Common Security Services Manager

Java* Application Programming Interface (API)

Revision 1.0
October 1996

Subject to Change Without Notice

Specification Disclaimer and Limited Use License

This specification is for release version 1.0, October 1996.

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Some aspects of this Specification may be covered under various United States or foreign patents. No license, express or implied, by estoppel or otherwise, to any other intellectual property rights is granted herein.

Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information in this specification. Intel doesn't warrant or represent that such implementation(s) will not infringe such rights.

If you are interested in receiving an appropriate license to Intel's intellectual property rights relating to the interface defined in this specification, contact us for details at cdsa@ibeam.intel.com.

Copyright© 1996 Intel Corporation. All rights reserved.
Intel Corporation, 5200 N.E. Elam Young Parkway, Hillsboro, OR 97124-6497

*Other product and corporate names may be trademarks of other companies and are used only for explanation and to the owner's benefit, without intent to infringe.

Table of Contents

1. INTRODUCTION 1

1.1 Intended Audience 1
1.2 Document Organization 1
1.3 CDSA and CSSM Architectural Overview 2

2. OVERVIEW OF THE CSSM JAVA API 4

2.1 Design Philosophy – Goals and Non-goals 5
2.2 Services available through the CSSM Java API 6

3. ARCHITECTURE OF THE CSSM JAVA API 7

3.1 CSSM Java Security Classes 9
3.2 Exchanging Objects between Java and C language Environments 9

4. JAVA CLASS DEFINITIONS 10

4.1 Certificate 11
4.2 CLRegistry 18
4.3 CRL 22
4.4 Crypto 26
4.5 CSPRegistry 36
4.6 DataStore 40
4.7 DLRegistry 46
4.8 Key 50
4.9 SecurityContext 52
4.10 TPRegistry 61
4.11 TrustPolicy 64
4.12 CSSMException 71

List of Figures

Figure 1: The Common Data Security Architecture. 2
Figure 2: Architecture of the Common Security Services Manager (CSSM). 3
Figure 3: Independent Security Service Stacks based on application's language. 5
Figure 4: A single Security Service Stack for all applications, regardless of implementation language. 5
Figure 5: Three layer architecture of the CSSM security services on the Java platform. 8
Figure 6: CSSM-managed data structures are referenced by Java objects. 10

1. Introduction

This document presents the Common Security Services Manager (CSSM) Java* Application Programming Interface (API). This interface allows Java applications and Java applets to access security services provided by the CSSM. CSSM is the central component of the Common Data Security Architecture (CDSA), a layered, extensible, multi-platform security architecture. CSSM also defines a C language API to security services.

1.1 Intended Audience

This document is for application developers and Independent Software Vendors (ISVs) who are familiar with:

- CSSM
- CSSM C language interface
- Security technologies
- Cryptographic operations
- Security protocols

This document is not intended for developers of security add-in modules. CSSM does not support add-in security modules written in Java. Therefore this document does not include information required to develop add-in trust policy modules, certificate library modules, data storage library modules, or cryptographic service provider modules.

1.2 Document Organization

This document, *CSSM Java Application Programming Interface*, is divided into these sections:

Section 1	Introduction
Section 2	Overview of the CSSM Java API
Section 3	Architecture of the CSSM Java API
Section 4	Java class definitions

The Common Data Security Architecture (CDSA) and the Common Security Services Manager (CSSM) are described in the *Common Data Security Architecture Specification*. The C language API for CSSM is presented in the *CSSM Application Programming Interface Specification*. CSSM is an extensible architecture.

1.3 CDSA and CSSM Architectural Overview

The Common Data Security Architecture (CDSA) defines the infrastructure for a complete set of security services. Cryptography is the computational base used to build secure protocols and secure systems. CDSA is an extensible architecture that provides mechanisms to manage add-in security modules dynamically. Figure 1 shows the three basic layers of the Common Data Security Architecture. The Common Security Services Manager (CSSM) is the core of CDSA. CSSM defines a common API that applications must use to access services of add-in security modules. Applications request security services through the CSSM security API or via layered security services and tools implemented over the CSSM API. The requested security services are performed by these four types of security modules:

- Cryptographic Services
- Trust Policy Services
- Certificate Library Services
- Data Storage Library Services

Independent software and hardware vendors can provide add-in security modules as competitive products. Applications can direct their requests to modules from specific vendors or to any module that performs the required services. These add-in modules augment the set of available security services.

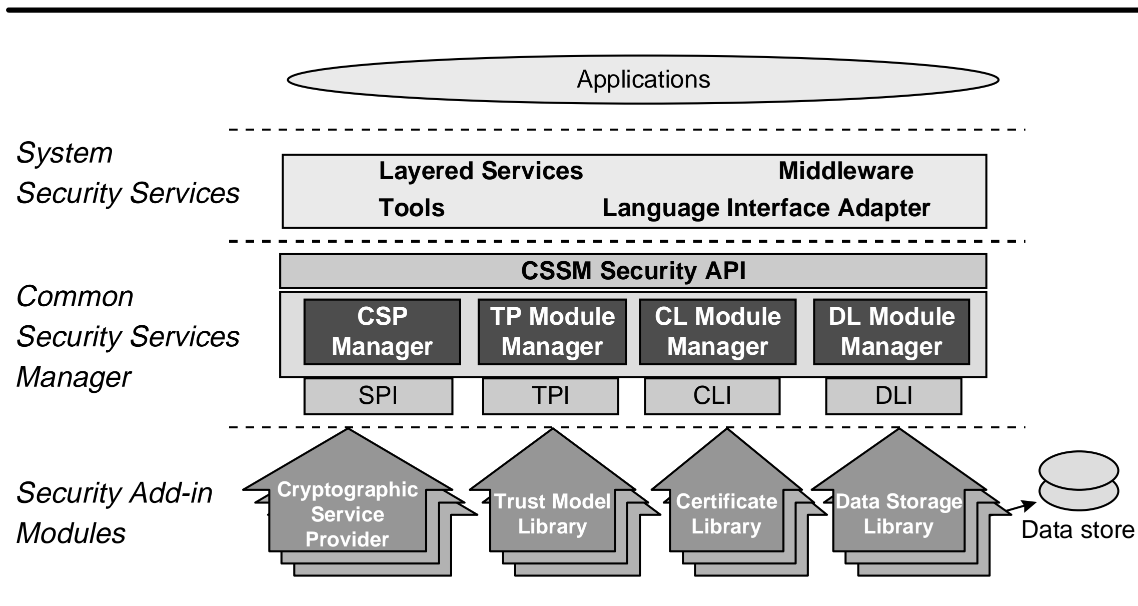

Figure 1: The Common Data Security Architecture.

The Common Security Services Manager bonds the various security functions required by applications to cryptographic service provider modules (or tokens) and certificate libraries. In particular, it facilitates linking digital certificates to cryptographic actions and trust protocols. Tokens and certificate libraries plug into the CSSM as add-in modules. Figure 2 shows the detailed design of the CSSM.

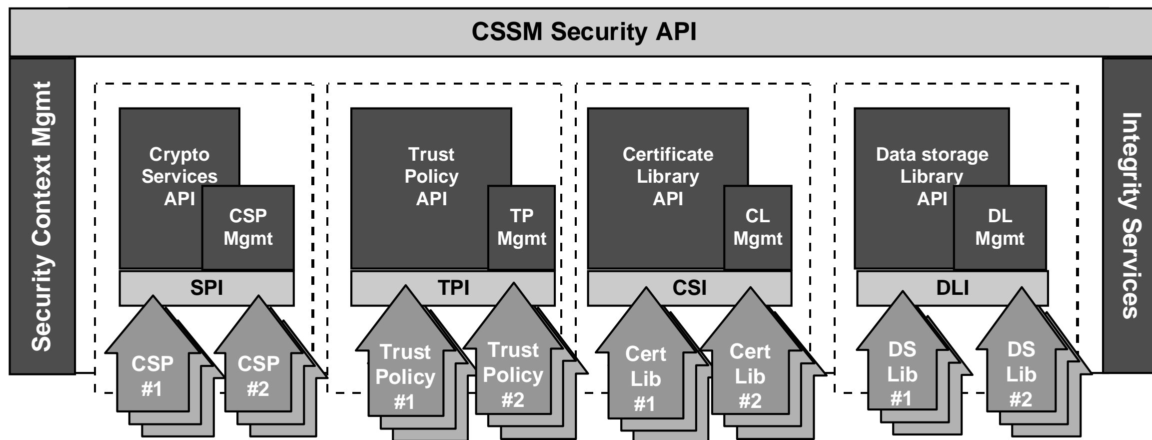

Figure 2: Architecture of the Common Security Services Manager (CSSM).

CSSM defines a rich operations API, which applications use to:

- Perform cryptographic operations.
- Determine the trustworthiness of a certificate holder to perform an action.
- Manipulate certificates.
- Access persistent storage.

This API defines four extensible service roles: Cryptography, trust policy enforcement, certificate manipulations, and persistent storage.

Extensible Services

Cryptographic Service Providers (CSPs) are add-in modules, which perform cryptographic operations including encryption, decryption, digital signaturing, key-pair generation, random-number generation, and key exchange.

Trust Policy (TP) modules implement policies defined by authorities and institutions, such as VeriSign* (as a certificate authority) and MasterCard* (as an institution). Each trust policy module embodies the semantics of a trust model, based on using digital certificates as credentials. Applications can use a digital certificate as an identity credential and/or an authorization credential.

Certificate Library (CL) modules provide format-specific, syntactic manipulation of memory-resident digital certificates and certificate revocation lists.

Data Storage Library (DL) modules provide persistent storage for certificates and certificate revocation lists. Certificate Libraries and Data Storage Libraries make the existence and manipulation of certificates and revocation lists orthogonal to the persistence of those objects. Add-in modules must implement some or all of the CSSM-defined security API in the delivery of their services.

The API calls defined for each add-in module are further categorized as service operations and module management operations. Module management functions support module installation, registration of module features and attributes, and queries to retrieve information on module availability and features.

Infrastructure Services

CSSM also provides integrity services and security-context management. CSSM applies the integrity check facility to itself to ensure that the currently executing instance of CSSM code has not been tampered with. The CSSM integrity manager also provides a built-in audit capability, which logs all of the occurrences of a selectable set of security operations.

Security-context management provides secured runtime caching of user-specific state information and secrets. The manager focuses on caching state information and parameters for performing cryptographic operations. Examples of secrets that must be cached during application execution include the application's private key, and the application's digital certificate.

Multiple add-in modules of each type may be concurrently active within the CSSM infrastructure. CSSM defines a dispatching mechanism that calls selected trust policy modules, certificate library modules, data storage library modules, and cryptographic service provider modules. The call being mapped by the dispatch mechanism can originate in an application, in another add-in security module, or in CSSM itself. Each add-in module has an associated handle. Callers select their target add-in module(s) and the CSSM dispatcher forwards the call to the selected module.

Modules must be loaded before they can receive function calls from the CSSM dispatcher. An error condition occurs if the invoked function is not implemented by the selected module.

In summary, the CSSM provides these services through its API calls:

- Certificate-based services and operations
- Comprehensive, extensible SPIs for cryptographic service-provider modules, trust policy modules, certificate library modules, and data storage modules
- Registration and management of available cryptographic service-provider modules, trust policy modules, certificate library modules, and data storage modules
- Caching of keys and secrets required as part of the run-time context of a user application
- Callback functions for disk, screen, and keyboard I/O supported by the operating system
- A test-and-check function to ensure CSSM integrity
- Management of concurrent security operations
- Auditable security functions

2. Overview of the CSSM Java API

2.1 Design Philosophy – Goals and Non-goals

The primary goal of the CSSM Java API is to provide Java applications and applets with an interface to access CSSM-managed security services on a large number of platforms.

With the CSSM Java API, you no longer need to host two security services – one to support Java applications and a second to support applications developed in other languages. Figure 3 shows the more cumbersome architecture of two independent security service stacks. Figure 4 shows a single implementation of security services accessible by applications developed in a range of languages.

