    (CSSM_DL_HANDLE DLHandle,
                    const char *DbDestinationName,
                    const char *DbSourceName,
                    CSSM_BOOL InfoOnly,
                    const CSSM_USER_AUTHENTICATION_PTR UserAuthentication)
```

Parameters

Input

DLHandle
The handle that describes the DL module to be used to perform this function.

DbSourceName
The name of the data store from which the records are to be exported.

DbDestinationName
The name of the destination data container that will contain a copy of the source data store's records.

InfoOnly
A Boolean value indicating what to export. If CSSM_TRUE, export only the DBInfo, which describes the data store. If CSSM_FALSE, export both the DBInfo and all of the records in the specified data store.

Input/optional

UserAuthentication
The caller's credentials as required for authorization to copy a data store. If the DL module requires no additional credentials to perform this operation, then user authentication can be NULL.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DL_DbImport

CSSM_DL_DbGetRecordParsingFunctions

Purpose

This function gets the records parsing function table, which operates on records of the specified type from the specified data store. Three record parsing functions can be returned in the table. The functions can be implemented to parse multiple record types. However, in order to parse multiple record types, multiple calls to CSSM_DL_DbGetRecordParsingFunctions must be made, once for each record type whose parsing functions are required by the caller. The DL module uses these functions to parse the opaque data object stored in a data store record. If no parsing function table has been set for a given record type, then a NULL value is returned.

Format

```
CSSM_DB_RECORD_PARSING_FNTABLE_PTR CSSMAPI CSSM_DL_DbGetRecordParsingFunction
              (CSSM_DL_HANDLE DLHandle,
              const char* DbName,
              CSSM_DB_RECORDTYPE RecordType)
```

Parameters

Input

DLHandle
The handle that describes the DL module to be used to perform this function.

DbName
The name of the data store with which the parsing functions are associated.

RecordType
The record type whose parsing functions are requested by the caller.

Return Value

A pointer to a function table for the parsing function appropriate to the specified record type. When NULL is returned, either no function table has been set for the

specified record type or an error has occurred. Use CSSM_GetError to obtain the error code and determine the reason for the NULL result.

Related Information

CSSM_DL_SetRecordParsingFunctions

CSSM_DL_DbImport

Purpose

This function creates a new data store, or adds to an existing data store, by importing records from the specified data source. It is assumed that the data source contains records exported from a data store using the function CSSM_DL_DbExport.

The *DbDestinationName* parameter specifies the name of a new or existing data store. If a new data store is being created, the *DBInfo* structure provides the meta-information (schema) for the new data store. This structure describes the record attributes and the index schema for the new data store. If the data store already exists, then the existing meta-information (schema) is used. (Dynamic schema evolution is not supported.)

Typically, user authentication is required to create a new data store or to write to an existing data store. An authentication credential is presented to the DL module in the form required by the module. (See the information provided with the DL module for information on the required form.) The resulting data store is not opened as a result of this operation.

Format

```
CSSM_RETURN CSSMAPI CSSM_DL_DbImport
               (CSSM_DL_HANDLE DLHandle,
               const char *DbDestinationName,
               const char *DbSourceName,
               const CSSM_DBINFO_PTR DBInfo,
               CSSM_BOOL InfoOnly,
               const CSSM_USER_AUTHENTICATION_PTR UserAuthentication)
```

Parameters

Input

DLHandle
The handle that describes the DL module to be used to perform this function.

DbDestinationName
The name of the destination data store into which the records will be inserted.

DbSourceName
The name of the data source from which to obtain the records that are added to the data store.

InfoOnly
A Boolean value indicating what to import. If CSSM_TRUE, import only the DBInfo, which describes the data store. If CSSM_FALSE, import both the DBInfo and all of the records exported from a data store.

Input/optional

DBInfo
A data structure containing a detailed description of the meta-information

(schema) for the new data store. If a new data store is being created, then the caller must specify the meta-information (schema), or the data source must include the meta-information required for proper import of the records. If meta-information is supplied by the caller and specified in the data source, then the meta-information provided by the caller overrides the meta-information recorded in the data source.

If the data store exists and records are being added, then this pointer must be NULL. The existing meta-information will be used and the schema cannot be evolved.

UserAuthentication
The caller's credential as required for authorization to create a data store. If the DL module requires no additional credentials to create a new data store, then user authentication can be NULL.

Return Value

A CSSM_OK return value signifies that the function completed successfully and the new data store was created. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DL_DbExport

CSSM_DL_DbOpen

Purpose

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*.

Format

```
CSSM_DB_HANDLE CSSMAPI CSSM_DL_DbOpen
                (CSSM_DL_HANDLE DLHandle,
                const char *DbName,
                const CSSM_DB_ACCESS_TYPE_PTR AccessRequest,
                const CSSM_USER_AUTHENTICATION_PTR UserAuthentication,
                const void *OpenParameters)
```

Parameters

Input

DLHandle
The handle that describes the DL module to be used to perform this function.

DbName
A pointer to the string containing the logical name of the data store.

AccessRequest
An indicator of the requested access mode for the data store, such as read-only or read/write.

Input/optional

UserAuthentication
The caller's credentials as required for obtaining access to the data store. If no credentials are required for the specified data store, then user authentication must be NULL.

OpenParameters
A pointer to a module-specific set of parameters required to open the data store.

Return Value

The handle to the opened data store. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DL_DbClose

CSSM_DL_DbSetRecordParsingFunctions

Purpose

This function sets the record parsing function table, overriding the default parsing module, for records of the specified type in the specified data store. Three record parsing functions can be specified in the table. The functions can be implemented to parse multiple record types. In this case, multiple calls to CSSM_DL_DbSetRecordParsingFunctions must be made, once for each record type that should be parsed using these functions. The DL module uses these functions to parse the opaque data object stored in a data store record. If no parsing function table has been set for a given record type, then the default parsing module is invoked for that record type.

Format

```
CSSM_RETURN CSSMAPI CSSM_DL_DbSetRecordParsingFunctions
               (CSSM_DL_HANDLE DLHandle,
               const char* DbName,
               CSSM_DB_RECORDTYPE RecordType,
               const CSSM_DB_RECORD_PARSING_FNTABLE_PTR FunctionTable)
```

Parameters

Input

DLHandle
The handle that describes the DL module to be used to perform this function.

DbName
The name of the data store with which to associate the parsing functions.

RecordType
One of the record types parsed by the functions specified in the function table.

FunctionTable
The function table referencing the three parsing functions to be used with the data store specified by *DbName*.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DL_GetRecordParsingFunctions

CSSM_DL_GetDbNameFromHandle

Purpose

This function retrieves the data source name corresponding to an opened database handle. A DL module is responsible for allocating the memory required for the list.

Format

```
char * CSSMAPI CSSM_DL_GetDbNameFromHandle (CSSM_DL_DB_HANDLE DLDBHandle)
```

Parameters

Input

DLDBHandle
The handle pair that describes the DL 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, the data store handle must be NULL.

Return Value

Returns a string that contains a data store name. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Data Record Operations

This describes the interfaces for the data record operations.

CSSM_DL_AbortQuery

Purpose

This function terminates the query initiated by CSSM_DL_DataGetFirst or CSSM_DL_DataGetNext, and allows a DL to release all intermediate state information associated with the query.

Format

```
CSSM_RETURN CSSMAPI CSSM_DL_DataAbortQuery (CSSM_DL_DB_HANDLE DLDBHandle,CSSM_HANDLE ResultsHandle)
```

Parameters

Input

DLDBHandle
The handle pair that describes the DL module to be used to perform this function and the open data store from which records were selected by the initiating query.

ResultsHandle
The selection handle returned from the initial query function.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DL_DataGetFirst
CSSM_DL_DataGetNext

CSSM_DL_DataDelete

Purpose

This function removes from the specified data store, the data record specified by *UniqueRecordIdentifier*.

Format

```
CSSM_RETURN CSSMAPI CSSM_DL_DataDelete
              (CSSM_DL_DB_HANDLE DLDBHandle,
              CSSM_DB_RECORDTYPE RecordType,
              const CSSM_DB_UNIQUE_RECORD_PTR UniqueRecordIdentifier)
```

Parameters

Input

DLDBHandle
The handle pair that describes the DL module to be used to perform this function and the open data store from which to delete the specified data record.

UniqueRecordIdentifier
A pointer to a CSSM_DB_UNIQUE_RECORD identifier containing unique identification of the data record to be deleted from the data store. The identifier may be unique only among records of a given type. Once the associated record has been deleted, this unique record identifier can not be used in future references.

Input/optional

RecordType
An indicator of the type of record to be deleted from the data store. The *UniqueRecordIdentifier* may be unique only among records of the same type. If the data store contains only one record type, or the unique identifiers managed are globally unique, then the record type need not be specified.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DL_DataInsert

CSSM_DL_DataGetFirst

Purpose

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. The DL module can use internally managed indexing structures to enhance the performance of the retrieval operation. This function returns the first record that satisfies the query in the list of *Attributes* and the opaque *Data* object. This function also returns a flag indicating whether additional records also satisfied the query and a results handle to be used when retrieving subsequent records. 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.

Format

```
CSSM_DB_UNIQUE_RECORD_PTR CSSMAPI CSSM_DL_DataGetFirst
                (CSSM_DL_DB_HANDLE DLDBHandle,
                const CSSM_QUERY_PTR Query,
                CSSM_HANDLE_PTR ResultsHandle,
                CSSM_BOOL*EndOfDataStore,
                CSSM_DB_RECORD_ATTRIBUTE_DATA_PTR Attributes,
                CSSM_DATA_PTR Data)
```

Parameters

Input

DLDBHandle
The handle pair that describes the DL module to be used to perform this function and the open data store to search for records satisfying the query.

Output

ResultsHandle
This handle should be used to retrieve subsequent records that satisfied this query.

EndOfDataStore
A flag indicating whether a record satisfying this query was available to be retrieved in the current operation. If CSSM_FALSE, then a record was available and was retrieved unless an error condition occurred. If CSSM_TRUE, then all records satisfying the query have been previously retrieved and no record has been returned by this operation.

Attributes
&tab;A list of attributes values (and corresponding meta-information) from the retrieved record.

Data
The opaque object stored in the retrieved record.

Input/optional

Query
The query structure specifying the selection predicates 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 and Data parameter. If no query is specified, the DL module can return the first record in the data store (i.e.,perform sequential retrieval) or return an error).

Return Value

If successful and *EndOfDataStore* is CSSM_FALSE, this function returns a pointer to a CSSM_UNIQUE_RECORD structure containing a unique record locator and the record. If the pointer is NULL and *EndOfDataStore* is CSSM_TRUE, then a normal termination condition has occurred. If the pointer is NULL and *EndOfDataStore* is CSSM_FALSE, then an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DL_DataGetNext
CSSM_DL_DataAbortQuery

CSSM_DL_DataGetNext

Purpose

This function returns the next data record referenced by the *ResultsHandle*. The *ResultsHandle* references a set of records selected by an invocation of the CSSM_DL_DataGetFirst function.

The record values are returned in the *Attributes* and *Data* parameters. A flag indicates whether additional records satisfying the original query remain to be retrieved. The function also returns a unique record identifier for the return record.

Format

```
CSSM_DB_UNIQUE_RECORD_PTR CSSMAPI CSSM_DL_DataGetNext
               (CSSM_DL_DB_HANDLE DLDBHandle,
               CSSM_HANDLE ResultsHandle,
               CSSM_BOOL *EndOfDataStore,
               CSSM_DB_RECORD_ATTRIBUTE_DATA_PTR Attributes,
               CSSM_DATA_PTR Data)
```

Parameters

Input

DLDBHandle
The handle pair that describes the DL module to be used to perform this function and the open data store from which records were selected by the initiating query.

Output

ResultsHandle
The handle identifying a set of records retrieved by a query executed by the CSSM_DL_DataGetFirst function.

EndOfDataStore
A flag indicating whether a record satisfying this query was available to be retrieved in the current operation. If CSSM_FALSE, then a record was available and was retrieved unless an error condition occurred. If CSSM_TRUE, then all records satisfying the query have been previously retrieved and no record has been returned by this operation.

Attributes
A list of attributes values (and corresponding meta-information) from the retrieved record.

Data
The opaque object stored in the retrieved record.

Return Value

If successful and *EndOfDataStore* is CSSM_FALSE, this function returns a pointer to a CSSM_UNIQUE_RECORD structure containing a unique record locator and the record. If the pointer is NULL and *EndOfDataStore* is CSSM_TRUE, then a normal termination condition has occurred. If the pointer is NULL and *EndOfDataStore* is CSSM_FALSE, then an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DL_DataGetFirst
CSSM_DL_DataAbortQuery

CSSM_DL_DataInsert

Purpose

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* fields. The attribute value list contains zero or more attribute values. The DL modules can assume default values for unspecified attribute values or can return an error condition when required attributes values are not specified by the caller. The *Data* is an opaque object to be stored in the new data record.

Format

```
CSSM_DB_UNIQUE_RECORD_PTR CSSMAPI CSSM_DL_DataInsert
                (CSSM_DL_DB_HANDLE DLDBHandle,
                CSSM_DB_RECORDTYPE RecordType,
                const CSSM_DB_RECORD_ATTRIBUTE_DATA_PTR Attributes,
                const CSSM_DATA_PTR Data)
```

Parameters

Input

DLDBHandle
The handle pair that describes the DL module to be used to perform this function and the open data store in which to insert the new data record.

RecordType
Indicates the type of data record being added to the data store.

Input/optional

Attributes
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 DL module can assume default values for those attributes that are not assigned values by the caller or may return an error. If the specified record type does not contain any attributes, this parameter must be NULL.

Data
A pointer to the CSSM_DATA structure that 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.

Return Value

A pointer to a CSSM_DB_UNIQUE_RECORD_POINTER containing a unique identifier associated with the new record. This unique identifier structure can be used in future references to this record. When NULL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DL_DataDelete

CSSM_DL_FreeUniqueRecord

Purpose

Frees the pointer to a CSSM_DB_UNIQUE_RECORD data structure. The record itself and the data it contains is unchanged. To delete the record, call CSSM_DL_DataDelete before invoking CSSM_DL_FreeUniqueRecord.

Format

```
CSSM_RETURN CSSMAPI CSSM_DL_FreeUniqueRecord(CSSM_DL_DB_HANDLEDLDBHandle,
                                             CSSM_DB_UNIQUE_RECORD_PTR UniqueRecord)
```

Parameters

Input

DLDBHandle
The handle pair that describes the DL module to be used to perform this &tab;function and the open data store in which to insert the new data record.

UniqueRecord
Pointer to a unique record.

Return Value

A CSSM_OK return value signifies that the unique record pointer was freed. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Related Information

CSSM_DL_DataDelete
CSSM_DL_DataInsert
CSSM_DL_DataGetFirst
CSSM_DL_DataGetNext

Extensibility Functions

This describes the extensibility function for the data storage library.

CSSM_DL_PassThrough

Purpose

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.

Format

```
void * CSSMAPI CSSM_DL_PassThrough (CSSM_DL_DB_HANDLE DLDBHandle,uint32 PassThroughId,
                                   const void *InputParams)
```

Parameters

Input

DLDBHandle
: The handle pair that describes the 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
: An identifier assigned by a DL module to indicate the exported function to be performed.

InputParams
: A pointer to a module, implementation-specific structure containing parameters to be interpreted in a function-specific manner by the requested DL module. This parameter can be used as a pointer to an array of CSSM_DATA_PTRs.

Return Value

A pointer to a module, implementation-specific structure containing the output from the passthrough function. The output data must be interpreted by the calling application based on externally-available information. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Chapter 17. OCSF Error Handling

This describes the error handling features in OCSF that provide a consistent mechanism across all layers of OCSF for returning errors to the caller. All OCSF API functions return one of these:

- CSSM_RETURN - An enumerated type consisting of CSSM_OK and CSSM_FAIL. If it is CSSM_FAIL, an error code indicating the reason for failure can be obtained by calling CSSM_GetError.
- CSSM_BOOL - OCSF functions returning this data type return either CSSM_TRUE or CSSM_FALSE. If the function returns CSSM_FALSE, an error code may be available (but not always) by calling CSSM_GetError.
- A pointer to a data structure, a handle, a file size, or whatever is logical for the function to return. An error code may be available (but not always) by calling CSSM_GetError.

The information returned from CSSM_GetError includes both the error number and a Globally Unique ID (GUID) that associates the error with the module that set it. The GUID of each module can be obtained by calling CSSM_ListModules. CSSM_CompareGuids can then be called to determine from which module an error came.

Each module must have a mechanism for reporting their errors to the calling application. In general, there are two types of errors a module can return:

Errors defined by OCSF that are common to a particular type of service provider module Errors reserved for use by individual service provider modules

Since some errors are predefined by OCSF, those errors have a set of predefined numeric values that are reserved by OCSF and cannot be redefined by modules. For errors that are particular to a module, a different set of predefined values has been reserved for their use. Table 39 lists the range of error numbers defined for OCSF and service provider modules. Appendix A, “OCSF Errors,” on page 267 lists the errors defined by OCSF.

Table 39. OCSF Framework and Module Error Numbers

Error Number Range	OCSF Component
1000 – 1999	CSP errors defined by OCSF
2000 - 2999	CSP errors reserved for individual CSP modules
3000 – 3999	CL errors defined by OCSF
4000 – 4999	CL errors reserved for individual CL modules
5000 – 5999	DL errors defined by OCSF
6000 – 6999	DL errors reserved for individual DL modules
7000 – 7999	TP errors defined by OCSF
8000 – 8999	TP errors reserved for individual TP modules
9000 – 9499	KR errors defined by OCSF
9500 – 9999	KR errors reserved for individual KR modules
10000 – 19999	OCSF Framework errors

The calling application must determine how to handle the error returned by CSSM_GetError. Detailed descriptions of the error values will be available in the corresponding specification, the cssmerr.h header file, and the information for specific modules. If a routine does not know how to handle the error, it may choose to pass the error to its caller.

Error values returned by a function should not be overwritten, if at all possible. For example, if a CSP call returns an error indicating that it could not encrypt the data, the caller should not overwrite it with an error simply indicating that the CSP failed, as it destroys valuable error handling and debugging information. However, after processing an error, the application should reset the error to zero using CSSM_ClearError, in order to prevent the error from being handled again later.

Errors are kept on a thread basis, and each error API affects only the current thread's error information.

Data Structures

This describes the data structures for OCSF error handling.

CSSM_BOOL

This data type is used to indicate a true or false condition.

```
typedef uint32 CSSM_BOOL;

#define CSSM_TRUE 1
#define CSSM_FALSE 0
```

Definitions:

CSSM_TRUE
: Indicates a true result or a true value.

CSSM_FALSE
: Indicates a false result or a false value.

CSSM_ERROR

```
typedef struct cssm_error {
    uint32 error;
    CSSM_GUID guid;
} CSSM_ERROR, *CSSM_ERROR_PTR
```

CSSM_RETURN

This data type is used to indicate whether a function was successful.

```
typedef enum cssm_return {
            CSSM_OK = 0,
            CSSM_FAIL = -1
} CSSM_RETURN
```

Definitions:

CSSM_OK
: Indicates operation was successful.

CSSM_FAIL
: Indicates operation was not successful.

Error Handling Functions

This describes the interfaces for OCSF error handling.

CSSM_ClearError

Purpose

This function sets the current error value for the current thread to CSSM_OK. This can be called if the current error value has been handled and therefore is no longer a valid error.

Format

```
void CSSMAPI CSSM_ClearError (void)
```

Parameters

None

Related Information

CSSM_SetError
CSSM_GetError

CSSM_CompareGuids

Purpose

This function determines if two GUIDs are equal.

Format

```
CSSM_BOOL CSSMAPI CSSM_CompareGuids (CSSM_GUID GUID1,CSSM_GUID GUID2)
```

Parameters

Input

GUID1
A GUID.

GUID2
A GUID.

Return Value

CSSM_TRUE if the two GUIDs are equal, CSSM_FALSE if they are not equal.

Notes

GUIDs are returned in the error information of CSSM_GetError. Once you know which type of error is returned (i.e., CSP, CL, TP, DL), you can call CSSM_ListModules to get a list of all the modules that are registered and their GUIDs in order to determine which module set the error. This can be useful for debugging purposes if there is more than one type of module for each type installed on the system.

Related Information

CSSM_GetError
CSSM_ListModules

CSSM_GetError

Purpose

This function returns the error information for the current thread.

Format

```
CSSM_ERROR_PTR CSSMAPI CSSM_GetError (void)
```

Parameters

None

Return Value

Returns the current error information. If a NULL pointer is returned, the error information for the current thread has not been initialized. CSSM_GetError attempts to initialize the information if it does not exist, but if that fails, a NULL pointer is returned. If the pointer is not NULL and the error code is CSSM_OK, then there is no current error.

Related Information

CSSM_InitError
CSSM_DestroyError
CSSM_ClearError
CSSM_SetError
CSSM_IsCSSMError
CSSM_IsCLError
CSSM_IsTPError
CSSM_Is_DLError
CSSM_IsCSPError

CSSM_SetError

Purpose

This function sets the current error information for the current thread to *error* and *GUID*.

Format

```
CSSM_RETURN CSSMAPI CSSM_SetError (CSSM_GUID_PTR GUID, uint32 error)
```

Parameters

Input

GUID
Pointer to the GUID of the module.

error
An error number. It should fall within one of the valid CSSM, CL, TP, DL, KRSP, or CSP error ranges.

Return Value

CSSM_OK if error was successfully set. A return value of CSSM_FAIL indicates that the error number passed is not within a valid range, the GUID passed is invalid. No error information is available.

Related Information

CSSM_InitError
CSSM_DestroyError
CSSM_ClearError
CSSM_GetError

Chapter 18. Application Memory Functions

When OCSF or modules return memory structures to applications, that memory is maintained by the application. Instead of using a model where the application passes memory blocks to the modules to work on, the OCSF model requires the application to supply memory functions. This has the advantage that applications are not required to know the size of memory blocks they supply to OCSF and the add-ins. The memory that the application receives is in its process space, and this prevents the application from walking through the memory of the OCSF or the modules. An application that has access to secure memory could supply functions to the Cryptographic Service Provider (CSP) for managing that memory. All data returned from the CSP will be through that secure memory. When the application no longer requires the memory, it is responsible for freeing it.

Applications register their memory functions with the service provider modules during attach time (CSSM_ModuleAttach), and with OCSF during initialization (CSSM_Init).

CSSM_MEMORY_FUNCS and CSSM_API_MEMORY_FUNCS

This structure is used by applications to supply memory functions for OCSF and the service provider modules. The functions are used when memory needs to be allocated by OCSF or service provider modules for returning data structures to the applications.