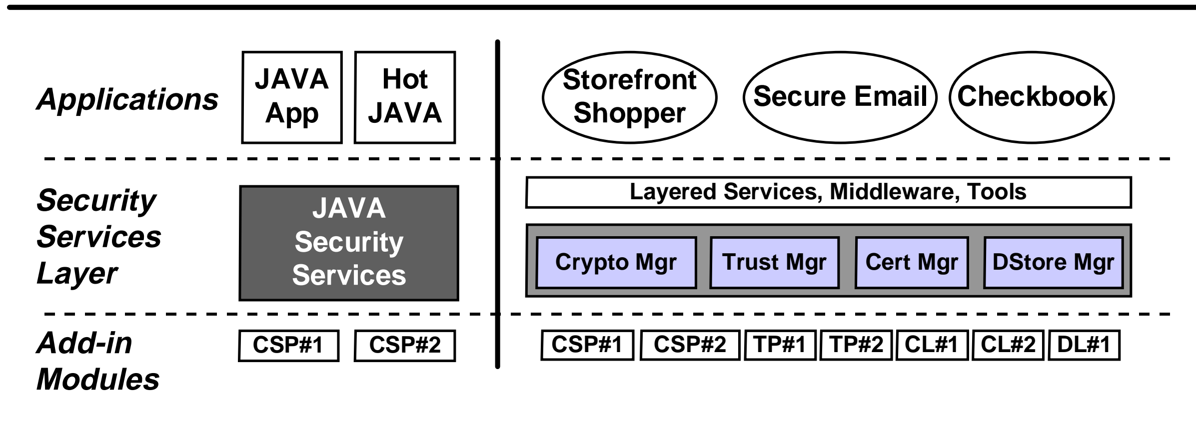

Figure 3: Independent Security Service Stacks based on application's language.

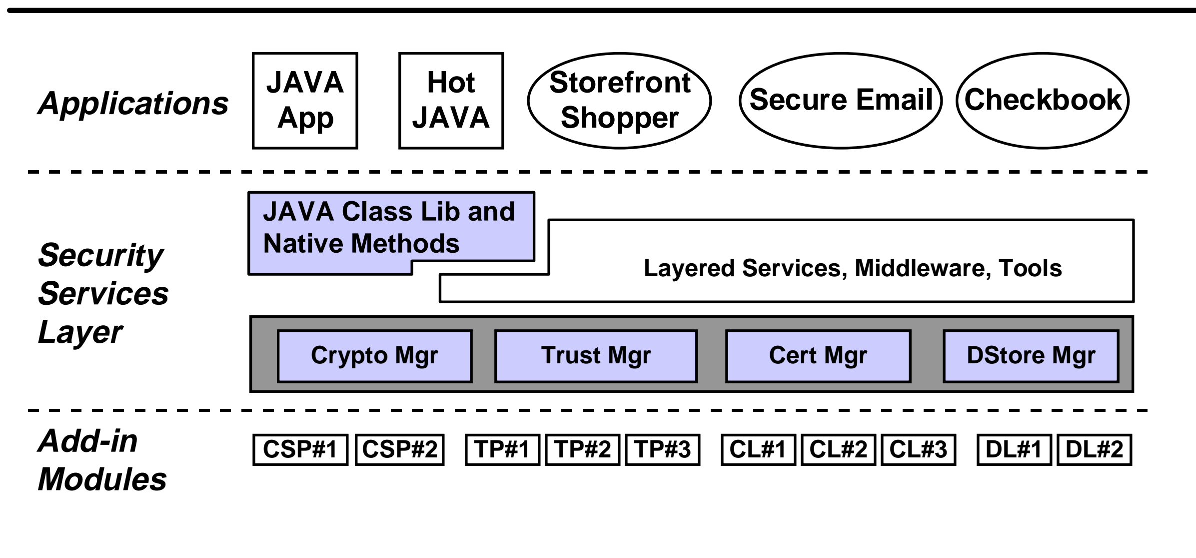

Figure 4: A single Security Service Stack for all applications, regardless of implementation language.

The first two language-specific APIs defined for CSSM were the C language API and the Java API. Both languages are widely used for the deployment of Internet applications.

The CSSM APIs were initially defined and specified in the C language. CSSM exports its abstract concepts and security services to other programming languages and language environments by defining a Language Adaptation Interface for each target language and environment. The Java classes defined in this document form the CSSM language adapter for the Java programming environment, thus achieving the single stack security architecture for Java. Implementation of the CSSM Java class methods invokes CSSM services as native methods, in the C language.

Two design approaches were considered for the CSSM Java classes:

- Begin with current CSSM concepts, objectify those concepts, and define Java classes and methods based on CSSM's C language-based concepts.
- Begin with a clean slate and apply general object modeling techniques to define Java classes and methods for security services.

The first design approach was applied in the design of the CSSM Java classes. The assumptions and motivations for this decision include:

- The most frequently used languages for implementing Internet applications are C and C++
- Java applications and applets are often developed in conjunction with other application modules implemented in C and C++
- A single security model, accessible from Java and the C language, simplifies software development (and training) of the many developers who are implementing applications in Java, C and C++
- Design and implementation of the Java methods to invoke CSSM security services must be achievable with reasonable effort, in a reasonable timeframe
- A pure object-oriented model of security services for Java applications is not required
- A usable, consistent object model of security services for Java applications is required

2.2 Services available through the CSSM Java API

The CSSM Security Services APIs work with four types of add-in security modules:

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

The API for each module type is divided into two categories:

- Security services, such as cryptographic operations, trust verification, certificate manipulation, and certificate storage and retrieval
- Management and maintenance of the CSSM security infrastructure, such as installing security add-in modules

The CSSM Java API does not provide access to all of the CSSM functions available to a C language or C++ language application. In general, the functionality provided by the security services portion of the

CSSM API calls are accessible through the Java API. The CSSM functions to manage and maintain the CSSM infrastructure itself are not accessible through the Java API. You must use the C language utilities and management tools to install and maintain the CSSM security environment. Java applications may access any of the security services that have been installed on a given platform.

Also CSSM does not support add-in security modules implemented in the Java language. This design decision precludes Java-based implementations of:

- Cryptographic Service Providers (CSPs)
- Trust Policy Modules (TPs)
- Certificate Library Modules (CLs)
- Data Storage Library Modules (DLs)
- System administration tools to install and configure the security infrastructure
- System administration tools to dynamically install add-in security modules such as CSPs, TPs, CLs, and DLs

This means that the Java applications and applets inherit the security infrastructure installed and configured on the local system by utilities and tools implemented in C and C++. Java applications can share in this environment, interoperating with each other and with C language applications, but they do not define the environment. Currently we do not view Java as a system administration language of choice in a multi-language environment. Administrative capabilities could be added by extending the CSSM Java API. Support for add-in security modules implemented in Java requires extensions in the installation, dynamic loading and registration facilities provided by CSSM. These capabilities are not supported at this time.

3. Architecture of the CSSM Java API

The architecture of Java access to CSSM services consists of the three layers shown in Figure 5. Java applications and Java applets use classes in the CSSM Java API to access security services. Each method in the Java security classes is backed by a native method which invokes one or more CSSM functions implemented in C.

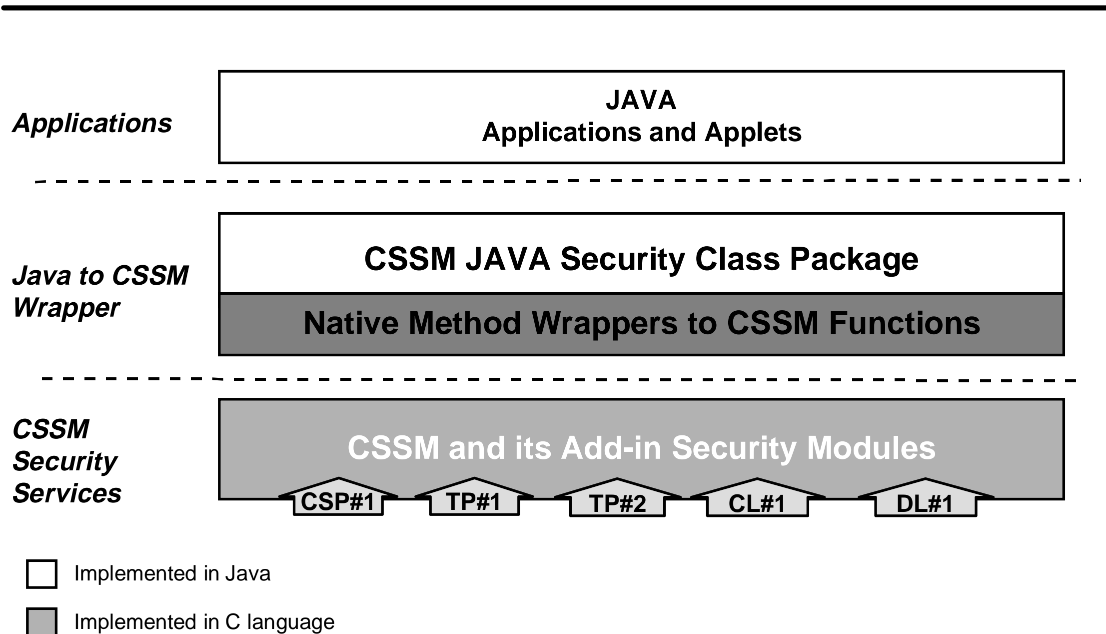

Figure 5: Three layer architecture of the CSSM security services on the Java platform.

3.1 CSSM Java Security Classes

The CSSM Java API defines twelve security classes:

CSSM Java Security Classes

Class Name	Brief Description
Certificate	Objects that bind identity to a public key; may also carry authorizations, and descriptive information
CRL	A set of revoked certificates
Key	Objects that store public or symmetric keys, used when performing cryptographic operations
DataStore	A persistent store of certificates and/or CRLs
Crypto	Provides all available cryptographic operations
SecurityContext	Stores parameters for cryptographic operations and other security services
TrustPolicy	Provides basic TP actions
TPRegistry	Supports queries for selecting locally available trust policy modules (TPs)
CLRegistry	Supports queries to select locally available certificate library modules (CLs)
CSPRegistry	Supports queries to select locally available cryptographic services (CSPs)
DLRegistry	Supports queries to select locally available data storage library modules (DLs)
CSSMException	Notification of an error when manipulating instances of above classes

The complete definition of these classes (with variables and methods) is presented in Section 4.

3.2 Exchanging Objects between Java and C language Environments

The CSSM C language API defines an application-to-service-provider interaction model. The CSSM Java API maintains this model, which includes:

- A certificate-based API
- State structures that are passed between the application and the service providers (CSPs, TPs, CLs, and DLs)
- Application handles that reference objects in a system-managed cache

The certificate focus of the CSSM is reflected in the CSSM Java class structure. The concept of certificate objects is easy to use in Java programming and in C language programming.

The exchange of state information via compound data structures is the standard approach used in C language programming. Visible, complex data structures is not the typical design approach used in Java programming, but it can be easily accommodated by the Java application developer. For a consistent model of CSSM security services, C language state structures are made visible in the Java classes.

CSSM represents its security objects as C language structures. Each memory-resident structure must be allocated and managed by exactly one layer in the three layer CSSM Java architecture. Other layers access the data contained in the structure by handles/references/pointers to the actual data. Figure 6 shows the physical implementation of an instance of the Java class named SecurityContext. The Java application invokes the constructor for the SecurityContext class, which is implemented as part of the CSSM Java wrapper. The wrapper invokes CSSM functions to create a multi-field security context structure that is cached by the CSSM. The CSSM returns the handle of the cached structure to the invoking wrapper. The wrapper completes the construction operation by creating a small Java object containing only the handle to the CSSM security context structure and returning this small object to the Java application. The multi-field data structure is not replicated in the Java Object. C language applications use the same handle-based interaction model to create a security context structure, but C language applications may directly manipulate handles.