```
typedef struct cssm_memory_funcs {
    void * (*malloc_func) (uint32 Size, void *AllocRef);
    void (*free_func) (void *MemPtr, void *AllocRef);
    void * (*realloc_func) (void *MemPtr, uint32 Size, void *AllocRef);
    void * (*calloc_func) (uint32 Num, uint32 Size, void *AllocRef);
    void *AllocRef;
} CSSM_ MEMORY_FUNCS, *CSSM_ MEMORY_FUNCS_PTR;

typedef CSSM_MEMORY_FUNCS CSSM_API_MEMORY_FUNCS;
typedef CSSM_API_MEMORY_FUNCS *CSSM_API_MEMORY_FUNCS_PTR;
```

Definitions:

Malloc_func
: Pointer to function that returns a void pointer to the allocated memory block of at least size bytes from heap *AllocRef*.

Free_func
: Pointer to function that deallocates a previously-allocated memory block (*memblock*) from heap *AllocRef*.

realloc_func
: Pointer to function that returns a void pointer to the reallocated memory block (*memblock*) of at least size bytes from heap *AllocRef*.

calloc_func
: Pointer to function that returns a void pointer to an array of num elements of length size initialized to zero from heap *AllocRef*.

AllocRef
: Pointer that can be used at the discretion of the application developer to implement additional memory management features such as usage counters.

Initialization of Memory Structure

The memory structure CSSM_MEMORY_FUNCS requires pointers to functions that implement the memory routines. The example is an application supplying the C run-time utilities malloc, realloc and free to the memory structure. The memory structure is then used by the CSSM_Init call.

```
        /* Allocating the structure */
MemoryFuncs = (CSSM_MEMORY_FUNCS_PTR) malloc (sizeof (CSSM_MEMORY_FUNCS));

/* Initialize the memory function structure */
MemoryFuncs->malloc_func = HeapMalloc;
MemoryFuncs->realloc_func = HeapRealloc;
MemoryFuncs->free_func = HeapFree;
MemoryFuncs->calloc_func = HeapCalloc;
MemoryFuncs->AllocRef = HeapID;

        /* Initialize the CSSM */
CSSM_Init (Version, MemoryFuncs, NULL);
```

CSSM_Memory_FUNCS Example

These two examples are application-defined memory functions. The first example, app_malloc, allocates memory using the system malloc, call and increments a counter palloc_ref, each time the function is called. The memory pointer value returned by malloc is returned to the caller. The second example, app_free, frees memory and decrements the counter palloc_ref.

```
/*****************************************************************************/
void * app_malloc (uint32 size, void *palloc_ref)
{

    if (palloc_ref != NULL)
        *(uint32 *) palloc_ref += 1;
    else
        printf("\tpalloc_ref NULL value passed to allocation function\n");

        return(malloc(size));
}
/*****************************************************************************/
/*****************************************************************************/
void app_free(void * ptr, void *palloc_ref)
{
    if (palloc_ref != NULL)
        * (uint32 *) palloc_ref -= 1;
    else
        printf("\tpallocref NULL value passed to free function\n");
    free(ptr);
    return;
}
/*****************************************************************************/
```

Appendix A. OCSF Errors

Appendix A, "OCSF Errors" lists all the errors used by the Open Cryptographic Services Facility.

Cryptographic Service Provider Module Errors

This table provides Cryptographic Service Provider (CSP) module errors.

Table 40. General CSP Messages and Errors

Error Code	Error Name
1001	CSSM_CSP_UNKNOWN_ERROR
1002	CSSM_CSP_REGISTER_ERROR
1003	CSSM_CSP_VERSION_ERROR
1004	CSSM_CSP_CONVERSION_ERROR
1005	CSSM_CSP_NO_TOKENINFO
1006	CSSM_CSP_INTERNAL_ERROR
1007	CSSM_CSP_SERIAL_REQUIRED
1008	CSSM_CSP_NOT_IMPLEMENTED

Table 41. CSP Memory Errors

Error Code	Error Name
1010	CSSM_CSP_MEMORY_ERROR
1011	CSSM_CSP_NOT_ENOUGH_BUFFER
1012	CSSM_CSP_ERR_OUTBUF_LENGTH
1013	CSSM_CSP_NO_OUTBUF
1014	CSSM_CSP_ERR_INBUF_LENGTH
1015	CSSM_CSP_ERR_KEYBUF_LENGTH
1016	CSSM_CSP_NO_SLOT

Table 42. Invalid CSP Parameters

Error Code	Error Name
1020	CSSM_CSP_INVALID_CSP_HANDLE
1021	CSSM_CSP_INVALID_POINTER
1022	CSSM_CSP_INVALID_CERTIFICATE
1023	CSSM_CSP_INVALID_ALGORITHM
1024	CSSM_CSP_INVALID_WINDOW_HANDLE
1025	CSSM_CSP_INVALID_CALLBACK
1026	CSSM_CSP_INVALID_CONTEXT
1027	CSSM_CSP_INVALID_CONTEXT_HANDLE
1028	CSSM_CSP_INVALID_CONTEXT_POINTER
1029	CSSM_CSP_INVALID_DATA_POINTER

Table 42. Invalid CSP Parameters (continued)

Error Code	Error Name
1030	CSSM_CSP_INVALID_DATA_COUNT
1031	CSSM_CSP_INVALID_KEY_LENGTH
1032	CSSM_CSP_INVALID_KEY
1033	CSSM_CSP_INVALID_KEY_POINTER
1034	CSSM_CSP_INVALID_ALGORITHM_MODE
1035	CSSM_CSP_INVALID_PADDING
1036	CSSM_CSP_INVALID_KEY_ATTRIBUTE
1037	CSSM_CSP_INVALID_PARAM_LENGTH
1038	CSSM_CSP_INVALID_IV_SIZE
1039	CSSM_CSP_INVALID_SIGNATURE
1040	CSSM_CSP_INVALID_DEVICE_ID
1041	CSSM_CSP_INVALID_KEYCLASS
1042	CSSM_CSP_INVALID_MODULE_HANDLE
1043	CSSM_CSP_INVALID_KEY_TYPE
1044	CSSM_CSP_INVALID_ITERATION_COUNT

Table 43. File I/O Errors

Error Code	Error Name
1050	CSSM_CSP_FILE_NOT_EXISTS
1051	CSSM_CSP_FILE_NOT_OPEN
1052	CSSM_CSP_FILE_OPEN_FAILED
1053	CSSM_CSP_FILE_CREATE_FAILED
1054	CSSM_CSP_FILE_READ_FAILED
1055	CSSM_CSP_FILE_WRITE_FAILED
1056	CSSM_CSP_FILE_CLOSE_FAILED
1057	CSSM_CSP_FILE_COPY_FAILED
1058	CSSM_CSP_FILE_DELETE_FAILED
1059	CSSM_CSP_FILE_FORMAT_ERROR

Table 44. CSP Cryptographic Errors

Error Code	Error Name
1065	CSSM_CSP_PUBKEY_GET_ERROR
1066	CSSM_CSP_QUERY_SIZE_FAILED
1067	CSSM_CSP_UNKNOWN_ALGORITHM
1068	CSSM_CSP_OPERATION_UNSUPPORTED
1069	CSSM_CSP_VECTOROFBUFS_UNSUPPORTED
1070	CSSM_CSP_STAGED_OPERATION_UNSUPPORTED
1071	CSSM_CSP_KEY_MODULUS_UNSUPPORTED
1072	CSSM_CSP_KEY_LENGTH_UNSUPPORTED
1073	CSSM_CSP_PADDING_UNSUPPORTED

Table 44. CSP Cryptographic Errors (continued)

Error Code	Error Name
1074	CSSM_CSP_IV_SIZE_UNSUPPORTED
1075	CSSM_CSP_GET_APIMEMFUNC_ERROR
1076	CSSM_CSP_INPUT_LENGTH_OVERSIZE
1077	CSSM_CSP_INPUT_LENGTH_ERROR
1078	CSSM_CSP_INPUT_DATA_ERROR
1079	CSSM_CSP_UNSUPPORTED_STORAGE_MASK
1080	CSSM_CSP_OPERATION_IN_PROGRESS
1081	CSSM_CSP_NO_WRITE_PERMISSIONS
1082	CSSM_CSP_EXCLUSIVE_UNAVAILABLE
1083	CSSM_CSP_UPDATE_WITHOUT_INIT
1084	CSSM_CSP_LOGIN_FAILED
1085	CSSM_CSP_ALREADY_LOGGED_IN
1086	CSSM_CSP_NOT_LOGGED_IN
1087	CSSM_CSP_KEY_PROTECTED
1088	CSSM_CSP_CALLBACK_FAILED
1089	CSSM_CSP_ROUNDS_UNSUPPORTED
1090	CSSM_CSP_EFFECTIVE_BITS_UNSUPPORTED
1091	CSSM_CSP_INCOMPATIBLE_VERSION
1092	CSSM_CSP_INCOMPATIBLE_KEY_VERSION
1093	CSSM_CSP_ALGORITHM_UNSUPPORTED
1094	CSSM_CSP_OPERATION_FAILED

Table 45. Missing or Invalid CSP Parameters

Error Code	Error Name
1100	CSSM_CSP_PARAM_NO_PARAM
1101	CSSM_CSP_PARAM_NO_PASSWORD
1102	CSSM_CSP_PARAM_NO_SEED
1103	CSSM_CSP_PARAM_NO_KEY
1104	CSSM_CSP_PARAM_NO_SALT
1105	CSSM_CSP_PARAM_NO_MODULUS
1106	CSSM_CSP_PARAM_NO_OUTPUT_SIZE
1108	CSSM_CSP_PARAM_NO_KEY_LENGTH
1109	CSSM_CSP_PARAM_NO_MODE
1110	CSSM_CSP_PARAM_NO_DATA
1111	CSSM_CSP_PARAM_NO_INIT_VECTOR
1112	CSSM_CSP_PARAM_NO_PADDING
1113	CSSM_CSP_PARAM_NO_ROUNDS
1114	CSSM_CSP_PARAM_NO_RANDOM
1115	CSSM_CSP_PARAM_NO_REMAINDATA
1116	CSSM_CSP_PARAM_NO_ALG_PARAMS

Table 45. Missing or Invalid CSP Parameters (continued)

Error Code	Error Name
1117	CSSM_CSP_PARAM_INVALID_VALUE
1118	CSSM_CSP_PARAM_NO_EFFECTIVE_BITS
1119	CSSM_CSP_PARAM_NO_PRIME
1120	CSSM_CSP_PARAM_NO_BASE
1121	CSSM_CSP_PARAM_NO_SUBPRIME
1122	CSSM_CSP_PARAM_NO_ALG_ID
1123	CSSM_CSP_PARAM_NO_KEY_TYPE
1124	CSSM_CSP_PARAM_NO_ITERATION_COUNT

Table 46. Password Errors

Error Code	Error Name
1130	CSSM_CSP_PASSWORD_INCORRECT
1131	CSSM_CSP_PASSWORD_SAME
1132	CSSM_CSP_PASSWORD_LENGTH_ERROR
1133	CSSM_CSP_PASSWORD_INVALID

Table 47. Key Management Messages and Errors

Error Code	Error Name
1140	CSSM_CSP_PRIKEY_LOAD_ERROR
1141	CSSM_CSP_PRIKEY_NOT_FOUND
1142	CSSM_CSP_PRIKEY_ALREADY_EXIST
1143	CSSM_CSP_PRIKEY_GET_ERROR
1144	CSSM_CSP_PRIKEY_PUBKEY_INCONSISTENT
1150	CSSM_CSP_KEY_DUPLICATE
1151	CSSM_CSP_KEY_BAD_KEY
1152	CSSM_CSP_KEY_BAD_LENGTH
1153	CSSM_CSP_KEY_NO_PARAM
1154	CSSM_CSP_KEY_ALGID_NOTMATCH
1155	CSSM_CSP_KEY_BLOBTYPE_INCORRECT
1156	CSSM_CSP_KEY_CLASS_INCORRECT
1157	CSSM_CSP_KEY_DELETE_FAILED
1158	CSSM_CSP_KEY_USAGE_INCORRECT
1159	CSSM_CSP_KEY_NOT_PROTECTED
1160	CSSM_CSP_KEY_FORMAT_INCORRECT

Table 48. Random Generation (RNG) Messages and Errors

Error Code	Error Name
1200	CSSM_CSP_RNG_FAILED
1201	CSSM_CSP_RNG_UNKNOWN_ALGORITHM
1202	CSSM_CSP_RNG_NO_METHOD

Table 49. Key Generation Messages and Errors

Error Code	Error Name
1210	CSSM_CSP_KEYGEN_FAILED
1211	CSSM_CSP_KEYGEN_UNKNOWN_ALGORITHM
1212	CSSM_CSP_KEYGEN_NO_METHOD

Table 50. Unique ID Generation Messages and Errors

Error Code	Error Name
1220	CSSM_CSP_UIDG_FAILED
1221	CSSM_CSP_UIDG_UNKNOWN_ALGORITHM
1222	CSSM_CSP_UIDG_NO_METHOD

Table 51. Encryption/Decryption Messages

Error Code	Error Name
1230	CSSM_CSP_ENC_UNKNOWN_ALGORITHM
1231	CSSM_CSP_ENC_NO_METHOD
1232	CSSM_CSP_ENC_FAILED
1233	CSSM_CSP_ENC_INIT_FAILED
1234	CSSM_CSP_ENC_UPDATE_FAILED
1235	CSSM_CSP_ENC_FINAL_FAILED
1236	CSSM_CSP_ENC_BAD_IV_LENGTH
1237	CSSM_CSP_ENC_IV_ERROR
1238	CSSM_CSP_ENC_BAD_KEY_LENGTH
1239	CSSM_CSP_ENC_UNKNOWN_MODE
1250	CSSM_CSP_DEC_UNKNOWN_ALGORITHM
1251	CSSM_CSP_DEC_NO_METHOD
1253	CSSM_CSP_DEC_FAILED
1254	CSSM_CSP_DEC_INIT_FAILED
1255	CSSM_CSP_DEC_UPDATE_FAILED
1256	CSSM_CSP_DEC_FINAL_FAILED
1257	CSSM_CSP_DEC_BAD_IV_LENGTH
1258	CSSM_CSP_DEC_IV_ERROR
1259	CSSM_CSP_DEC_BAD_KEY_LENGTH
1260	CSSM_CSP_DEC_UNKNOWN_MODE

Table 52. Sign/Verify Messages and Errors

Error Code	Error Name
1350	CSSM_CSP_SIGN_UNKNOWN_ALGORITHM
1351	CSSM_CSP_SIGN_NO_METHOD
1352	CSSM_CSP_SIGN_FAILED
1353	CSSM_CSP_SIGN_INIT_FAILED
1354	CSSM_CSP_SIGN_UPDATE_FAILED

Table 52. Sign/Verify Messages and Errors (continued)

Error Code	Error Name
1355	CSSM_CSP_SIGN_FINAL_FAILED
1360	CSSM_CSP_VERIFY_FAILED
1361	CSSM_CSP_VERIFY_INIT_FAILED
1362	CSSM_CSP_VERIFY_UPDATE_FAILED
1363	CSSM_CSP_VERIFY_FINAL_FAILED
1365	CSSM_CSP_VERIFY_UNKNOWN_ALGORITHM
1366	CSSM_CSP_VERIFY_NO_METHOD

Table 53. Digest Function Errors

Error Code	Error Name
1380	CSSM_CSP_DIGEST_UNKNOWN_ALGORITHM
1382	CSSM_CSP_DIGEST_NO_METHOD
1383	CSSM_CSP_DIGEST_FAILED
1384	CSSM_CSP_DIGEST_INIT_FAILED
1385	CSSM_CSP_DIGEST_UPDATE_FAILED
1386	CSSM_CSP_DIGEST_CLONE_FAILED
1387	CSSM_CSP_DIGEST_FINAL_FAILED

Table 54. Message Authentication Code (MAC) Function Errors

Error Code	Error Name
1390	CSSM_CSP_MAC_UNKNOWN_ALGORITHM
1392	CSSM_CSP_MAC_NO_METHOD
1393	CSSM_CSP_MAC_FAILED
1394	CSSM_CSP_MAC_INIT_FAILED
1395	CSSM_CSP_MAC_UPDATE_FAILED
1396	CSSM_CSP_MAC_CLONE_FAILED
1397	CSSM_CSP_MAC_FINAL_FAILED

Table 55. Key Exchange Errors

Error Code	Error Name
1410	CSSM_CSP_KEYEXCH_GENPARAM_FAILED
1411	CSSM_CSP_KEYEXCH_PHASE1_FAILED
1412	CSSM_CSP_KEYEXCH_PHASE2_FAILED
1413	CSSM_CSP_KEYEXCH_UNKNOWN_ALGORITHM
1414	CSSM_CSP_KEYEXCH_NO_METHOD

Table 56. PassThrough Custom Errors

Error Code	Error Name
1420	CSSM_CSP_INVALID_PASSTHROUGH_ID
1421	CSSM_CSP_INVALID_PASSTHROUGH_PARAMS

Table 57. Wrap/Unwrap Errors

Error Code	Error Name
1450	CSSM_CSP_WRAP_UNKNOWN_ALGORITHM
1451	CSSM_CSP_WRAP_NO_METHOD
1452	CSSM_CSP_WRAP_FAILED
1456	CSSM_CSP_UNWRAP_UNKNOWN_ALGORITHM
1457	CSSM_CSP_UNWRAP_NO_METHOD
1458	CSSM_CSP_UNWRAP_FAILED

Table 58. Hardware CSP Errors

Error Code	Error Name
1470	CSSM_CSP_DEVICE_ERROR
1471	CSSM_CSP_DEVICE_MEMORY_ERROR
1472	CSSM_CSP_DEVICE_REMOVED
1473	CSSM_CSP_DEVICE_NOT_PRESENT
1474	CSSM_CSP_DEVICE_UNKNOWN
1490	CSSM_CSP_PERMISSIONS_READ_ONLY
1491	CSSM_CSP_PERMISSIONS_WRITE_PROTECT
1492	CSSM_CSP_PERMISSIONS_NOT_EXCLUSIVE

Table 59. Query Size Errors

Error Code	Error Name
1500	CSSM_CSP_QUERY_SIZE_UNKNOWN
1501	CSSM_CSP_QUERY_KEYSIZEINBITS_UNKNOWN