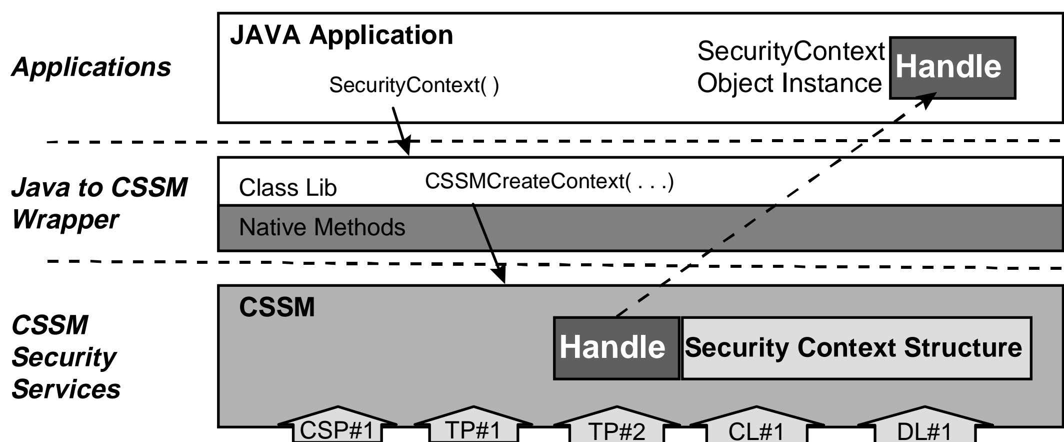

Figure 6: CSSM-managed data structures are referenced by Java objects.

A Java application manipulates the value of a SecurityContext object by applying methods to the Java object that contains the handle to the CSSM-managed security context structure. The *Get* method defined in the CSSM Java security classes and implemented by the CSSM Java wrapper make the handle transparent to the Java application (i.e. the Java wrapper manages the indirection from the handle to the structure it references). The *Get* method allows the Java application to retrieve individual fields within the CSSM-managed data structure. These CSSM structures contain low-level data types, such as bit arrays and byte arrays. Low-level data types were selected for efficient manipulation by C language programs. These data types are not the typical choice for a Java application developer, but the Java environment does provide classes for manipulating these data types. In order to limit the overhead of constant type conversion as data moves between the Java application and the CSSM (via the *Get* method), most of the data types remain faithful to the CSSM C language-type definitions and are delivered to the Java application with little or no conversion. This ensures that Java applications can interoperate with data created by C language applications, and developers working in both languages will have fewer concepts and types to convert or translate when moving among implementation languages.

4. Java Class Definitions

4.1 Certificate

This class provides generic functions for applications to create, query, update, sign, and verify a digital certificate.

Certificate operations are performed by add-in Trust Policy modules (TPs), Certificate Library modules (CLs) and Data Storage Library modules (DLs). TP is to answer the question *Is this certificate trusted for this action?* When a TP function has determined the trustworthiness of performing an action, the TP function may invoke CL functions or DL functions to carry out the mechanics of the approved action. CL modules implement syntactic manipulation of memory-resident certificates and certificate revocation lists. DL modules provide persistence of certificates and certificate revocation lists.

public final class Certificate
extends Object

Constructors:

public Certificate (CLRegistry cl, byte[ ][ ] cssmByteOIDs, byte[ ][ ] byteValues, byte[ ][ ] cssmNumberOIDs, int [ ] numberValues, byte[ ] cssmKeyOID, KeypublicKey) throws CSSMException

Creates a certificate based on the input data. The information of CL is saved in the object. Memory for the certificate is allocated and managed by the CSSM. The application must invoke the free method even though the certificate object is no longer needed; this allows CSSM to clean up the memory cache.

Parameters:

cl - The CL that will perform certificate creation.

cssmByteOIDs - Array of field identifiers that identify the byte array field values used to create the new certificate.

byteValues - Array of the byte array field values corresponding to the cssmByteOIDs.

cssmNumberOIDs - Array of field identifiers that identify the integer field values used to create the new certificate.

numberValues - Array of the integer field values corresponding to the cssmNumberOIDs.

cssmKeyOID - Field identifier that identify the key value used to create the new certificate.

publicKey - Key value corresponding to the cssmKeyOID.

Throws: CSSMException

- Invalid CL
- Invalid OID
- Specified CL doesn’t support this function
- Unable to create certificate

public Certificate (CLRegistry cl, InputStream is)
throws CSSMException, IOExcepiton

Instantiates a memory-resident certificate with certificate data from an InputStream. The certificate data must have been saved using the save method. Memory for the certificate is allocated and managed by the CSSM. The application must invoke the free method even though the certificate object is no longer needed; this allows CSSM to clean up the memory cache.

Parameters:

cl - The certificate library that will be used to perform the certificate operations.

is - The InputStream containing the field values corresponding to cssmOIDs.

Throws: CSSMException

- Invalid CL
- Invalid OID
- Specified CL doesn't support this function
- Unable to create certificate

public Certificate (CLRegistry cl)

Creates an empty certificate. This method creates an empty certificate in preparation for retrieving a certificate from a Data Store. The information of the CL is saved in the object. Memory for the certificate is allocated and managed by the CSSM. The application must invoke the free method even though the certificate object is no longer needed; this allows CSSM to clean up the memory cache.

Parameters:

cl - The certificate library that will be used to perform the certificate operations.

Variables:

private int certHandle - The handle to the CSSM-cached data structure that corresponds to this Certificate object.

private int CLHandle - The handle to the CSSM-cached data structure that corresponds to the specified certificate library object.

private int resultHandle - The handle to the CSSM-cached results returned in response to the getFirstFieldValue method. This handle is used to retrieve subsequent matches based on the same query.

private CLRegistry clRegistry - The certificate library that will be used to perform the certificate operations.

Methods:

public int getHandle ( )

Gets the handle of this certificate object.

Returns:

The certificate handle.

protected int getCLHandle ( )

Gets the handle of the CLRegistry used to construct this certificate object.

Returns:

The CLRegistry (certificate library) handle.

public void free ( )

Frees memory that CSSM allocated to cache a certificate.

public Certificate certSign (SecurityContext context, Certificate signerCert, byte[ ][ ] scopeByteOIDs, byte[ ][ ] scopeByteValues, byte[ ][ ] scopeNumberOIDs, int [ ] scopeNumberValues, byte[ ] scopeKeyOID, Key scopeKeyValue) throws CSSMException

Signs the fields of the certificate indicated in the signing Scope using the private key associated with the public key found in the signer's certificate.

Parameters:

context - The security context that describes the environment and parameters under which the cryptographic signing operations should be performed.

signerCert - Signer's certificate.

scopeByteOIDs - An array of field IDs that identify which byte array fields of the certificate should be signed. This parameter is optional.

scopeByteValues - An array of byte array field values corresponding to the scopeByteOIDs.

scopeNumberOIDs - An array of field IDs that identify which int fields of the certificate should be signed. This parameter is optional.

scopeNumberValues - An array of int field values corresponding to the scopeNumberOIDs.

scopeKeyOID - Field ID that identify key field of the certificate should be signed. This parameter is optional.

scopeKeyValue - Key value corresponding to the scopeKeyOID.

Returns:

A certificate containing the signer's signature.

Throws: CSSMException

- The security context is invalid
- No signing certificate was specified
- Invalid CL
- Unrecognized certificate format
- Revoked or expired signer certificate
- Invalid scope
- Not enough memory
- CL doesn't support this function
- Unable to sign certificate

public boolean certVerify (SecurityContext context, Certificate signerCert, byte[ ][ ] scopeByteOIDs, byte[ ][ ] scopeByteValues, byte[ ][ ] scopeNumberOIDs, int [ ] scopeNumberValues, byte[ ] scopeKeyOID, Key scopeKeyValue) throws CSSMException

Verifies the certificate using the public key contained in the signer's certificate. This checks for certificate tampering by verifying the digital signature on the verify Scope fields.

Parameters:

context - The security context that describes the environment and parameters under which the cryptographic verification operation should be performed.

signerCert - The signer's certificate.

scopeByteOIDs - An array of field IDs that identify which byte array fields of the certificate should be verified. This parameter is optional.

scopeByteValues - An array of byte array field values corresponding to the scopeByteOIDs.

scopeNumberOIDs - An array of field IDs that identify which int fields of the certificate should be verified. This parameter is optional.

scopeNumberValues - An array of int field values corresponding to the scopeNumberOIDs.

scopeKeyOID - Field ID that identify key field of the certificate should be verified. This parameter is optional.

scopeKeyValue - Key value corresponding to the scopeKeyOID.

Returns:

True - Verify successfully.

False - The signature associated with the certificate is not verified.

Throws: CSSMException

- The security context is invalid
- No signed certificate was specified
- Cannot retrieve the public key from the signer's certificate
- Unrecognized certificate format
- Invalid scope
- CL doesn't support this function

public Certificate unsign (SecurityContext context, Certificate signerCert, byte[ ][ ] scopeByteOIDs, byte[ ][ ] scopeByteValues, byte[ ][ ] scopeNumberOIDs, int [ ] scopeNumberValues, byte[ ] scopeKeyOID, Key scopeKeyValue) throws CSSMException

Removes the signature from the in-memory (cached) certificate.

Parameters:

context - The security context that describes the environment and parameters under which the cryptographic signing operations should be performed.

signerCert - Signer's certificate.

scopeByteOIDs - An array of field IDs that identify which byte array fields of the certificate should be unsigned. This parameter is optional.

scopeByteValues - An array of byte array field values corresponding to the scopeByteOIDs.

scopeNumberOIDs - An array of field IDs that identify which int fields of the certificate should be unsigned. This parameter is optional.

scopeNumberValues - An array of int field values corresponding to the scopeNumberOIDs.

scopeKeyOID - Field ID that identify key field of the certificate should be unsigned. This parameter is optional.

scopeKeyValue - Key value corresponding to the scopeKeyOID.

Returns:

The newly-unsigned certificate.

Throws: CSSMException

- The security context is invalid
- No signing certificate was specified
- CL doesn't support this function
- Invalid CL

public Key getPublicKey ( )

Gets a copy of the public key from the certificate.

Returns:

Public key information. Return NULL if no public key is found.

public byte [ ] getFirstByteValue (byte[ ] cssmByteOID) throws CSSMException

Gets a copy of the first value of a field stored in the certificate. The selected field is specified by the cssmByteOID. This method should be used to retrieve certificate information that is stored as a string. Sets the resultsHandle variable, which is used to retrieve subsequent matches for the specified OID.

Parameters:

cssmByteOID - The ID of the certificate field whose value is to be retrieved.

Returns:

The value stored in the specified field.

Throws: CSSMException

- cssmByteOID is invalid

public byte[ ] getNextByteValue ( )
throws CSSMException

Gets the next value of a field stored in the certificate using the resultHandle variable that was initialized by invoking the getFirstByteValue method.

Returns:

The next field value of the selected field.

Throws: CSSMException

- cssmByteOID is invalid
- resultsHandle variable is invalid

public int getFirstNumberValue (byte[ ] cssmNumberOID)
throws CSSMException

Gets a copy of the first value of a field stored in the certificate. The selected field is specified by the cssmNumberOID. This method should be used to retrieve certificate information that is stored as a number. Sets the resultsHandle variable, which is used to retrieve subsequent matches for the specified OID.

Parameters:

cssmNumberOID - The ID of the certificate field whose value is to be retrieved.

Returns:

The value stored in the specified field.

Throws: CSSMException

- cssmNumberOID is invalid

public int getNextNumberValue ( )
throws CSSMException

Gets the next value of a field stored in the certificate using the resultHandle variable that was initialized by invoking the getFirstNumberValue method.

Returns:

The next field value of the selected field.

Throws: CSSMException

- cssmNumberOID is invalid
- resultsHandle variable is invalid

public void abortGetFieldValue ( )

Clears the cached results and terminates the query initiated by invoking the getFirstByteValue method or the getFirstNumberValue method.

public byte[ ][ ] getViewOIDs ( ) throws CSSMException

Returns all the displayable field IDs of the certificate.

Returns:
An array of displayable certificate field IDs.

Throws: CSSMException
- The security context is invalid
- Invalid certificate library
- Unrecognized certificate format
- Not enough memory
- CL doesn't support this function

public byte[ ][ ] getViewValues ( ) throws CSSMException

Returns all the field values of the certificate corresponding to the CSSM OIDs returned from getViewOIDs method. The values are in displayable format. Be aware that many native certificate data formats are not displayable. The certificate library translates only displayable formats.

Returns:
An array of field values corresponding to cssmOIDs returned from getViewOIDs method.

Throws: CSSMException
- The security context is invalid
- Invalid certificate library
- Unrecognized certificate format
- Not enough memory
- CL doesn't support this function

public Certificate importCert (int foreignCertType) throws CSSMException

Imports a certificate from the input format into the native format of the specified certificate library.

Parameters:
foreignCertType - The type of non-native certificate format.

Returns:
A new certificate in the native certificate format, whose values are initialized from this certificate object.

Throws: CSSMException
- The security context is invalid
- Invalid certificate library
- Unrecognized certificate format
- Not enough memory
- CL doesn't support this function

– Unable to import certificate

public Certificate exportCert (int targetCertType) throws CSSMException

Exports a certificate from the native format of the specified certificate library into the specified target certificate format.

Parameters:
targetCertType - The certificate type of the resulting certificate.

Returns:
A new certificate of the specified non-native certificate type, whose values are initialized from this certificate object.

Throws: CSSMException
- The security context is invalid
- Invalid certificate library
- Unrecognized certificate format
- Not enough memory
- CL doesn't support this function
- Unable to export certificate

public void save (OutputStream os) throws IOException

Writes the certificate bytes to the OutputStream. This OutputStream can be used to regenerate this certificate using the Certificate (CLRegistry cl, InputStream is) constructor.

Parameters:
os - OutputStream contains the certificate bytes.

protected void finalize ( ) throws Throwable

Free the resource allocated by CSSM, if not yet freed.

4.2 CLRegistry

Every Certificate Library module (CL) must be installed and registered with the CSSM Certificate Services Manager. The CSSM Certificate Services Manger maintains this registration information in the registration database. Applications may query the Services Manager to retrieve properties of the CL module as defined during installation. Users must use Operating System tools, provided by CL providers, to do the installation and registration.

The CLRegistry class provides methods to query CL registry information. These methods allow applications to:

- Find out what CLs have registered with CSSM
- Find out which CLs service a particular certificate type
- Find out which CLs service a particular certificate translation type
- Find out CL's version information

- Attach and detach with a particular CL module

Note that this class does not provide functions to install and register a CL with CSSM. Installation and registration of CLs must be performed via CSSM's C language API.

public final class CLRegistry
extends Object

Constructors:

public CLRegistry (byte[ ] clGUID, String clName) throws CSSMException

Constructs a CLRegistry object with the specified CL's global unique ID (GUID) and logical name. Save the GUID and logical name in the object. Sets CLHandle to NULL.

Parameters:

clGUID - The global unique ID of the CL.

clName - The logical name of the CL.

Variables:

private byte[ ] guid - CL's global unique ID.

private String CLName - CL's logical name.

private int CLHandle - The handle to the CSSM-cached data structure that corresponds to the CL.

Methods:

public static CLRegistry[ ] getList (int certType)

Gets a list of CLs that service the specified certificate type. If no certType is specified then the complete list of installed CLs is returned.

Parameters:

certType - The certificate type. This parameter may be null.

Returns:

A list of CLRegistry objects with the clGUIDs set to corresponding certificate libraries. Returns NULL if no CL is found.

protected int getHandle ( )

Gets the handle to the CSSM-cached data structure that corresponds to the CL. The handle may be obtained only after it has been created using the attach method.

Returns:

The CL handle.

public int getCertType ( )

Gets the certificate type associated with the CL in the registry.

Returns:

The certificate type.

public int [ ] getTranslationType ( )

Gets the translation types supported by a certificate library in the registry. This defines the import and export types supported by instances of the certificate class. (See methods import ( ) and export ( ) in the Certificate class.)

Returns:

A list of translation types that are supported by this CLRegistry object.

public int getMajorVersion ( )

Gets the certificate library's major version information.

Returns:

The certificate library's major version information.

public int getMinorVersion ( )

Gets the certificate library's minor version information.

Returns:

The certificate library's minor version information.

public String getName ( )

Gets the logical name of this certificate library.

Returns:

A string containing the name of this certificate library.

public byte[ ] getGUID ( )

Gets the GUID of this certificate library.

Returns:

The GUID of this certificate library.

public void attach (int majorVersion, int minorVersion) throws CSSMException

Attaches the application with the CL module and creates a handle associated with the CL module. If the available version of the CL module is compatible with the version level specified, saves the handle in CLHandle variable.

Parameters:

majorVersion **-** Input parameter specifies the major version number of the CL module the application is compatible with.

minorVersion - Input parameter specifies the minor version number of the CL module the application is compatible with.

Throws: CSSMException

– The CSSM external registry has been modified by an external installation procedure since the CLRegistry object was created, making the CLRegistry object invalid
– The major and minor version number of the caller is incompatible with the CL version

<u>public byte[ ][ ] describeCertFormat ( )</u>

Returns a list of the field identifiers used to describe the certificate type supported by the specified CL.

Returns:

A template description for certificates. Returns NULL if no CL is found.

<u>public byte[ ][ ] describeCRLFormat ( )</u>

Returns a list of the field identifiers used to describe the CRL supported by the specified CL.

Returns:

A template description for CRLs. Returns NULL if no CL is found.

<u>public boolean detach ( )</u>

Dynamic unload of this certificate library, invalidating the CLHandle. An application must invoke this method when the CLRegistry object is no longer needed. For each instance of class CLRegistry, CSSM caches a corresponding data structure. Invoking this method allows CSSM to delete the corresponding data structure and invalidate the CLHandle.

Returns:

True - Successfully detached.

False - Failed to detach.

<u>public byte[ ] passThrough (SecurityContext context, int passThroughId,</u>
<u>byte[ ][ ] inputParams) throws CSSMException</u>

Allows applications to call certificate library module-specific operations that have been exported. Such operations may include queries or services that are specific to the domain represented by the CL module.

Parameters:

context **-** The security context that describes the environment and parameters under which the cryptographic operations should be performed. This parameter can be NULL depending upon the operation ID.

passThroughId - The identifier assigned by the CL module to indicate the export function to perform.

InputParams - The input parameters.

Returns:

Output data from the pass through operation.

Throws: CSSMException

- Invalid CL
- CL does not support this function
- Unable to perform pass through operation
- Invalid security context

<u>protected void finalize ( ) throws Throwable</u>

Frees the resource allocated by CSSM, if it not yet freed.

4.3 CRL

This class provides the functions to create a memory-resident certificate revocation list (CRL) for use by a certificate authority (CA). Given this list, a certificate can be checked against it to determine if the certificate has been revoked. Certificates may be revoked before they expire. For example, the certificate owner may leave the domain (an Intel employee with an Intel employee certificate may leave Intel to work for another company), or the certificate may be temporarily suspended. This class should be used by CAs who wish to cache a list of revoked certificates.

<u>public final class CRL</u>
<u>extends Object</u>

Constructors:

<u>public CRL (CLRegistry cl) throws CSSMException</u>

Given a certificate library, generates an empty, memory-resident CRL of revoked certificates from database. Memory for the CRL is allocated and managed by the CSSM. The application must invoke the free method even though the CRL object is no longer needed; this allows CSSM to cleanup the memory cache.

Parameters:

cl - The certificate library to perform the operation.

Throws: CSSMException

- The certificate library is invalid

Variables:

private int crlHandle - The handle to the CSSM-cached data structure the corresponds to this CRL object.

private int clHandle - The handle to the CSSM-cached data structure that corresponds to the specified CLRegistry object.

private int resultHandle - The handle to the CSSM-cached query results retrieved by invoking the getFirstFieldValue method. This handle is used to retrieve subsequent matches based on the same query.

private CLRegistry clRegistry - The certificate library that will be used to perform the CRL operations.

Methods:

public int getHandle ( )

Gets the handle of this CRL object.

Returns:
The CRL handle.

protected int getCLHandle ( )

Gets the handle of the certificate library that was used to create this CRL object.

Returns:
The certificate library handle.

public CRL addCert (SecurityContext context, Certificate cert, Certificate revokerCert, int revokeReason) throws CSSMException

Adds a record representing the certificate to the CRL and uses the keys associated with the revoker's certificate to syntactically sign the new record added to the CRL.

Parameters:

context **-** The security context that describes the environment and parameters under which the cryptographic operation should be performed.

cert - The certificate to be added to the CRL object.

revokerCert - The certificate of the revoking agent.

revokeReason - The reason for revoking the certificate. The possible values are listed in Table 5.

Returns:
A new CRL.

Throws: CSSMException
- Context is invalid
- NULL certificate was specified
- NULL revoker's certificate was specified
- Invalid CRL
- Not enough memory
- Unable to add certificate to CRL

public CRL removeCert (Certificate cert) throws CSSMException

Removes a record representing the specified certificate from the CRL.

Parameters:
cert - The certificate to be removed to the CRL object.

Returns:
A new CRL.

Throws: CSSMException
- NULL certificate was specified
- Invalid CRL
- Not enough memory
- Unable to remove a revocation record from CRL

public CRL crlSign (SecurityContext context, Certificate signerCert,
byte[ ][ ] scopeByteOIDs, byte[ ][ ] scopeByteValues,
byte[ ][ ] scopeNumberOIDs, int [ ] scopeNumberValues) throws CSSMException

Performs the mechanical operation of signing the fields of in-memory CRL indicated in the signing Scope.

Parameters:
context - The security context that describes the environment and parameters under which the cryptographic signing operations should be performed.

signerCert - Signer's certificate.

scopeByteOIDs - An array of field IDs that identify which byte array fields of the CRL should be signed. This parameter is optional.

scopeByteValues - An array of byte array field values corresponding to the scopeByteOIDs.

scopeNumberOIDs - An array of field IDs that identify which int fields of the CRL should be signed. This parameter is optional.

scopeNumberValues - An array of int field values corresponding to the scopeNumberOIDs.

Returns:
A CRL containing the signer's signature.

Throws: CSSMException
- The security context is invalid
- Invalid CL
- No signing certificate was specified
- Signing scope is invalid
- Not enough memory to allocate the CRL
- Unable to sign CRL

<u>public boolean crlVerify (SecurityContext context, Certificate signerCert, byte[ ][ ] scopeByteOIDs, byte[ ][ ] scopeByteValues, byte[ ][ ] scopeNumberOIDs, int [ ] scopeNumberValues) throws CSSMException</u>

Performs the mechanical operation of verifying the signature on the fields of in-memory CRL indicated in verify Scope.

Parameters:

context - The security context that describes the environment and parameters under which the cryptographic verification operation should be performed.

signerCert - The signer's certificate.

scopeByteOIDs - An array of field IDs that identify which byte array fields of the CRL should be verified. This parameter is optional.

scopeByteValues - An array of byte array field values corresponding to the scopeByteOIDs.

scopeNumberOIDs - An array of field IDs that identify which int fields of the CRL should be verified. This parameter is optional.

scopeNumberValues - An array of int field values corresponding to the scopeNumberOIDs.

Returns:

True - Verify successfully.

False - The signature associated with the CRL is not verified.

Throws: CSSMException

- The security context is invalid
- Invalid CL
- No signing certificate was specified
- Invalid CRL
- Verify scope is invalid
- Not enough memory to allocate the CRL
- Unable to verify CRL

<u>public boolean isCertIn (Certificate cert) throws CSSMException</u>

Searches the CRL for a record corresponding to the certificate.

Parameters:

cert - The certificate to be located.

Returns:

True - Certificate is in the CRL.

False - Certificate is not is the CRL.

Throws: CSSMException

- Invalid CL
- Invalid certificate
- Invalid CRL

public byte[ ] getFirstFieldValue (byte[ ] cssmOID)
throws CSSMException

Returns the value of the designated CRL field.