Mapping OCSF Error Codes to ICSF Error Codes

This table is a translation mapping between the OCSF error codes and the ICSF error codes. If you do not find the OCSF error code in this table refer to the *z/OS Cryptographic Services ICSF Application Programmer's Guide*, SA22-7522.

Table 60. Mapping the OCSF Error Codes to ICSF Error Codes

CDSA	Description	Hexidecimal	Decimal
2001	RS_0_OK	0X00000000L	/* 00 / 0 */
2002	RS_0_PARITY	0X00000004L	/* 00 / 4 */
2003	RS_0_CKDS_NULL_RECORD	0X00000008L	/* 00 / 8 */
2004	RS_0_INOUT_KEYID_IGNORE	0X0000000CL	/* 00 / 12 */
2005	RS_0_KEYID_REENCIPH	0X00002710L	/* 00 / 10000 */
2006	RS_4_KEYID_REENCIPH	0X00042710L	/* 04 / 10000 */
2007	RS_4_CHARCONV_ODD_LENGTH	0X000407D0L	/* 04 / 2000 */
2008	RS_4_PIN_DIDNT_VERIFY	0X00040BD4L	/* 04 / 3028 */
2009	RS_4_RFOMK_AND_PIN_DIDNT_VERIFY	0X00040BD8L	/* 04 / 3032 */
2010	RS_4_CVV_DIDNT_VERIFY	0X00040FA0L	/* 04 / 4000 */
2011	RS_4_MAC_DIDNT_VERIFY	0X00041F40L	/* 04 / 8000 */
2012	RS_4_RFOMK_AND_MAC_DIDNT_VERIFY	0X00041F44L	/* 04 / 8004 */
2013	RS_4_KEYTEST_DIDNT_VERIFY	0X00042328L	/* 04 / 9000 */
2014	RS_4_RFOMK_AND_KEYTEST_DIDNT_VER	0X0004232CL	/* 04 / 9004 */
2015	RS_4_LATCH_CONTENTION	0X00042330L	/* 04 / 9008 */
2016	RS_4_DIG_SIG_DIDNT_VERIFY	0X00042AF8L	/* 04 / 11000 */
2017	RS_4_GIVEN_AP_MISMATCH	0X000436B4L	/* 04 / 14004 */
2018	RS_4_AUTH_CODE_MISMATCH	0X000436B8L	/* 04 / 14008 */
2019	RS_8_IV_LENGTH	0X000807D4L	/* 08 / 2004 */
2020	RS_8_OVERLAP	0X000807D8L	/* 08 / 2008 */
2021	RS_8_IV_RA_COUNT	0X000807DCL	/* 08 / 2012 */
2022	RS_8_IV_RA_CONTENT	0X000807E0L	/* 08 / 2016 */
2023	RS_8_IV_FORM_CONTENT	0X000807E2L	/* 08 / 2018 */
2024	RS_8_FIELD_NOT_ZERO	0X000807E8L	/* 08 / 2024 */
2025	RS_8_IV_PAD_COUNT	0X000807ECL	/* 08 / 2028 */
2026	RS_8_IV_WHAT_KEY	0X000807F0L	/* 08 / 2032 */
2027	RS_8_IV_CV	0X000807F4L	/* 08 / 2036 */
2028	RS_8_IV_KEY_ID	0X000807F8L	/* 08 / 2040 */
2029	RS_8_IV_FORM_ID	0X000807FCL	/* 08 / 2044 */
2030	RS_8_FORM_KEY_TYPE_MISMATCH	0X00080800L	/* 08 / 2048 */
2031	RS_8_IV_CLEAR_KEY	0X00080804L	/* 08 / 2052 */
2032	RS_8_IV_KEY_FORM	0X00080808L	/* 08 / 2056 */
2033	RS_8_BAD_KEY_LENGTH	0X0008080CL	/* 08 / 2060 */
2034	RS_8_BAD_LENGTH_COMBO	0X00080810L	/* 08 / 2064 */
2035	RS_8_CALLER_NOT_AUTH	0X00080814L	/* 08 / 2068 */
2036	RS_8_KGN_IMEX_BAD_KEK_1	0X00080818L	/* 08 / 2072 */

Table 60. Mapping the OCSF Error Codes to ICSF Error Codes (continued)

CDSA	Description	Hexidecimal	Decimal
2037	RS_8_KIM_CHK_RIGHT_KEY	0X0008081CL	/* 08 / 2076 */
2038	RS_8_CANNOT_WIMP_KEY	0X00080824L	/* 08 / 2084 */
2039	RS_8_INVALID_ASCII_INPUT	0X00080828L	/* 08 / 2088 */
2040	RS_8_KEY_VALUES_NOT_ASCII	0X0008082CL	/* 08 / 2092 */
2041	RS_8_INVALID_ASCII_DECIMAL	0X00080830L	/* 08 / 2096 */
2042	RS_8_TSS_COMPAT_ERROR	0X00080834L	/* 08 / 2100 */
2043	RS_8_TEXT_NOTIN_CODETAB	0X00080838L	/* 08 / 2104 */
2044	RS_8_UNUSED_FIELD	0X0008083CL	/* 08 / 2108 */
2045	RS_8_WRONG_KEY_LENGTH_FOR_TYPE	0X00080840L	/* 08 / 2112 */
2046	RS_8_IV_PARAMETER_VALUE	0X00080844L	/* 08 / 2116 */
2047	RS_8_IV_PIN_RULE	0X00080BB8L	/* 08 / 3000 */
2048	RS_8_IV_PIN_LENGTH	0X00080BBCL	/* 08 / 3004 */
2049	RS_8_IV_PIN_CHECK_LENGTH	0X00080BC0L	/* 08 / 3008 */
2050	RS_8_IV_TSP	0X00080BC4L	/* 08 / 3012 */
2051	RS_8_IV_PIN_BLOCK_FORMAT	0X00080BC8L	/* 08 / 3016 */
2052	RS_8_IV_FORMAT_CONTROL	0X00080BD0L	/* 08 / 3024 */
2053	RS_8_IV_IGBP_OFFS_PIN_DIGIT	0X00080BD4L	/* 08 / 3028 */
2054	RS_8_IV_SEQUENCE_NUMBER	0X00080BDCL	/* 08 / 3036 */
2055	RS_8_NON_NUMERIC_DATA	0X00080BE0L	/* 08 / 3040 */
2056	RS_8_NOT_MULTI8	0X00080FA0L	/* 08 / 4000 */
2057	RS_8_TGT_CRYPT_NOT_AVAIL	0X00081388L	/* 08 / 5000 */
2058	RS_8_IV_CAMQ_MESSAGE_TYPE	0X0008138CL	/* 08 / 5004 */
2059	RS_8_IV_CAMQ_MESSAGE_LEN	0X00081390L	/* 08 / 5008 */
2060	RS_8_KEYID_DIDNT_CHECK	0X00082710L	/* 08 / 10000 */
2061	RS_8_RECREATE_KEY	0X00082714L	/* 08 / 10004 */
2062	RS_8_KEY_NOT_FOUND	0X0008271CL	/* 08 / 10012 */
2063	RS_8_IV_TYPE_KEY	0X00082720L	/* 08 / 10016 */
2064	RS_8_TOKEN_AND_BAD_CV	0X00082724L	/* 08 / 10020 */
2065	RS_8_IV_CV_LEFT	0X0008272CL	/* 08 / 10028 */
2066	RS_8_IV_CV_RIGHT	0X00082730L	/* 08 / 10032 */
2067	RS_8_IV_CVS	0X00082734L	/* 08 / 10036 */
2068	RS_8_IV_KI_VERSION	0X00082738L	/* 08 / 10040 */
2069	RS_8_KI_TYPE_AND_CV_MISMATCH	0X0008273CL	/* 08 / 10044 */
2070	RS_8_IV_KEY_TYPE	0X00082740L	/* 08 / 10048 */
2071	RS_8_NULL_KEY_ID_AND_TOKEN	0X00082744L	/* 08 / 10052 */
2072	RS_8_TWIST_AND_TOKEN	0X00082748L	/* 08 / 10056 */
2073	RS_8_LABEL_KEY_ID_AND_TOKEN	0X0008274CL	/* 08 / 10060 */
2074	RS_8_FLAG_MKVP_NOT_ON	0X00082754L	/* 08 / 10068 */
2075	RS_8_FLAG_ENC_KEY_NOT_ON	0X00082758L	/* 08 / 10072 */
2076	RS_8_FLAG_CV_NOT_ON	0X0008275CL	/* 08 / 10076 */

Table 60. Mapping the OCSF Error Codes to ICSF Error Codes (continued)

CDSA	Description	Hexidecimal	Decimal
2077	RS_8_PHIL_YEH_FLAG_ON	0X00082760L	/* 08 / 10080 */
2078	RS_8_CANT_REENCIPHER_EXPORTER	0X00082764L	/* 08 / 10084 */
2079	RS_8_IV_KEY_TYPE_FOR_SERVICE	0X00082768L	/* 08 / 10088 */
2080	RS_8_ANSI_PARITY_ENFORCED	0X0008276CL	/* 08 / 10092 */
2081	RS_8_ANSI_SINGLE_AKEK	0X00082770L	/* 08 / 10096 */
2082	RS_8_NOTARZ_NOT_ALLOWED	0X00082774L	/* 08 / 10100 */
2083	RS_8_INAKEK_PART_NOTARZD	0X00082778L	/* 08 / 10104 */
2084	RS_8_INVALID_KEYID_KPI	0X0008277CL	/* 08 / 10108 */
2085	RS_8_INVALID_CPLTNOT	0X00082780L	/* 08 / 10112 */
2086	RS_8_IV_KPI_RA_FOR_TOKEN	0X00082784L	/* 08 / 10116 */
2087	RS_8_IV_TOKEN_KEYTYPE_FOR_SERV	0X00082788L	/* 08 / 10120 */
2088	RS_8_NO_EXPORT_FOR_KID	0X0008278CL	/* 08 / 10124 */
2089	RS_8_RULE_ARRAY_KEYWORD_MISMATCH	0X00082790L	/* 08 / 10128 */
2090	RS_8_IV_FIELD_LENGTH	0X00082AF8L	/* 08 / 11000 */
2091	RS_8_PKA_IV_AUTH_VALUE	0X00082AFCL	/* 08 / 11004 */
2092	RS_8_PKA_IV_KEY_VALUES	0X00082B0CL	/* 08 / 11020 */
2096	RS_8_PKA_TOKEN_INCOMP	0X00082B10L	/* 08 / 11024 */
2097	RS_8_MSG_TOOLONG_FOR_SIG	0X00082B14L	/* 08 / 11028 */
2098	RS_8_PKA_KMMGMT_NOT_ALLOWED	0X00082B18L	/* 08 / 11032 */
2099	RS_8_INVALID_TEXT	0X00082B1CL	/* 08 / 11036 */
2100	RS_8_IV_RESULT_RSA_ENCDEC	0X00082B20L	/* 08 / 11040 */
2101	RS_8_IV_FIRST_SECTION_KEY_ID	0X00082B24L	/* 08 / 11044 */
2102	RS_8_IV_EYECATCHER	0X00082B28L	/* 08 / 11048 */
2103	RS_8_PKA_PRIVATE_REQ	0X00082B2CL	/* 08 / 11052 */
2104	RS_8_IV_PKA_INTERNAL_TOKLEN	0X00082B30L	/* 08 / 11056 */
2105	RS_8_IV_RSAOAEP_BT	0X00082B38L	/* 08 / 11064 */
2106	RS_8_IV_RSAOAEP_V	0X00082B3CL	/* 08 / 11068 */
2107	RS_8_IV_RSAOAEP_I	0X00082B40L	/* 08 / 11072 */
2108	RS_8_IV_MODULUS_LENGTH	0X00082B48L	/* 08 / 11080 */
2109	RS_8_PKA_PUBLIC_REQ	0X00082B4CL	/* 08 / 11084 */
2110	RS_8_PKA_SIGNONLY_REQ	0X00082B50L	/* 08 / 11088 */
2111	RS_8_FAILED_RACF_SERVICE	0X00083E80L	/* 08 / 16000 */
2112	RS_8_FAILED_RACF	0X00083E84L	/* 08 / 16004 */
2113	RS_8_NOT_SUPVR_STATE	0X00083E8CL	/* 08 / 16012 */
2114	RS_8_INOUT_KEYID_INVALID	0X00083E90L	/* 08 / 16016 */
2115	RS_8_SYSTEM_KEY_FUNC_NOTALLOW	0X00083E94L	/* 08 / 16020 */
2116	RS_8_INVALID_KEY_TOKEN	0X00083E98L	/* 08 / 16024 */
2117	RS_8_SYNTAX_ERROR_IN_KEY_LABEL	0X00083EA0L	/* 08 / 16032 */
2118	RS_8_DUPLICATE_KEY_LABEL	0X00083EA4L	/* 08 / 16036 */
2119	RS_8_LABEL_CHECK_FAILED	0X00083EA8L	/* 08 / 16040 */

Table 60. Mapping the OCSF Error Codes to ICSF Error Codes (continued)

CDSA	Description	Hexidecimal	Decimal
2120	RS_12_NOT_ACTIVE	0X000C0000L	/* 12 / 0 */
2121	RS_12_DYN_SERV_NOTAVAIL	0X000C0004L	/* 12 / 4 */
2122	RS_12_SERV_NOTAVAIL	0X000C0008L	/* 12 / 8 */
2123	RS_12_FAILED_EXIT	0X000C000CL	/* 12 / 12 */
2124	RS_12_INST_SERVICE_NOT_FOUND	0X000C0010L	/* 12 / 16 */
2125	RS_12_INTERNAL_SERVICE_CC3	0X000C0014L	/* 12 / 20 */
2126	RS_12_INTERNAL_ANSI_PARMERR	0X000C0018L	/* 12 / 24 */
2127	RS_12_CAMQ_ERROR	0X000C001CL	/* 12 / 28 */
2128	RS_12_CAMQ_INCOMPLETE_RESPONSE	0X000C0020L	/* 12 / 32 */
2129	RS_12_CAMQ_RETRY	0X000C0024L	/* 12 / 36 */
2130	RS_12_KEY_FAILED_MAC	0X000C2724L	/* 12 / 10020 */
2131	RS_12_INST_EXIT_REJECT	0X000C2728L	/* 12 / 10024 */
2132	RS_12_NOT_ACTIVE_SKI	0X000C272CL	/* 12 / 10028 */
2133	RS_12_LABEL_NOT_UNIQUE	0X000C2734L	/* 12 / 10036 */
2134	RS_12_CKDS_DYNALLOC_FAILED	0X000C1790L	/* 12 / 6032 */
2135	RS_12_CKDS_UNALLOC_FAILED	0X000C1794L	/* 12 / 6036 */
2136	RS_12_CKDS_OPEN_FAILED	0X000C273CL	/* 12 / 10044 */
2137	RS_12_CKDS_IOERROR	0X000C2740L	/* 12 / 10048 */
2138	RS_12_NO_SPACE_CKT	0X000C2744L	/* 12 / 10052 */
2139	RS_12_ESTAE_FAILED_IN_DYNIO	0X000C178CL	/* 12 / 6028 */
2140	RS_12_NO_IO_SUBTASK	0X000C274CL	/* 12 / 10060 */
2141	RS_12_NOT_ACTIVE_INIT	0X000C2EDCL	/* 12 / 11996 */
2142	RS_12_NOT_ACTIVE_PIN	0X000C2EE0L	/* 12 / 12000 */
2143	RS_12_LATCH_ERROR	0X000C2EE4L	/* 12 / 12004 */
2144	RS_12_INVALID_CKDS	0X000C8CB4L	/* 12 / 36020 */
2145	RS_12_PKA_FUNCTION_UNAVAIL_CCC	0X000C2B00L	/* 12 / 11008 */
2146	RS_12_PKA_FUNCTION_UNAVAIL_ECM	0X000C2B04L	/* 12 / 11012 */
2147	RS_12_PKA_MK_INVALID	0X000C2B08L	/* 12 / 11016 */
2148	RS_12_KEY_SIZE_INVALID	0X000C2B0CL	/* 12 / 11020 */