Parameters:
cssmOID - The ID of the CRL field whose value is to be retrieved.

Returns:
The value stored in the specified field.

Throws: CSSMException
- Invalid CRL
- Invalid certificate library
- Invalid OID
- Unable to retrieve the field

public byte[ ] getNextFieldValue ()
throws CSSMException

Returns the value of the designated CRL field. Uses the resultHandle variable that was initialized by invoking the getFirstFieldValue method.

Returns:
The next field value of the selected field.

Throws: CSSMException
- Invalid CRL
- Invalid certificate library
- Unable to retrieve the field

public void abortGetFieldValue ( )

Clears the cached results and terminates the query initiated by invoking the getFirstFieldValue method.

public void free ( )

Frees the memory allocated and managed by CSSM to cache the CRL.

protected void finalize ( ) throws Throwable

Frees the resource allocated by CSSM, if it has not been freed yet.

4.4 Crypto

This class provides the methods to access the cryptographic functions provided by CSPs through the CSSM. These functions include:

- Sign blocks of contiguous data
- Verify the signature associated with one or more blocks of data
- Compute a message digest for a block of data

- Encrypt a block of data
- Decrypt a block of data
- Generate a public/private key pair
- Generate a random bit sequence of arbitrary length

Before an application requests these services from a CSP, it must create a security context for that CSP. The security context supplies the CSP, through CSSM, with the input parameters that CSP requires to perform correct cryptographic operations. For example, encryption and decryption require input parameters to set up or select keys. When the operation is completed, the application should invoke the delete method in SecurityContext class to free the CSSM context structure associated with the SecurityContext object.

For staged cryptographic operation, an initialization call precedes a series of update calls. A final operation call is used to terminate the operation. The staging initialization call provides information that may be useful in allocating buffers for the update call and final output. In the case of signatures and hash functions, the amount of memory for the result is returned. In the case of encryption algorithm, the output is typically a multiple of some block size.

public final class Crypto
extends Object

Constructors:

Crypto ( )

Constructs a crypto object.

Methods:

public static int encryptData (SecurityContext context, byte[ ] clearBuf, byte[ ] cipherBuf, byte[ ] remData) throws CSSMException

Encrypts the supplied data with information from the security context. This operation is not staged, and the input data must be a single byte array object.

Parameters:

context - The security context containing the input parameters required to perform the encryption operation.

clearBuf - The buffer containing the clear text data to be encrypted.

cipherBuf - Output parameter that returns the encrypted data.

remData - The last encrypted block containing padded data.

Returns:

The size of the encrypted data in bytes.

Throws: CSSMException

– The security context is invalid

public static int decryptData (SecurityContext context, byte[ ] cipherBuf, byte[ ] clearBuf, byte[ ] remData) throws CSSMException

Decrypts the supplied encrypted data using the algorithm specified by the security context. This operation is not staged and the input data must be a single byte array object.

Parameters:

context - The security context containing the input parameters required to perform the decryption operation.

cipherBuf - The buffer containing the encrypted data.

clearBuf - Output parameter that returns the decrypted data.

remData - The last decrypted block.

Returns:

The size of the decrypted data in bytes.

Throws: CSSMException

- The security context is invalid
- Unable to decrypt data
- Unable to allocate memory

public static byte[ ] generateRandom (SecurityContext context) throws CSSMException

Generates a random key value using the algorithm specified by the security context.

Parameters:

context - The security context containing the input parameters required to perform the random number generation operation.

Returns:

The random number.

Throws: CSSMException

- The security context is invalid
- Unable to generate random data
- Unable to allocate memory

public static byte[ ] keyExchGenParam (SecurityContext context, int paramBits) throws CSSMException

Generates key exchange parameter data for keyExchPhase1.

Parameters:

context - The security context containing the input parameters required to perform this cryptographic operation.

paramBits - Used to generate parameters for the key exchange algorithm. For example, Diffie-Hellman.

Returns:

The parameter data.

Throws: CSSMException
- The security context is invalid
- Unable to allocate memory
- Unable to generate exchange param data

public static byte[ ] keyExchPhase1 (SecurityContext context, byte[ ] params)
throws CSSMException

Phase 1 of the staged key exchange function.

Parameters:

context - The security context containing the input parameters required to perform this cryptographic operation.

params - The output data from keyExchGenParam.

Returns:

The parameter data.

Throws: CSSMException
- The security context is invalid
- Unable to allocate memory
- Unable to perform this function

public static byte[ ] keyExchPhase2 (SecurityContext context, byte[ ] params)
throws CSSMException

Phase 2 of the staged key exchange function.

Parameters:

context - The security context containing the input parameters required to perform this cryptographic operation.

params - The output data from keyExchPhase1 calls.

Returns:

An array containing the key.

Throws: CSSMException
- The security context is invalid
- Unable to allocate memory
- Unable to stage key exchange

public static Key generateKey (SecurityContext context)
throws CSSMException

Generates a public/private key pair or a symmetric key using the algorithm specified in the security context. The private key is securely and persistently stored by the CSP specified in the security context. The public key is returned.

Parameters:

context - The security context containing the input parameters required to perform this cryptographic operation.

Returns:

The public key or the symmetric key.

Throws: CSSMException

- The security context is invalid
- Unable to allocate memory
- Unable to generate keys

public static byte[ ] generateUniqueId (SecurityContext context)
throws CSSMException

Generates unique identification data.

Parameters:

context - The security context containing the input parameters required to perform the key generation operation.

Returns:

The unique ID.

Throws: CSSMException

- The security context is invalid
- Unable to allocate memory
- Unable to generate unique ID

public static byte[ ] signData (SecurityContext context, byte[ ] inputData)
throws CSSMException

Signs data with the private key associated with the public key and algorithm specified in the security context. This operation is not staged and the input data must be a single byte array object.

Parameters:

context - The security context containing the input parameters required to perform the key generation operation.

inputData - The buffer containing the data to be signed.

Returns:

An array containing the actual signature.

Throws: CSSMException

- The security context is invalid

public static void signDataInit (SecurityContext context) throws CSSMException

Initializes the staged sign data function. The object variables inBlockSize and outBlockSize are not set by this method.

Parameters:

context - The security context containing the input parameters required to perform the signing operation.

Throws: CSSMException
- The security context is invalid
- Unable to initialize staged sign data

public static void signDataUpdate (SecurityContext context, byte[ ] inputData) throws CSSMException

Inputs the next block of data to the staged sign data function.

Parameters:
context - The security context containing the input parameters required to perform the signing operation.

inputData - The buffer containing data to be signed.

Throws: CSSMException
- The security context is invalid
- Unable to update data

public static byte[ ] signDataFinal (SecurityContext context) throws CSSMException

Finalizes the staged sign data function.

Parameters:
context - The security context containing the input parameters required to perform the signing operation.

Returns:
An array containing the final signature computed over all the data submitted for signing through the staged sign data function.

Throws: CSSMException
- The security context is invalid
- Unable to allocate memory
- Unable to sign data

public static boolean VerifyData (SecurityContext context, byte[ ] inputData, byte[ ] signature) throws CSSMException

Verifies that the signature was derived at some earlier time by applying to the input data using the signaturing algorithm specified in the security context using the key associated with the security context. This operation is not staged and the input data must be a single byte array object.

Parameters:
context - The security context containing the input parameters required to perform the verify operation.

inputData - Data that was allegedly signed to produce the signature.

signature - Signature associated with the input data.

Returns:
True - Verify successfully.

False - The signature is not verified.

Throws: CSSMException
- The security context is invalid
- Unable to verify data with signature

public static void VerifyDataInit (SecurityContext context, byte[ ] signature) throws CSSMException

Initializes the staged verify data function.

Parameters:

context - The security context containing the input parameters required to perform the verify operation.

signature - The signature to be compared with the finalized signature from the verification calculation.

Throws: CSSMException
- The security context is invalid
- Unable to initialize verify data

public static void VerifyDataUpdate (SecurityContext context, byte[ ] inputData) throws CSSMException

Inputs the next block of data for the staged verify data function.

Parameters:

context - The security context containing the input parameters required to perform the verify operation.

inputData - The data associated with the signature.

Throws: CSSMException
- The security context is invalid
- Unable to update verify data

public static boolean VerifyDataFinal (SecurityContext context) throws CSSMExceptionion

Final stage of verify data function. The intermediate signature calculation that has been performed by the verifyDataUpdate method is finalized and the resulting signature is compared with the signature specified as a parameter.

Parameters:

context - The security context containing the input parameters required to perform the verify operation.

Returns:

True - Verify successfully.

False - The signature is not verified.

Throws: CSSMException

- The security context is invalid
- Unable to verify the data

public static byte[ ] digestData (SecurityContext context, byte[ ] inputData) throws CSSMException

Computes a message digest of the input data. This operation is not staged and the input data must be a single byte array object.

Parameters:

context - The security context containing the input parameters required to perform the digest operation.

inputData - Input data to the hash/digest operation.

Returns:

An array containing the digest.

Throws: CSSMException

- The security context is invalid

public static void digestDataInit (SecurityContext context) throws CSSMException

Initializes the staged message digest function. Also, the size of the final digest is determined by the CSP, based on the algorithm and other crypto-parameters specified in the security context, and returned by this method.

Parameters:

context - The security context containing the input parameters required to perform the digest operation.

Throws: CSSMException

- The security context is invalid
- Unable to perform digest initialization

public static void digestDataUpdate (SecurityContext context, byte[ ] inputData) throws CSSMException

Inputs the next block of data to the staged message digest function.

Parameters:

context - The security context containing the input parameters required to perform the digest operation.

inputData - The next block of input data.

Throws: CSSMException

- The security context is invalid
- Unable to perform digest on data

public static byte[ ] digestDataFinal (SecurityContext context) throws CSSMException

Finalizes the staged message digest function.

Parameters:
context - The security context containing the input parameters required to perform the digest operation.

Returns:
An array containing the digest.

Throws: CSSMException
- The security context is invalid
- Unable to allocate memory
- Unable to perform digest on data

public static SecurityContext digestDataClone (SecurityContext context)
throws CSSMException

Creates a new security Context object that clones the current state of a given staged message digest function. The current state consists of the cryptographic attributes and intermediate digest result in the security context. The returned context object may be used to continue a staged digest operation with arbitrary follow-on input data via the digestDataUpdate method.

Parameters:
context - The security context containing the input parameters required to perform the digest operation.

Returns:
A new security context object which clones the input security context. Returns NULL if an error has occurred.

Throws: CSSMException
- The security context is invalid
- Unable to clone context

public static void encryptDataInit (SecurityContext context) throws CSSMException

Initializes the staged encrypt function.

Parameters:
context - The security context containing the input parameters required to perform the encryption operation.

Throws: CSSMException
- The security context is invalid
- Unable to encrypt data

public static int encryptDataUpdate (SecurityContext context, byte[ ] clearBuf, byte[ ] cipherBuf) throws CSSMException

Updates the staged encrypt function. There may be algorithm-specific and token-specific rules restricting the lengths of data in encryptDataUpdate methods which make use of the block sizes returned by the setSize method.

Parameters:

context - The security context containing the input parameters required to perform the encryption operation.

clearBuf - The clear text data to be encrypted.

cipherBuf - Output parameter that returns the encrypted data.

Returns:

The size of the encrypted data in bytes.

Throws: CSSMException

- The security context is invalid
- Unable to encrypt data

public static byte[ ] encryptDataFinal (SecurityContext context) throws CSSMException

Finalizes the staged encrypt function.

Parameters:

context - The security context containing the input parameters required to perform the encryption operation.

Returns:

The last encrypted block containing padded data.

Throws: CSSMException

- The security context is invalid
- Unable to allocate memory
- Unable to encrypt data

public static void decryptDataInit (SecurityContext context) throws CSSMException

Initializes the staged decrypt function.

Parameters:

context - The security context containing the input parameters required to perform the decryption operation.