2149	RS_12_PKA_SERV_NOTAVAIL	0X000C2B10L	/* 12 / 11024 */
2150	RS_12_ESYS_KEYS_NOT_FOUND	0X000C2B14L	/* 12 / 11028 */
2151	RS_12_CAMQ_NOT_VALID_FOR_PKA	0X000C2B18L	/* 12 / 11032 */
2152	RS_12_PKDS_NOT_AVAILABLE	0X000C2B1CL	/* 12 / 11036 */
2153	RS_12_PKDS_CONTROL_RECORD_HASH_E	0X000C2B20L	/* 12 / 11040 */
2154	RS_12_SERIALIZATION_ON_PKDS_FAIL	0X000C2B24L	/* 12 / 11044 */
2155	RS_16_BIG_ERROR	0X00100004L	/* 16 / 4 */
2156	RS_12_CCP_ERROR	0X000C2B28L	/* 12 / 11048 */
2157	RS_8_INVALID_KEY_BYTE	0X000800B5L	/* 08 / 00181 */

IBM Software CSP and IBM Weak Software CSP Errors

This table shows the return codes from the OCSF Software Service providers.

Table 61. OCSF Software Service Provider Errors

Error Code	Error Name
2515	IBMSWCSP_ALGORITHM_NOT_SET
2516	IBMSWCSP_ALGORITHM_OBJ
2517	IBMSWCSP_ALG_OPERATION_UNKNOWN
2518	IBMSWCSP_ALLOC
2519	IBMSWCSP_CANCEL
2520	IBMSWCSP_DATA
2521	IBMSWCSP_EXPONENT_EVEN
2522	IBMSWCSP_EXPONENT_LEN
2523	IBMSWCSP_HARDWARE
2524	IBMSWCSP_INPUT_DATA
2525	IBMSWCSP_INPUT_LEN
2526	IBMSWCSP_KEY_ALREADY_SET
2527	IBMSWCSP_KEY_INFO
2528	IBMSWCSP_KEY_LEN
2529	IBMSWCSP_KEY_NOT_SET
2530	IBMSWCSP_KEY_OBJ
2531	IBMSWCSP_KEY_OPERATION_UNKNOWN
2532	IBMSWCSP_MEMORY_OBJ
2533	IBMSWCSP_MODULUS_LEN
2534	IBMSWCSP_NOT_INITIALIZED
2535	IBMSWCSP_NOT_SUPPORTED
2536	IBMSWCSP_OUTPUT_LEN
2537	IBMSWCSP_OVER_32K
2538	IBMSWCSP_RANDOM_NOT_INITIALIZED
2539	IBMSWCSP_RANDOM_OBJ
2540	IBMSWCSP_SIGNATURE
2541	IBMSWCSP_WRONG_ALGORITHM_INFO
2542	IBMSWCSP_WRONG_KEY_INFO
2543	IBMSWCSP_INPUT_COUNT
2544	IBMSWCSP_OUTPUT_COUNT
2545	IBMSWCSP_METHOD_NOT_IN_CHOOSER
2546	IBMSWCSP_KEY_WEAK

Certificate Library Module Errors

This table provides the Certificate Library (CL) module errors.

Table 62. Certificate Library

Error Code	Error Name
3001	CSSM_CL_UNKNOWN_FORMAT
3002	CSSM_CL_UNKNOWN_TAG
3003	CSSM_CL_INVALID_CONTEXT
3004	CSSM_CL_INVALID_CL_HANDLE
3005	CSSM_CL_INVALID_CC_HANDLE
3006	CSSM_CL_INVALID_CERT_POINTER
3007	CSSM_CL_INVALID_FIELD_POINTER
3008	CSSM_CL_INVALID_TEMPLATE
3009	CSSM_CL_INVALID_DATA_POINTER
3010	CSSM_CL_INVALID_SCOPE
3012	CSSM_CL_CERT_CREATE_FAIL
3013	CSSM_CL_CERT_VIEW_FAIL
3014	CSSM_CL_CERT_GET_FIELD_VALUE_FAIL
3015	CSSM_CL_CERT_GET_KEY_INFO_FAIL
3016	CSSM_CL_CERT_IMPORT_FAIL
3017	CSSM_CL_CERT_EXPORT_FAIL
3018	CSSM_CL_PASS_THROUGH_FAIL
3019	CSSM_CL_CERT_DESCRIBE_FORMAT_FAIL
3020	CSSM_CL_UNSUPPORTED_OPERATION
3021	CSSM_CL_MEMORY_ERROR
3022	CSSM_CL_CERT_SIGN_FAIL
3023	CSSM_CL_CERT_UNSIGN_FAIL
3024	CSSM_CL_CERT_VERIFY_FAIL
3025	CSSM_CL_RESULTS_HANDLE
3026	CSSM_CL_INVALID_SIGNER_CERTIFICATE
3027	CSSM_CL_NO_FIELD_VALUES
3028	CSSM_CL_INVALID_CRL_PTR
3029	CSSM_CL_CERT_ABORT_QUERY_FAIL
3030	CSSM_CL_CRL_CREATE_FAIL
3031	CSSM_CL_CRL_SET_FAIL
3032	CSSM_CL_CRL_ADD_CERT_FAIL
3033	CSSM_CL_CRL_REMOVE_CERT_FAIL
3034	CSSM_CL_CRL_SIGN_FAIL
3035	CSSM_CL_CRL_VERIFY_FAIL
3036	CSSM_CL_IS_CERT_IN_CRL_FAIL
3037	CSSM_CL_CRL_GET_FIELD_VALUE_FAIL
3038	CSSM_CL_CRL_ABORT_QUERY_FAIL

Table 62. Certificate Library (continued)

Error Code	Error Name
3039	CSSM_CL_CRL_DESCRIBE_FORMAT_FAIL
3040	CSSM_CL_INVALID_POINTER
3041	CSSM_CL_INVALID_DATA
3042	CSSM_CL_INITIALIZE_FAIL
3100	CSSM_CL_SIG_NOT_IN_CERT
3101	CSSM_CL_INVALID_REVOKER_CERT_PTR
3102	CSSM_CL_NO_REVOKED_CERTS_IN_CRL
3103	CSSM_CL_CERT_NOT_FOUND_IN_CRL
3104	CSSM_CL_CRL_SIGNSCOPE_NOT_SUPPORTED
3105	CSSM_CL_CRL_VERIFYSCOPE_NOT_SUPPORTED
3106	CSSM_CL_CRL_NOT_SIGNEDBY_SIGNER
3107	CSSM_CL_CRL_NO_FIELD_OID
3108	CSSM_CL_INVALID_REVOKED_CERT_PTR
3109	CSSM_CL_INVALID_INPUT_PTR
3110	CSSM_CL_KEY_ALGID_NOT_SUPPORTED
3111	CSSM_CL_GET_KEY_ATTRIBUTE_FAIL
3112	CSSM_CL_CERT_ENCODE_FAIL
3113	CSSM_CL_CERT_DECODE_FAIL
3114	CSSM_CL_SIGNATURE_ALGID_NOT_SUPPORTED
3115	CSSM_CL_KEY_FORMAT_UNKNOWN
3116	CSSM_CL_INVALID_CERT_ISSUER_NAME
3117	CSSM_CL_INVALID_CERT_SUBJECT_NAME
3118	CSSM_CL_MISSING_CERT_SUBJECT_NAME
3119	CSSM_CL_MISSING_CERT_ISSUER_NAME
3120	CSSM_CL_MISSING_CERT_VALIDITY
3121	CSSM_CL_MISSING_SUBJECT_PUB_KEY
3122	CSSM_CL_FIELD_NOT_PRESENT
3123	CSSM_CL_SIGNER_CERT_EXPIRED
3124	CSSM_CL_SUBJECT_CERT_EXPIRED
3125	CSSM_CL_INCOMPATIBLE_CSP
3126	CSSM_CL_GET_CSP_HANDLE_ATTRIBUTE_FAIL
3127	CSSM_CL_GET_GUID_FROM_HANDLE_FAIL
3128	CSSM_CL_FAILED_ TO_GET_TIME

Data Storage Library Module Errors

This table provides the Data Storage Library (DL) module errors.

Table 63. Data Storage Errors

Error Code	Error Name
5001	CSSM_DL_NOT_LOADED
5002	CSSM_DL_INVALID_DL_HANDLE
5003	CSSM_DL_DATASTORE_NOT_EXISTS
5004	CSSM_DL_MEMORY_ERROR
5005	CSSM_DL_DB_OPEN_FAIL
5006	CSSM_DL_INVALID_DB_HANDLE
5007	CSSM_DL_DB_CLOSE_FAIL
5008	CSSM_DL_DB_CREATE_FAIL
5009	CSSM_DL_DB_DELETE_FAIL
5010	CSSM_DL_INVALID_PTR
5011	CSSM_DL_DB_IMPORT_FAIL
5012	CSSM_DL_DB_EXPORT_FAIL
5013	CSSM_DL_INVALID_CERTIFICATE_PTR
5014	CSSM_DL_CERT_INSERT_FAIL
5015	CSSM_DL_CERTIFICATE_NOT_IN_DB
5016	CSSM_DL_CERT_DELETE_FAIL
5017	CSSM_DL_CERT_REVOKE_FAIL
5018	CSSM_DL_INVALID_SELECTION_PTR
5019	CSSM_DL_NO_CERTIFICATE_FOUND
5020	CSSM_DL_CERT_GETFIRST_FAIL
5021	CSSM_DL_NO_MORE_CERTS
5022	CSSM_DL_CERT_GET_NEXT_FAIL
5023	CSSM_DL_CERT_ABORT_QUERY_FAIL
5024	CSSM_DL_INVALID_CRL_PTR
5025	CSSM_DL_CRL_INSERT_FAIL
5026	CSSM_DL_CRL_NOT_IN_DB
5027	CSSM_DL_CRL_DELETE_FAIL
5028	CSSM_DL_NO_CRL_FOUND
5029	CSSM_DL_CRL_GET_FIRST_FAIL
5030	CSSM_DL_NO_MORE_CRLS
5031	CSSM_DL_CRL_GET_NEXT_FAIL
5032	CSSM_DL_CRL_ABORT_QUERY_FAIL
5033	CSSM_DL_GET_DB_NAMES_FAIL
5034	CSSM_DL_INVALID_PASSTHROUGH_ID
5035	CSSM_DL_PASS_THROUGH_FAIL
5036	CSSM_DL_INVALID_POINTER
5037	CSSM_DL_NO_DATASOURCES

Table 63. Data Storage Errors (continued)

Error Code	Error Name
5038	CSSM_DL_INCOMPATIBLE_VERSION
5039	CSSM_DL_INVALID_FIELD_INFO
5040	CSSM_DL_INVALID_ATTRIBUTE_NAME_FORMAT
5041	CSSM_DL_CONJUNCTIVE_NOT_SUPPORTED
5042	CSSM_DL_OPERATOR_NOT_SUPPORTED
5043	CSSM_DL_NO_MORE_OBJECT
5044	CSSM_DL_INVALID_RESULTS_HANDLE
5045	CSSM_DL_INVALID_ATTRIBUTE_NAME
5046	CSSM_DL_INVALID_ATTRIBUTE
5047	CSSM_DL_UNKNOWN_KEY_TYPE
5048	CSSM_DL_BUFFER_TOO_SMALL
5100	CSSM_DL_INVALID_DATA_POINTER
5101	CSSM_DL_INVALID_DLINFO_POINTER
5102	CSSM_DL_INSTALL_FAIL
5103	CSSM_DL_INVALID_GUID
5104	CSSM_DL_UNINSTALL_FAIL
5105	CSSM_DL_LIST_MODULES_FAIL
5107	CSSM_DL_ATTACH_FAIL
5108	CSSM_DL_DETACH_FAIL
5109	CSSM_DL_GET_INFO_FAIL
5110	CSSM_DL_FREE_INFO_FAIL
5111	CSSM_DL_INVALID_DLINFO_PTR
5112	CSSM_DL_INVALID_CL_HANDLE
5113	CSSM_DL_INVALID_CERTIFICATE_PTR
5114	CSSM_DL_INVALID_CRL
5115	CSSM_DL_INVALID_CRL_POINTER
5116	CSSM_DL_INVALID_RECORD_TYPE
5117	CSSM_DL_DATA_INSERT_FAIL
5118	CSSM_DL_DATA_GETFIRST_FAIL
5119	CSSM_DL_DATA_GETNEXT_FAIL
5120	CSSM_DL_NO_DATA_FOUND
5121	CSSM_DL_INVALID_AUTHENTICATION
5122	CSSM_DL_DATA_ABORT_QUERY_FAIL
5123	CSSM_DL_DATA_DELETE_FAIL

LDAP Data Library Module Errors

This table provides the LDAP Data Library module errors.

Table 64. LDAP Data Library Errors

Error Code	Error Name
6001	LDAPDL_OPERATIONS_ERROR
6002	LDAPDP_PROTOCOL_ERROR
6003	LDAPDL_TIMELIMIT_EXCEEDED
6004	LDAPDL_SIZELIMIT_EXCEEDED
6007	LDAPDL_STRONG_AUTH_NOT_SUPPORTED
6008	LDAPDL_STONG_AUTH_REQUIRED
6009	LDAPDL_PARTIAL_RESULTS
6010	LDAPDL_REFERRAL_NOT_FOLLOWED
6011	LDAPDL_ADMIN_LIMIT_EXCEEDED
6012	LDAPDL_UNAVAILABLE_CRITICAL_EXTENSION
6013	LDAPDL_CONFIDENTIALITY_REQUIRED
6014	LDAPDL_SASLBIND_IN_PROGRESS
6020	LDAPDL_NO_SUCH_ATTRIBUTE
6021	LDAPDL_UNDEFINED_TYPE
6022	LDAPDL_INAPPROPIRATE_MATCHING
6023	LDAPDL_CONSTRAINT_VIOLATION
6024	LDAPDL_TYPE_OR_VALUE_EXISTS
6025	LDAPDL_INVALID_SYNTAX
6032	LDAPDL_NO_SUCH_OBJECT
6033	LDAPDL_ALIAS_PROBLEM
6034	LDAPDL_INVALID_DN_SYNTAX
6035	LDAPDL_IS_LEAF
6036	LDAPDL_ALIAS_DEREF_PROBLEM
6048	LDAPDL_INAPPROPRIATE_AUTH
6049	LDAPDL_INVALID_CREDENTIALS
6050	LDAPDL_INSUFFICIENT_ACCESS
6051	LDAPDL_BUSY
6052	LDAPDL_UNAVAILABLE
6053	LDAPDL_UNWILLING_TO_PERFORM
6054	LDAPDL_LOOP_DETECT
6064	LDAPDL_NAMING_VIOLATION
6065	LDAPDL_OBJECT_CLASS_VIOLATION
6066	LDAPDL_NOT_ALLOWED_ON_NONLEAF
6067	LDAPDL_NOT_ALLOWED_ON_RDN
6068	LDAPDL_ALREADY_EXISTS
6069	LDAPDL_NO_OBJECT_CLASS_MODS
6070	LDAPDL_RESULTS_TOO_LARGE

Table 64. LDAP Data Library Errors (continued)

Error Code	Error Name
6071	LDAPDL_AFFECTS_MULTIPLE_DSAS
6080	LDAPDL_OTHER
6081	LDAPDL_SERVER_DOWN
6082	LDAPDL_LOCAL_ERROR
6083	LDAPDL_ENCODING_ERROR
6084	LDAPDL_DECODING_ERROR
6085	LDAPDL_TIMEOUT
6086	LDAPDL_AUTH_UNKNOWN
6087	LDAPDL_FILTER_ERROR
6088	LDAPDL_USER_CANCELLED
6089	LDAPDL_PARAM_ERROR
6090	LDAPDL_NO_MEMORY
6091	LDAPDL_CONNECT_ERROR
6092	LDAPDL_NOT_SUPPORTED
6093	LDAPDL_CONTROL_NOT_FOUND
6094	LDAPDL_NO_RESULTS_RETURNED
6095	LDAPDL_MORE_RESULTS_TO_RETURN
6096	LDAPDL_URL_ERR_NOTLDAPDL
6097	LDAPDL_URL_ERR_NODN
6098	LDAPDL_URL_ERR_BADSCOPE
6099	LDAPDL_URL_ERR_MEM
6100	LDAPDL_CLIENT_LOOP
6101	LDAPDL_REFERRAL_LIMIT_EXCEEDED
6112	LDAPDL_SSL_ALREADY_INITIALIZED
6113	LDAPDL_SSL_INITIALIZE_FAILED
6114	LDAPDL_SSL_CLIENT_INIT_NOT_CALLED
6115	LDAPDL_SSL_PARAM_ERROR
6116	LDAPDL_SSL_HANDSHAKE_FAILED
6117	LDAPDL_SSL_GET_CIPHER_FAILED
6128	LDAPDL_NO_EXPLICIT_OWNER

Trust Policy Module Errors

This table provides the Trust Policy (TP) module errors.

Table 65. Trust Policy Errors

Error Code	Error Name
7001	CSSM_TP_NOT_LOADED
7002	CSSM_TP_INVALID_TP_HANDLE
7003	CSSM_TP_INVALID_CL_HANDLE
7004	CSSM_TP_INVALID_DL_HANDLE
7005	CSSM_TP_INVALID_DB_HANDLE
7006	CSSM_TP_INVALID_CC_HANDLE
7007	CSSM_TP_INVALID_CERTIFICATE
7008	CSSM_TP_NOT_SIGNER
7009	CSSM_TP_NOT_TRUSTED
7010	CSSM_TP_CERT_VERIFY_FAIL
7011	CSSM_TP_CERTIFICATE_CANT_OPERATE
7012	CSSM_TP_MEMORY_ERROR
7013	CSSM_TP_CERT_SIGN_FAIL
7014	CSSM_TP_INVALID_CRL
7015	CSSM_TP_CERT_REVOKE_FAIL
7016	CSSM_TP_CRL_VERIFY_FAIL
7017	CSSM_TP_CRL_SIGN_FAIL
7018	CSSM_TP_APPLY_CRL_TO_DB_FAIL
7019	CSSM_TP_INVALID_GUID
7020	CSSM_TP_UNISTALL_FAIL
7021	CSSM_TP_INCOMPATIBLE_VERSION
7022	CSSM_TP_INVALID_ACTION
7023	CSSM_TP_VERIFY_ACTION_FAIL
7024	CSSM_TP_INVALID_DATA_POINTER
7025	CSSM_TP_INVALID_ID
7026	CSSM_TP_PASS_THROUGH_FAIL
7027	CSSM_TP_INVALID_CSP_HANDLE
7028	CSSM_TP_ANCHOR_NOT_SELF_SIGNED
7029	CSSM_TP_ANCHOR_NOT_FOUND

Key Recovery Module Errors

This table provides the Key Recovery (KR) module errors.

Table 66. Key Recovery Errors

Error Code	Error Name
9001	CSSM_KRSP_AUTHINFO_BUFFER_TOO_SMALL
9002	CSSM_KRSP_COULD_NOT_GET_HOSTINGO
9003	CSSM_KRSP_COULD_NOT_GET_USERID
9004	CSSM_KRSP_CRYPTO_CONTEXT_KEY_NOT_FOUND
9005	CSSM_KRSP_MEMORY_ERROR
9006	CSSM_KRSP_INTEGRITY_CHECK_FAILED
9007	CSSM_KRSP_INTEGRITY_TYPE_NOT_SUPPORTED
9008	CSSM_KRSP_INVALID_AUTHINFO_BUFFER
9009	CSSM_KRSP_INVALID_CRYPTO_CONTEXT
9010	CSSM_KRSP_INVALID_CRYPTO_CONTEXT_KEY
9011	CSSM_KRSP_INVALID_JURIS_PROFILE
9012	CSSM_KRSP_INVALID_KRCONTEXT
9013	CSSM_KRSP_INVALID_KRSP_CONFIG
9014	CSSM_KRSP_INVALID_KRTYPE
9015	CSSM_KRSP_INVALID_LOCAL_KRPROFILE
9016	CSSM_KRSP_KRPROFILE_ATTRIBUTE_NOT_FOUND
9017	CSSM_KRSP_LEMAN_GEN_REQUIRED
9018	CSSM_KRSP_LEUSE_GEN_REQUIRED
9019	CSSM_KRSP_ENT_GEN_REQUIRED

OCSF Framework Errors

These tables provide the OSCF framework errors.

Table 67. Memory Allocation Errors

Error Code	Error Name
10001	CSSM_MALLOC_FAILED
10002	CSSM_CALLOC_FAILED
10003	CSSM_REALLOC_FAILED

Table 68. File I/O Errors

Error Code	Error Name
10010	CSSM_FWRITE_FAILED
10011	CSSM_FREAD_FAILED
10012	CSSM_CANT_FSEEK
10013	CSSM_INVALID_FILE_PTR
10014	CSSM_END_OF_FILE

Table 69. Miscellaneous Errors

Error Code	Error Name
10020	CSSM_CANT_GET_USER_NAME
10021	CSSM_GETCWD_FAILED
10022	CSSM_ENV_VAR_NOT_FOUND
10023	CSSM_BAD_HASH_CONTEXT_INDEX
10024	CSSM_SET_ERROR_FAILED
10025	CSSM_RNG_INIT_FAILED
10026	CSSM_RNG_LOOP_LIMIT_EXCEEDED

Table 70. Dynamic Library Error

Error Code	Error Name
10030	CSSM_FREE_LIBRARY_FAILED
10031	CSSM_LOAD_LIBRARY_FAILED
10032	CSSM_CANT_GET_PROC_ADDR
10033	CSSM_CANT_GET_MODULE_HANDLE
10034	CSSM_CANT_GET_MODULE_FILE_NAME
10035	CSSM_INVALID_LIB_HANDLE
10036	CSSM_BAD_MODULE_HANDLE

Table 71. Registry Errors

Error Code	Error Name
10040	CSSM_CANT_CREATE_KEY
10041	CSSM_CANT_SET_VALUE
10042	CSSM_CANT_GET_VALUE
10043	CSSM_CANT_DELETE_SECTION

Table 71. Registry Errors (continued)

Error Code	Error Name
10044	CSSM_CANT_DELETE_KEY
10045	CSSM_CANT_ENUM_KEY
10046	CSSM_CANT_OPEN_KEY
10047	CSSM_CANT_QUERY_KEY
10048	CSSM_CANT_CREATE_REGISTRY
10049	CSSM_CANT_OPEN_REGISTRY

Table 72. Mutex/Synchronization Errors

Error Code	Error Name
10050	CSSM_CANT_CREATE_MUTEX
10051	CSSM_LOCK_MUTEX_FAILED
10052	CSSM_TRYLOCK_MUTEX_FAILED
10053	CSSM_UNLOCK_MUTEX_FAILED
10054	CSSM_CANT_CLOSE_MUTEX
10055	CSSM_INVALID_MUTEX_PTR

Table 73. Shared Memory File Errors

Error Code	Error Name
10060	CSSM_CANT_CREATE_SHARED_MEMORY_FILE
10061	CSSM_CANT_OPEN_SHARED_MEMORY_FILE
10062	CSSM_CANT_MAP_SHARED_MEMORY_FILE
10063	CSSM_CANT_UNMAP_SHARED_MEMORY_FILE
10064	CSSM_CANT_FLUSH_SHARED_MEMORY_FILE
10065	CSSM_CANT_CLOSE_SHARED_MEMORY_FILE
10066	CSSM_INVALID_PERMS
10067	CSSM_BAD_FILE_HANDLE
10068	CSSM_BAD_FILE_ADDR

Table 74. Key Formats

Error Code	Error Name
10080	CSSM_KEY_FORMAT_NOT-SUPPORTED

Table 75. General Errors

Error Code	Error Name
10100	CSSM_BAD_PTR_PASSED

Table 76. OCSF API Errors

Error Code	Error Name
10301	CSSM_INVALID_POINTER
10302	CSSM_EXPIRED
10303	CSSM_MEMORY_ERROR

Table 76. OCSF API Errors (continued)

Error Code	Error Name
10304	CSSM_INVALID_ATTRIBUTE
10305	CSSM_NOT_INITIALIZE
10306	CSSM_INSTALL_FAIL
10307	CSSM_REGISTRY_ERROR
10308	CSSM_INVALID_CONTEXT_HANDLE
10309	CSSM_INVALID_CSP_HANDLE
10310	CSSM_INVALID_TP_HANDLE
10311	CSSM_INVALID_CL_HANDLE
10312	CSSM_INVALID_DL_HANDLE
10313	CSSM_INCOMPATIBLE_VERSION
10314	CSSM_ATTACH_FAIL
10315	CSSM_NO_ADDIN
10316	CSSM_FUNCTION_NOT_IMPLEMENTED
10317	CSSM_INVALID_CONTEXT_POINTER
10318	CSSM_INVALID_MANIFEST_ATTRIB_POINTER
10319	CSSM_MODE_UNSUPPORTED
10320	CSSM_KEY_LENGTH_UNSUPPORTED
10321	CSSM_IV_SIZE_UNSUPPORTED
10322	CSSM_PADDING_UNSUPPORTED
10323	CSSM_KEY_MODULUS_UNSUPPORTED
10324	CSSM_PARAM_NO_KEY
10325	CSSM_INVALID_KRSP_HANDLE
10326	CSSM_KR_FIELDS_NOT_GENERATED
10327	CSSM_ENT_KR_POLICY_MODULE_NOT_FOUND
10328	CSSM_ENT_KR_POLICY_FUNC_NOT_FOUND
10329	CSSM_LE_POLICY_MODULE_CORRUPT
10330	CSSM_ENT_POLICY_MODULE_CORRUPT
10331	CSSM_LE_KR_NOT_ALLOWED
10340	CSSM_INVALID_SERVICE_MASK
10341	CSSM_INVALID_SUBSERVICEID
10342	CSSM_INVALID_INFO_LEVEL
10343	CSSM_MULTIPLE_ENCRYPT_ATTEMPT
10344	CSSM_ADDIN_AUTHENTICATION_FAILED
10345	CSSM_EISL_PKCS7_INVALID
10346	CSSM_EISL_SIGROOT_INVALID
10347	CSSM_EISL_MANIFEST_SECTION_NOT_FOUND
10348	CSSM_EISL_MODULE_VERIFICATION_FAILED
10349	CSSM_EISL_MODULE_LOAD_FAILED
10350	CSSM_EISL_CERTIFICATE_EXPIRED

Table 77. OCSF Privilege Mechanism Errors

Error Code	Error Name
10360	CSSM_INVALID_CREDENTIALS
10361	CSSM_NOT_AUTHORIZED
10362	CSSM_STRONG_CRYPTO_NOT_ALLOWED
10363	CSSM_CANT_GET_THREAD_ID
10364	CSSM_THREAD_EXEMPTION_ERROR
10365	CSSM_CANT_CREATE_CLEANUP_THREAD
10366	CSSM_PRIV_NOT_INITIALIZED
10367	CSSM_INVALID_NAME
10368	CSSM_INVALID_ATTRIBUTE_COUNT
10500	CSSM_INVALID_ADDIN_HANDLE
10501	CSSM_INVALID_GUID
10502	CSSM_MEM_FUNCS_NOT_MATCHING
10503	CSSM_VALUE_TOO_LARGE
10504	CSSM_VALUE_TOO_SMALL
10505	CSSM_RACF_PROFILE_READ_FAILURE

Appendix B. Accessibility

Accessible publications for this product are offered through the z/OS® Information Center, which is available at www.ibm.com/systems/z/os/zos/bkserv/.

If you experience difficulty with the accessibility of any z/OS information, please send a detailed message to mhvrcfs@us.ibm.com or to the following mailing address:

IBM® Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Accessibility features

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size.

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users accessing the z/OS Information Center using a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually

exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, you know that your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol giving information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this indicates a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should refer to separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

- ? means an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.
- ! means a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP will be applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1!

(KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

- * means a syntax element that can be repeated 0 or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

  Note:

  1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
  2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
  3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

- + means a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times; that is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS™, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted

for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (http://www.ibm.com/software/support/systemsz/lifecycle/)
- For information about currently-supported IBM hardware, contact your IBM representative.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml (http://www.ibm.com/legal/copytrade.shtml).

Glossary

This glossary defines technical terms and abbreviations used in Open Cryptographic Services Facility information. If you do not find the term you are looking for, refer to the index of the appropriate OCSF manual or view IBM Glossary of Computing Terms, located at:

http://www.ibm.com/ibm/terminology

Asymmetric algorithms
Cryptographic algorithms, where one key is used to encrypt and a second key is used 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.

certificate
See 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 (CA). CAs issue, verify, and revoke certificates.

certificate chain
The hierarchical chain of all the other certificates used to sign the current certificate. This includes the 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 CA can sign certificates it issues or co-sign 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 information for libraries of executable code.