Throws: CSSMException

- The security context is invalid
- Unable to initialize the stage decryption

public static int decryptDataUpdate (SecurityContext context, byte[ ] cipherBuf, byte[ ] clearBuf) throws CSSMException

Updates the staged decrypt function. There may be algorithm-specific and token-specific rules restricting the lengths of data in decryptDataUpdate methods which make use of the block sizes returned by the setSize method.

Parameters:

context - The security context containing the input parameters required to perform the decryption operation.

cipherBuf - The encrypted data.

clearBuf - Output parameter that returns the decrypted data.

Returns:

The size of the decrypted data in bytes.

Throws: CSSMException

- The security context is invalid or unable to decrypt data

public static byte[ ] decryptDataFinal (SecurityContext context) throws CSSMException

Finalizes the staged decrypt function. This is a cleanup function. No additional decrypted data is returned.

Parameters:

context - The security context containing the input parameters required to perform the decryption operation.

Returns:

the last decrypted block.

Throws: CSSMException

- The security context is invalid
- Unable to encrypt data

public static int getOutputBlockSize (SecurityContext context, int inputBlockSize)

Given the size of the input data, 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. For encryption, the total size of all output buffers must always be a multiple of the block size. This function can also be used to query the output size requirements for the intermediate steps of a staged cryptographic operation (for example, **encryptDataUpdate** and **decryptDataUpdate**). There may be algorithm-specific and token-specific rules restricting the lengths of data following data update calls.

4.5 CSPRegistry

This class provides functions to query Cryptographic Service Providers (CSPs) regarding the cryptographic services they provide. The queries allow the application to:

- Find out what CSPs have registered with CSSM
- Find out what services a particular CSP is capable of performing
- Find out the version numbers
- Attach and detach with a CSP

All Cryptographic Service Providers (CSPs) must register themselves and their services with the CSSM Cryptographic Services Manager before applications can access them. User can use tools, provided by CSP vendors, to perform CSP installation and registration.

Before an application calls a CSP to perform a cryptographic operation, it uses the query service to determine what CSPs are installed and what they can do.

public final class CSPRegistry
extends Object

Constructor:

public CSPRegistry (byte[ ] cspGUID, String cspName) throws CSSMException

Constructs a CSP with the specified CSP's global unique ID (GUID) and logical name. Both input parameters are required. Use the static method getList ( ) to obtain the list of all CSPs and search the list for possible matches based on a GUID only or a CSPName only. This constructor saves the GUID and name in the object and sets CSPHandle to NULL.

Parameters:

cspGUID- The global unique ID of the CSP.

cspName - The logical name of the CSP.

Variables:

private byte[ ] guid - CSP's global unique ID.

private String CSPName - The logical name of this CSP object.

private int CSPHandle - The handle to the CSSM-cached data structure that corresponds to this CSP object.

Methods:

protected int getHandle ( )

The handle to the CSSM-cached data structure corresponding to this CSP object.

Returns:

The CSP handle.

public static CSPRegistry[ ] getList ( )

Gets a list of cryptographic service providers (CSPs) registered with CSSM. CSPs must be registered and installed using OS tools on the platform or by installation utilities implemented in the C language.

Returns:

A list of CSP objects. Returns NULL if no CSP is found.

public static CSPRegistry[ ] getAlgorithmProviders (int contextType, int algorithmID)

Gets a list of CSPs that support the specified context and algorithm types.

Parameters:

contextType - The context type. The possible values are listed in Table 1.

algorithmID - The algorithm type. If it is NULL, all the algorithm types will be searched. The possible values are listed in Table 2.

Returns:

A list of CSP objects that support the specified context and algorithm. Returns NULL if no CSP is found.

public String getName ( )

Gets the logical name of this CSP.

Returns:

A string contains the name of this CSP.

public byte[ ] getGUID ( )

Gets the GUID of this CSP.

Returns:

The global unique ID of this CSP.

public String getDescription ( )

Gets the description of this CSP.

Returns:

A string contains the description this CSP.

public int[ ] getContextType ( )

Gets a list of context types that are supported by this CSP.

Returns:

A list of context types. Returns NULL if no context type is found.

public String getVendorName ( )

Gets this CSP's vendor name.

Returns:

Vendor name.

public int[ ] getAlgorithms (int contextType)

Gets a list of algorithms supported by this CSP within the specified context type.

Parameters:
contextType - The context type.

Returns:
A list of algorithm IDs of this CSP for the specified context.

public int getMajorVersion ( )

Gets the major version number of this CSP object.

Returns:
Major version number.

public int getMinorVersion ( )

Gets the minor version number of this CSP object.

Returns:
Minor version number.

public void attach (int majorVersion, int minorVersion) throws CSSMException

Creates a handle associated with the CSP. If the available version of the CSP module is compatible with the version level specified, saves the handle in CSPHandle variable.

Returns:
majorVersion - Input parameter specifies the major version number of the CSP module that application is compatible with.

minorVersion - Input parameter specifies the minor version number of the CSP module that application is compatible with.

Throws: CSSMException
- The CSSM external registry has been modified by an external installation procedure since the CSPRegistry object was created making the CSPRegistry object invalid
- The major and minor version number of the caller is incompatible with the CSP version

public boolean detach ( )

Deletes the CSP object, invalidating the CSPHandle. An application must invoke this method when the CSP object is no longer needed. For each instance of class CSP, CSSM caches a corresponding data structure. Invoking this method allows CSSM to delete the corresponding data structure and invalidate the CSP handle.

Returns:

True - Successfully detached.

False - Failed to detach.

public byte[ ] passThrough (int passThroughId, SecurityContext context,
byte[ ][ ] inputParams) throws CSSMException

Accepts as input an operation ID and a set of arbitrary input parameters. The operation ID may specify any type of operation the CSP wishes to export for use by an application.

Parameters:

passThroughId - An identifier specifying the exported function to be performed.

context - The security context containing the input parameters required to perform the pass through operation.

inputParams - The input parameters.

Returns:

Output data from the pass through operation.

Throws: CSSMException

- Invalid CSP
- Invalid security context
- Invalid pass through ID
- Unable to perform pass through function

protected void finalize ( ) throws Throwable

Free the resource allocated by CSSM, if it not yet freed.

4.6 DataStore

This class provides functions to create, query, and update a certificate/CRL data store. The certificate/CRL methods defined in the Certificate class are carried out on memory-based certificate/CRL objects. Memory-based certificate/CRL objects can be initialized from persistent certificate/CRL objects or from gathered, non-persistent values. This class moves certificate/CRL object between memory and persistent storage.

Certificates and CRLs must be securely stored as persistent entities on one or more systems. Multiple certificate/CRL data stores may exist on a single system. For example, it may be necessary to create one certificate/CRL data store per end-user on a multi-user system. Also, applications may wish to create a certificate/CRL data store specific to that application.

public final class DataStore
extends Object

Constructors:

public DataStore (DLRegistry dl, String datastoreName) throws CSSMException

Constructs a certificate/CRL DataStore object with the specified data store name. The data store name is saved in the object and data store handle is set to NULL.

Parameters:

dl **-** Handle of the data storage library that will perform the database operations.

datastoreName - Data store name of this object.

Throws: CSSMException

– If data store name is NULL or data storage library is NULL.

Variables:

private int DLHandle - The handle to the CSSM-cached data structure that corresponds to the specified data storage library.

private String DSName - Data store name of this object.

private int DSHandle - The handle to the CSSM-cached data structure that corresponds to this data storage object.

Methods:

protected int getDataStoreHandle ( )

Gets the handle of this data store object.

Returns:

The data store handle.

protected int getDLHandle ( )

Gets the handle of the specified data storage library object.

Returns:

The data storage library handle.

public String getDataStoreName ( )

Gets the name of this data store object.

Returns:

A string contains the name of this data store.

public void createNew (CLRegistry cl) throws CSSMException

Creates and opens a data store of the name specified by the object. The data store handle is saved in datastoreHandle variable. If the specified name already exists, the data store is opened.

Throws: CSSMException

– Invalid DL

- Invalid CL
- Unable to create new data store

public boolean delete ( )

Deletes all the records from the data store and removes the data store from the list of known data store names.

Returns:

True - Successfully deleted.

False - Failed to delete.

public void open ( ) throws CSSMException

Opens the data store for accessing and saves the data storage handle in databaseHandle variable.

Throws: CSSMException

- Data store name not found
- Invalid DL

public boolean close ( )

Closes the data store.

Returns:

True - Successfully closed the data store.

False - Failed to close the data store.

public void imortDS (String DSlogicalName, String DSFileName)
throws CSSMException

Accepts as input a filename and a logical name for a data store. The file is an exported copy of a certificate data store or a CRL data store that is being presented for import to this data store object. The certificates or CRLs contained in the file must be in the native format of the data storage library (DL). The DL imports all certificate or CRL records in the file, creating a new certificate data store or CRL data store, respectively. The specified logical name is assigned to the new data store. Note: This mechanism can be used to copy data stores among systems or to restore a persistent data store of certificates or CRLs from a back-up copy.

Parameters:

DSLogicalName **-** The logical name of a data store to be created.

DSFileName - The name of the file containing the data to be imported.

Throws: CSSMException

- DSLogicalName is NULL
- DSFileName is NULL

public void exportDS (String DSlogicalName, String DSFileName) throws CSSMException

Accepts as input the logical name of a data store and the name of a file. The specified data store contains persistent certificates or persistent CRL records. A copy of the records are exported from the data store, creating a file of the specified name. The file may be imported on the same or another system at a later time. Note: This mechanism can be used to copy data stores among systems or to create a back-up of persistent data stores for certificates and CRLs.

Parameters:

DSLogicalName **-** The logical name of a data store from which the data should be copied for export.

DSFileName - The name of a file to be created to hold the exported data.

Throws: CSSMException

- DSLogicalName is NULL
- DSFileName is NULL

public void addCertificate (Certificate cert) throws CSSMException

Adds a certificate to the data store, or updates an existing certificate in the data store.

Parameters:

cert - Certificate to be added to the data store.

Throws: CSSMException

- Invalid DataStore
- Invalid certificate

public void deleteCertificate (Certificate cert) throws CSSMException

Deletes a certificate from the data store.

Parameters:

cert - Certificate to be deleted from the data store.

Throws: CSSMException

- Invalid DataStore
- Invalid certificate

public int getFirstCertificate (byte[ ][ ] cssmOIDs, byte[ ][ ] fieldValues, int [ ] dbOperators, int dbConjunctive, Certificate cert) throws CSSMException

Retrieves the first certificate from the data store based on the criteria given in the cssmOID, filedValue, dbOperators and dbConjunctive. The CSSM allocates memory for a certificate. After using the memory-based copy of the certificate, the caller must free the memory allocated for this copy by invoking the free method.

Parameters:

cssmOIDs - Array of fields IDs that identify the data used to search the certificate.

fieldValues - The field values of the specified cssmOIDs.

dbConjunctive - Conjunctive operator.

cert - Must be an empty certificate object. The retrieved certificate is returned in this object. Return NULL certHandle in the Certificate object if there is no certificate that matches the search criteria.

Returns:
A results handle used to get subsequent certificates that matched the selection criteria specified in this method. (See getNextCertificate ( ).)

Throws: CSSMException
- Invalid DataStore
- Invalid DL
- Unable to get first record

public void getNextCertificate (int resultHandle, Certificate cert) throws CSSMException

After an initial call to getFirstCertificate, this method retrieves the next certificate from the data store matching the search criteria established in the initial call.

The application must call getFirstCertificate before calling this method. The CSSM keeps track of the previous search criteria used by the application and applies it again to find the next certificate matching the same criteria.

If a match is found, CSSM allocates the memory for the copy of the certificate. After using the copy of the certificate object, the caller must invoke the free method to free the memory.

When no additional certificates match the given criteria, a NULL is returned.

Parameters:
resultHandle - A results handle from getFirstCertificate ( ) used to get subsequent certificates matching the selection criteria specified in this method.

cert - Must be an empty certificate object. The retrieved certificate is returned in this object.

Returns:
A certificate. Return NULL if no more certificates match the given criteria.

Throws: CSSMException
- Invalid DataStore
- Invalid DL
- Unable to get next record

public void abortGetCertificate (int resultHandle)

Clears the cached results and terminates the query initiated by invoking the getFirstCertificate method.