certificate validity date
A start date and a stop date for the validity of the certificate. If a certificate expires, the CA may issue a new certificate.

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, generating a random number, etc. OCSF accommodates Data Encryption Standard (DES), RC2, RC4, International Data Encryption Algorithm (IDEA), and other encryption algorithms.

cryptographic service provier
Cryptographic Service Providers (CSPs) are 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 science for keeping data secure. Cryptography provides the ability to store information or to communicate between parties in such a way that prevents other non-involved parties from understanding the stored information or accessing and understanding the communication. The encryption process takes understandable text and transforms it into an unintelligible piece of data (called ciphertext); the decryption process restores the understandable text from the unintelligible data. Both involve a mathematical formula or algorithm and a secret sequence of data called a key.

Cryptographic services provide confidentiality (keeping data secret), integrity (preventing data from being modified), authentication (proving the identity of a resource or a user), and non-repudiation (providing proof that a message or transaction was send and/or received). There are two types of cryptography:

In shared/secret key (symmetric) cryptography there is only one key that is a shared secret between the two communicating parties. The same key is used for encryption and decryption.

In public key (asymmetric) cryptography different keys are used for encryption and decryption. A party has two keys: a public key and a private key. The two keys are mathematically related, but it is virtually impossible to derive the private key from the public key. A message that is encrypted with someone's public key (obtained from some public directory) can only be decrypted with the associated private key. Alternately, the private key can be used to "sign" the information; the public key can be used as verification of the source of the information.

cryptoki
Short for cryptographic token interface. See Token.

data encryption standard
In computer security, the National Institute of Standards and Technology (NIST) Data Encryption Standard (DES), adopted by the U.S. Government as Federal Information Processing Standard (FIPS) Publication 46, which allows only hardware implementations of the data encryption algorithm.

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 unforgettable credential in cyberspace. The certificate is issued by a trusted authority, 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 information
- Verify that the contents 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 MD5 with RSA encryption, and DSS, the proposed Digital Signature Standard defined as part of the U.S. Government Capstone project.

hash algorithm
A cryptographic algorithm used to hash a variable-size input stream into a unique, fixed-sized output value. 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.

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.

message digest
The digital fingerprint of an input stream. A cryptographic hash function is applied to an input message arbitrary length and returns a fixed-size output, which is called the digest value.

Open Cryptographic Services Facility (OCSF) Framework
Open Cryptographic Services Facility (OCSF) Framework. The Open Cryptographic Services Facility (OCSF) framework defines four key service components:

- Cryptographic Module Manager
- Trust Policy Module Manager
- Certificate Library Module Manager
- Data Storage Library Module Manager

The OCSF binds together all the security services required by applications. In particular, it facilitates linking digital certificates to cryptographic actions and trust protocols.

owned certificate
A certificate whose associated secret or private key resides in a local Cryptographic Service Provider (CSP). Digital-signing algorithms require using owned certificates when signing data for purposes of authentication and non-repudiation. A system may use certificates it does not own for purposes other than signing.

private key
The cryptographic key is used to decipher messages in public-key cryptography. This key is kept secret by its owner.

public key
The cryptographic key is used to encrypt messages in public-key cryptography. The public key is available to multiple users (i.e., the public).

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. Each Certificate Authority (CA) has a self-signed root certificate. The root certificate's public key is the foundation of signature verification in its domain.

S/MIME
Secure/Multipurpose Internet Mail Extensions (S/MIME) is a protocol that adds digital signatures and encryption to Internet MIME messages. MIME is the official proposed standard format for extended Internet electronic mail. Internet e-mail messages consist of two parts, the header and the body. The header forms a collection of field/value pairs structured to provide information essential for the transmission of the message. The body is normally unstructured unless the e-mail is in MIME format. MIME defines how the body of an e-mail message is structured. The MIME format permits e-mail to include enhanced text, graphics, audio, and more in a standardized manner via MIME-compliant mail systems. However, MIME itself does not provide any security services.

The purpose of S/MIME is to define such services, following the syntax given in PKCS #7 for digital signatures and encryption. The MIME body carries a PKCS #7 message, which itself is the result of cryptographic processing on other MIME body parts.

secure electronic transaction
A mechanism for securely and automatically routing payment information among users, merchants, and their banks. Secure Electronic Transaction (SET) is a protocol for securing bankcard transactions on the Internet or other open networks using cryptographic services. SET is a specification designed to utilize technology for authenticating parties involved in payment card purchases on any type of on-line network, including the Internet. SET was developed by Visa and MasterCard, with participation from leading technology companies, including Microsoft, IBM, Netscape, SAIC, GTE, RSA, Terisa Systems, and VeriSign. By using sophisticated cryptographic techniques, SET will make cyberspace a safer place for conducting business and is expected to boost consumer confidence in electronic commerce. SET focuses on maintaining confidentiality of

information, ensuring message integrity, and authenticating the parties involved in a transaction.

security context
A control structure that retains state information shared between a CSP and the application agent requesting service from the CSP. Only one context can be active for an application at any given time, but the application is free to switch among contexts at will, or as required. A security context specifies CSP and application-specific values, such as required key length and desired hash functions.

security-relevant event
An event where a CSP-provided function is performed, a security module is loaded, or a breach of system security is detected.

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.

signature
See Digital signature.

signature chain
The hierarchical chain of signers, from the root certificate to the leaf certificate, in a certificate chain.

symmetric algorithm
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 Data Encryption Standard (DES) and International Data Encryption Algorithm (IDEA). The U.S. Government endorsed DES as a standard in 1977. It is 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, one of the best known public algorithms, 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 smart cards and Personal Computer Memory Card International Association (PCMCIA) cards.

verification
The process of comparing two message digests. One message digest is generated by the message sender and included in the message. The message recipient computes the digest again. If the message digests are exactly the same, it shows or proves there was no tampering of the message contents by a third party (between the sender and the receiver).

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. Based on the trust level, secret keys may be shared and used to encrypt and decrypt all messages exchanged between the two parties. Encrypted exchanges are private, trusted communications.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W

Product Number: 5650-ZOS

Printed in USA

SC14-7513-00