Parameters:
resultHandle - A results handle from the getFirstCertificate. It indicates the query to be terminated.

public void revokeCertificate (Certificate revokedCert) throws CSSMException

Updates the data store with the revoked certificate, which was revoked using the certRevoke method in the Certificate class.

Parameters:
revokedCert - The revoked certificate to be updated or added to the data store. The memory-resident certificate must already have been revoked.

Throws: CSSMException
- Invalid DL
- Invalid DataStore
- NULL revoked certificate

public void addCRL (CRL revocationList) throws CSSMException

Adds a CRL record to or updates an existing certificate in the data store.

Parameters:
revocationList - The revocation list to be added.

Throws: CSSMException
- Invalid DL
- Invalid DataStore
- CRL is NULL

public void deleteCRL (CRL revocationList) throws CSSMException

Deletes a CRL record from the data store.

Parameters:
revocationList - The revocation list to be deleted.

Throws: CSSMException
- Invalid DL
- Invalid DataStore
- CRL is NULL

public int getFirstCRL (byte[ ][ ] cssmOIDs, byte[ ][ ] fieldValues,
int [ ] dbOperators, int dbConjunctive, CRL selectedCRL) throws CSSMException

Retrieves the first CRL record from the data store based on the criteria given in the cssmOID, filedValue, dbOperators and dbConjunctive. The CSSM allocates memory for a CRL. After using the memory-based copy of the CRL, the caller must free the memory allocated for this copy by invoking the free method.

Parameters:
cssmOIDs - Array of fields IDs that identify the data to be used to search the CRL.

fieldValues - The field values of the specified cssmOIDs.

dbConjunctive - Conjunctive operator.

cert - Must be an empty CRL object. The retrieved CRL is returned in this object. Return NULL crlHandle in the Certificate object, if there is no record in the data store matching the search criteria.

Returns:

A results handle that should be used to get subsequent CRL matching the selection criteria specified in this method. (See getNextCRL ( ).)

Throws: CSSMException

- Invalid DataStore
- Invalid DL
- Unable to get first record

public void getNextCRL (int resultsHandle, CRL revocationList)
throws CSSMException

After an initial call to getFirstCRL, this method retrieves the next CRL record from the data store matching the search criteria established in the initial call.

The application must call getFirstCRL before calling this method. The CSSM keeps track of the previous search criteria used by the application and applies it again, to find the next certificate matching the same criteria.

If a match is found, CSSM allocates the memory for the copy of the CRL. After using the copy of the CRL object, the caller must invoke the free method to free the memory.

When no additional certificates match the given criteria, a NULL is returned.

Parameters:

resultHandle - A results handle from getFirstCRL ( ) that used to get subsequent certificates matching the selection criteria specified in this method.

revocationList - Must be an empty revocation list to hold the returned list.

Returns:

A CRL. Return NULL if no more certificates match the given criteria.

Throws: CSSMException

- Invalid DataStore
- Invalid DL
- Unable to get next record

public void abortGetCrl (int resultHandle)

Clears the cached results and terminates the query initiated by invoking the getFirstCRL method.

Parameters:

resultHandle - A results handle from the getFirstCRL. It indicates the query to be terminated.

4.7 DLRegistry

The primary purpose of a Data Storage Library (DL) module is to provide persistent storage of certificates and certificate revocation lists (CRLs). A single DL module may be tightly tied to a CL or may be independent of all CLs. A Data Storage Library that is tightly tied to a certificate library module would implement a persistent storage mechanism that was dependent on the data format of the certificate. An independent Data Storage Library would implement a *blob-based* storage mechanism, storing certificates and CRLs without regard for their specific format. A single, physical data store managed by such DL modules may contain certificates of multiple formats.

Every DL must be installed and registered with the CSSM Data Store Services Manager, that maintains this information in the registration database. Applications may query the Services Manager to retrieve properties of the DL module as defined during installation. Users must use OS tools, provided by DL providers, to do the installation and registration.

The DLRegistry class provides methods to query DLs registry information. These methods allow applications to:

- Find out what DLs have registered with CSSM
- Find out a DL's version information
- Get the handle of a particular DL
- Get a list of logical data store names that a particular DL can access
- Attach and detach with a particular DL

Note that this class does not provide functions to install and register a DL with CSSM. Installation and registration of DLs must be performed via CSSM's C language API.

public final class DLRegistry
extends Object

Constructors:

public DLRegistry (byte[ ] dlGUID, String dlName) throws CSSMException

Constructs a DLRegistry with the specified DL name and GUID. Saves the DL name and GUID in the object and sets the DL handle to NULL.

Parameters:

dlGUID - The global unique ID of the DL.

dlNname - The name of the DL.

Variables:

private String DLName - The name of this DL object.

private byte[ ] guid - The global unique ID of this DL object.

private int DLHandle - The handle to the CSSM-cached data structure corresponding to this DL object.

Methods:

protected int getHandle ( )

Gets the handle to the CSSM-cached data structure corresponding to this DL object.

Returns:
The DL handle.

public int getMajorVersion ( )

Gets the major version number of this DL object.

Returns:
Major version number.

public int getMinorVersion ( )

Gets the minor version number of this DL object.

Returns:
Minor version number.

public static DLRegistry[ ] getList ( )

Gets a list of the DLs currently installed. DLs must be registered and installed using OS tools on the platform or by installation utilities implemented in the C language. The DL List cannot be set from a Java application/applet.

Returns:
A list of DL objects. Return NULL if no DL is found.

public String[ ] getDSNames ( )

Gets a list of logical data store names that this DL can access.

Returns:
A list of logical data store names that this DL can access.

public String getName ( )

Returns:
A string containing the name of this DL.

public byte[ ] getGUID ( )

Returns:
The global unique ID of this DL.

public void attach (int majorVersion, int minorVersion) throws CSSMException

Creates a handle associated with the DL and saves it in DLHandle variable.

Returns:

majorVersion - Input parameter specifies the major version number of the DL module that the application is compatible with.

minorVersion - Input parameter specifies the minor version number of the DL module that the application is compatible with.

Throws: CSSMException

- Invalid DL
- Incompatible version
- Unable to attach to DL module

public boolean detach ( )

Deletes the DL object, invalidating the DL handle. An application must invoke this method when the DL object is no longer needed. For each instance of class TP, CSSM caches a corresponding data structure. Invoking this method allows CSSM to delete the corresponding data structure and invalidate the DL handle.

Returns:

True - Successfully detached.

False - Failed to detach.

public byte[ ] passThrough (DataStore ds, int passThroughId, byte[ ][ ] inputParams) throws CSSMException

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 the DL module.

Parameters:

ds - The handle to the data store that this operation should be performed upon.

passThroughId - An identifier specifying the exported function to be performed.

InputParams - The input parameters.

Returns:

Output data from the pass through operation.

Throws: CSSMException

- Invalid DL
- Invalid DataStore
- Invalid pass through ID
- Memory error
- Unable to perform pass through function

protected void finalize ( ) throws Throwable

Free the resource allocated by CSSM, if it not yet freed.

4.8 Key

This class provides functions to retrieve the key information (such as key length and key bytes) from key objects. An instance of class key can be an asymmetric key or a symmetric key. Symmetric cryptographic algorithms use exactly one key, the symmetric key, which is stored in an instance of class key. Asymmetric cryptographic algorithms use public/private key pairs. The public key is stored in an instance of class key. The associated private key is securely stored by the CSP that generated the key pair.

public final class Key
extends Object

Constructors:

public Key ( )

Constructs a key object.

public Key (int keyLengthBits, byte[ ] keyData, int keyBlobType, int keyBlobFormatVersion, int keyAlgorithmId, int keyAlgorithmMode, byte[ ] CSPID, int keyWrapMethod)

Constructs a key object and saves the input data in the object.

Parameters:

keyLengthBits - The key length in bits.

keyData - Key Data bytes.

keyBlobType - The key blob type. The key blob types currently defined are CSSM_SESSION_KEY_BLOB, CSSM_RSA_PUBLIC_KEY_BLOB, CSSM_RSA_PRIVATE_KEY_BLOB, CSSM_DSA_PUBLIC_KEY_BLOB, and CSSM_DSA_PRIVATE_KEY_BLOB.

keyBlobFormatVersion - Version number of the key blob format. Current value is 0x01.

keyAlgorithmId - Algorithm identifier for the key contained by the key blob.

keyAlgorithmMode - Algorithm mode for the key contained by the key blob.

CSPID - Globally-unique ID of the CSP that generated the key (if appropriate).

keyWrapMethod - The method for wrapping the secret key. The methods currently defined are: CSSM_KEYWRAP_MD5WITHDES, CSSM_KEYWRAP_MD5WITHIDEA, CSSM_KEYWRAP_SHAWITHDES, AND CSSM_KEYWRAP_SHAWITHIDEA.

Variables:

private int keyLengthInBits

private byte[ ] key // a symmetric key or a public key from a public/private key pair.

private int blobType

private int formatVersion

private int algorithmID

private int algorithmMode

private byte[ ] cspID

private in wrapMethod

Methods:

public int getKeyWrapMethod ( )

Gets the the method for wrapping the secret key.

Returns:

The wrapping method.

public int getKeyLengthBits ( )

Gets the key length, in bits, of this key object.

Returns:

Number of bits in the key.

public byte[ ] getKey ( )

Gets the key bytes.

Returns:

The key.

public int getBlobType ( )

Gets key blob type.

Returns:

Key blob type.

public int getFormatVersion ( )

Gets the version number of the key blob format.

Returns:

Version number of the key blob format.

public int getAlgorithmId ( )

Gets algorithm identifier for the key contained by the key blob.

Returns:

Algorithm ID for the key.

public int getAlgorithmMode ( )

Gets algorithm mode for the key contained by the key blob.

Returns:
Algorithm mode for the key.

public byte[ ] getCspId ( )Gets the globally-unique ID of the CSP that generated the key (if appropriate).

Returns:
CSP ID.

4.9 SecurityContext

This class provides functions to create and query a security context.

An application/applet must create a security context object before requesting cryptographic services from a Cryptographic Service Provider (CSP). The security context object is used to communicate cryptographic parameters to the CSP. The application/applet must specify parameters for the cryptographic operations it needs to use, including:

- Context type (asymmetric encryption, symmetric encryption, signaturing, etc.)
- Algorithm type (RSA, DES, etc.)
- Optional attributes type (e.g., public key and padding information)

Once created and initialized, the application should use the security context to perform the proper cryptographic operations. After the application/applet has completed the operations, it should delete the security context as soon as possible.

Security context information is not persistent; it is not saved permanently in a file or a database. So, security context information will be gone after the CSSM is unloaded.

The constants defined for specifying service types, algorithm types, and attribute types are defined at the end of this class definition. These constants must be used when specifying context, algorithm, and attribute types as parameters to any method defined in the SecurityContext class. These constants should also be used to test results returned by methods of the class CSP.

public final class SecurityContext extends Object

Constructors:

public SecurityContext (CSPRegistry csp, int contextType, int algorithmID) throws CSSMException

This constructor creates a security context with specified CSP, context type, and algorithm ID for message digesting.

Parameters:
csp - The CSP that will perform the subsequent cryptographic operations.

contextType - The context type. The value must be CSSM_ALGCLASS_DIGEST. **This input parameter is required**. Throws CSSMException if this is NULL.

algorithmID - The algorithm ID. The possible values are listed in Table 2. **This input parameter is required.** Throws CSSMException if this is NULL or any other value.

public SecurityContext (CSPRegistry csp, int contextType, int algorithmID, int parameter) throws CSSMException

This constructor creates a security context with specified CSP, context type, algorithm ID and parameter for key exchanging.

Parameters:

csp - The CSP that will perform the subsequent cryptographic operations.

contextType - The context type. The value must be CSSM_ALGCLASS_KEYXCH. **This input parameter is required**. Throws CSSMException if this is NULL or any other value.

algorithmID - The algorithm ID. The possible values are listed in Table 2. **This input parameter is required.** Throws CSSMException if this is NULL.

parameter - Optional input parameter for key exchange. Zero may be specified.

public SecurityContext (CSPRegistry csp, int contextType, int algorithmID, int algorithmMode, Key keys, byte[ ] initVector, byte[ ] padding, int loops) throws CSSMException

This constructor creates a security context with specified CSP, context type, algorithm ID, key mode, keys, vector, padding and loop information for symmetric encryption/decryption.

Parameters:

csp - The CSP that will perform the subsequent cryptographic operations.

contextType - The context type. Specified CSSM_ALGCLASS_SYMMETRIC. **This input parameter is required**. Throws CSSMException if this is NULL or any other value.

algorithmID - The algorithm ID. The possible values are listed in Table 2. **This input parameter is required.** Throws CSSMException if this is NULL.

algorithmMode - The mode to be used by a block symmetric cipher. The possible values are listed in Table 4. If the specified algorithm is not a block cipher this input parameter need not be specified.

keys - The keys to be used for performing the cryptographic operations.

initVector - The initial vector for symmetric encryption. Typically specified for block ciphers.

padding - The method for padding.

loops - Specifies the number of rounds of encryption. Used for ciphers with variable number of rounds, for example, RC5.

public SecurityContext (CSPRegistry csp, int contextType, int algorithmID, Key key, byte[ ] padding, int keyMode, String passPhrase) throws CSSMException

This constructor creates a security context with specified CSP, context type, algorithm ID, keys, vector, padding and keyMode information for asymmetric encryption and decryption.

Parameters:

csp - The CSP that will perform the subsequent cryptographic operations.

contextType - The context type. Specified CSSM_ALGCLASS_ASYMMETIRC. **This input parameter is required**. Throws CSSMException if this is NULL.

algorithmID - The algorithm ID. The possible values are listed in Table 2. **This input parameter is required.** Throws CSSMException if this is NULL.

key - The key to be used for performing the cryptographic operations.

padding - The method for padding.

keyMode - Indicates whether to use the private or public key.

passPhrase - String containing pass word to wrap the private key.

public SecurityContext (CSPRegistry csp, int contextType, int algorithmID, int szOfRandom, byte[ ] seed) throws CSSMException

This constructor creates a security context with specified CSP, context type, algorithm ID, size of random number and seed information for generating random numbers.

Parameters:

csp - The CSP that will perform the subsequent cryptographic operations.

contextType - The context type. The value must be CSSM_ALGCLASS_RANDOMGEN. **This input parameter is required**. Throws CSSMException if this is NULL or any other value.

algorithmID - The algorithm ID. The possible values are listed in Table 2. **This input parameter is required.** Throws CSSMException if this is NULL.

szOfRandom - Size of random number to be generated.

seed - Seed to be used for generating the random number.

public SecurityContext (CSPRegistry csp, int contextType, byte[ ] seed, int algorithmID, int szOfUniqueID) throws CSSMException

This constructor creates a security context with specified CSP, context type, seed, algorithm ID and size of unique ID for generating a unique identifier.

Parameters:

csp - The CSP that will perform the subsequent cryptographic operations.

contextType - The context type. The value must be CSSM_ALGCLASS_UNQUEGEN. **This input parameter is required**. Throws CSSMException if this is NULL or any other value.

seed - Seed to be used for generating the random number.

algorithmID - The algorithm ID. The possible values are listed in Table 2. **This input parameter is required.** Throws CSSMException if this is NULL.

szOfUniqueID - Size of unique identifier to be generated.

public SecurityContext (CSPRegistry csp, int contextType, int algorithmID, int modulusSize, int keySizeInBits, String passPhrase, byte[ ] seed, byte[ ] salt) throws CSSMException

This constructor creates a security context with specified CSP, context type, algorithm ID, modulus size, keySizeInBit, pass phrase, seed and salt for generating a public/private key pair or a symmetric key.

Parameters:

csp - The CSP that will perform the subsequent cryptographic operations.

contextType - The context type. Specified CSSM_ALGCLASS_KEYGEN. **This input parameter is required**. Throws CSSMException if this is NULL or any other value.

algorithmID - The algorithm ID. The possible values are listed in Table 2. **This input parameter is required.** Throws CSSMException if this is NULL.

modulusSize - Size of modulus to be used. **This input parameter is required for generating a public/private key pair.**

keySizeInBits - Size of key in bits. **This input parameter is required for generating a symmetric key.**

passPhrase - String containing pass word to wrap the private key. In subsequent operations this passphrase must be supplied as input to unwrap the private key so it can be used in cryptographic operations.

seed - Seed to be used for generating a random number.

salt - A random sequence of bytes to be used with the passphrase.

public SecurityContext (CSPRegistry csp, int contextType, Key key, int algorithmID) throws CSSMException

This constructor creates a security context with specified CSP, context type, algorithm ID, and key for computing a Message Authentication Code (MAC).

Parameters:

csp - The CSP that will perform the subsequent cryptographic operations.

contextType - The context type. The value must be CSSM_ALGCLASS_MAC. **This input parameter is required**. Throws CSSMException if this is NULL or any other value.

key - The key to be used in computing the MAC.

algorithmID - The algorithm ID. The possible values are listed in Table 2. **This input parameter is required.** Throws CSSMException if this is NULL.

public SecurityContext (CSPRegistry csp, int contextType, int algorithmID, Key key, String passPhrase) throws CSSMException

This constructor creates a security context with specified CSP, context type, algorithm ID, and key for calculating a digital signature.

Parameters:

csp - The CSP that will perform the subsequent cryptographic operations.

contextType - The context type. The value must be CSSM_ALGCLASS_SIGNATURE. **This input parameter is required**. Throws CSSMException if this is NULL or any other value.

algorithmID - The algorithm ID. The possible values are listed in Table 2. **This input parameter is required.** Throws CSSMException if this is NULL.

Key - The key to be used in calculating the signature.

passPhrase - String containing pass word to wrap the private key.

public SecurityContext (CSPRegistry csp, int contextType, Key key, byte[ ] input) throws CSSMException

This constructor creates a custom security context with specified CSP, context type and input data.

Parameters:

csp - The CSP that will perform the subsequent cryptographic operations.

ContextType - Specifies CSSM_ALGCLASS_CUSTOM. **This input parameter is required**. Throws CSSMException if this is NULL or any other value.

key - The key that will be used to perform the subsequent cryptographic operations.

Input - Custom input data.

protected SecurityContext ( )

This constructor creates an empty security context.

Variables:

private int contextHandle -The handle to the CSSM-cached data structure corresponding to this security context object.

Methods:

protected int getHandle ( )

Gets the handle to the CSSM-cached data structure that corresponds to this security context object.

Returns:

The handle to the CSSM-cached data structure that corresponds to this security context object.

public int getContextType ( )

Gets the context type of this context.

Returns:

The context type. The possible return values are defined in Table 1.

public int getAlgorithmType ( )

Gets algorithm type of this context.

Returns:

The algorithm ID. The possible return values are defined in Table 2.

public int getAlgorithmMode ( )

Gets algorithm mode of this context.

Returns:

The algorithm mode ID. Return NULL if no algorithm mode was found. The possible values returned as defined in Table 4.

public byte[ ] getContextAttributeByte (int attributeType) throws CSSMException

Gets attribute value associated with the specified attribute type. This method should be used to retrieve context attribute information that is stored as a string.

Parameters:

attributeType - The attribute type whose value to be retrieved. This parameter is required. The possible values are listed in Table 3. Throws CSSMException if it is NULL.

Returns:

The attribute value.

Throws: CSSMException

- Invalid context
- Invalid attribute type

public int getContextAttributeValue (int attributeType) throws CSSMException

Gets attribute value associated with the specified attribute type. This method should be used to retrieve context attribute information that is stored as a number.

Parameters:

attributeType - The attribute type whose value to be retrieved. This parameter is required. The possible values are listed in Table 3. Throws CSSMException if it is NULL.

Returns:

The attribute value.

Throws: CSSMException

- Invalid context
- Invalid attribute type

public void free ( )

Frees the context object, invalidating the context handle. An application must invoke this method when the context object is no longer needed. For each instance of class SecurityContext, CSSM caches a corresponding data structure. Invoking this method allows CSSM to delete the corresponding data structure and invalidate the context handle.

protected void finalize ( ) throws Throwable

Frees the resource allocated by CSSM, if it has not yet been freed.

Constants:

Table 1. Context types defining cryptographic services.

Value	Description
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
CSSM_ALGCLASS_MAC	Message Authentication Code Algorithms
CSSM_ALGCLASS_ASYMMETRIC	Asymmetric Encryption Algorithms
CSSM_ALGCLASS_KEYGEN	Key Generation Algorithms
CSSM_ALGCLASS_CUSTOM	Custom Algorithms

Table 2. Algorithms for a security context.

Value	Description
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	MD2 Hash Algorithm
CSSM_ALGID_MD4	MD4 Hash Algorithm
CSSM_ ALGID_MD5	MD5 Hash 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)
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_1KEY	Triple-DES block cipher (with 1 key)
CSSM_ALGID_IDEA	IDEA block cipher
CSSM_ALGID_RC2	RC2 block cipher
CSSM_ALGID_RC5	RC5 block cipher
CSSM_ ALGID_RC4	RC4 stream cipher
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-64 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_MD2Random	MD2-based random numbers
CSSM_ALGID_MD5Random	MD5-based random numbers
CSSM_ALGID_ELGAMAL	Signaturing Algorithm
CSSM_ALGID_SHARandom	SHA-based random numbers
CSSM_ALGID_DESRandom	DES-based random numbers
CSSM_ALGID_CUSTOM	Custom Algorithm

Table 3. Attribute types.

Value	Description
CSSM_ATTRIBUTE_DESCRIPTION	PKCS1 description of attribute
CSSM_ATTRIBUTE_KEY	Indicates key information is in the attribute
CSSM_ATTRIBUTE_INITIAL_VECTOR	Indicates initialization vector in the attribute
CSSM_ATTRIBUTE_SALT	Indicates salt is in the attribute
CSSM_ATTRIBUTE_PADDING	Indicates padding information in attribute
CSSM_ATTRIBUTE_RANDOM	Indicates random data in the attribute
CSSM_ATTRIBUTE_SEED	Indicates a seed is in the attribute
CSSM_ATTRIBUTE_PASSPHRASE	Indicates a passphrase is in the attribute
CSSM_ATTRIBUTE_INPUT_SIZE	Indicates input buffer size is in the attribute
CSSM_ATTRIBUTE_OUTPUT_SIZE	Indicates output buffer size in the attribute
CSSM_ATTRIBUTE_ROUNDS	Indicates the number of runs in the attribute
CSSM_ATTRIBUTE_KEY_LENGTH	Indicates the key size (in bits) is in the attribute
CSSM_ATTRIBUTE_MODULUS_LEN	Indicates the modulus size is in the attribute
CSSM_ATTRIBUTE_CUSTOM	Custom data is in the attribute

Table 4. Modes of cryptographic algorithms.

Value	Description
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
CSSM_ALGMODE_CFB_IV8	CFB with Initialization Vector of 8 bytes
CSSM_ALGMODE_OFB	Output FeedBack
CSSM_ALGMODE_OFB_IV8	OFB with Initialization Vector of 8 bytes
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_CUSTOM	Custom mode

4.10 TPRegistry

Every Trust Policy module (TP) must be installed and registered with the CSSM Trust Policy Services Manager. The Trust Policy Services Manager maintains this registration information in the registration database. Applications may query the Services Manager to retrieve properties of the TP module as defined during installation. Users must use OS tools, provided by TP providers, to do the installation and registration.

The TPRegistry class provides methods to query TP registry information. These methods allow applications to:

- Get a list of TPs registered with CSSM
- Get version information for a selected TP
- Attach and detach a particular TP

Note that this class does not provide functions to install and register a TP with CSSM. Installation and registration of TPs must be performed via CSSM's C language API.

public final class TPRegistry
extends Object

Constructors:

public TPRegistry (byte[ ] tpGUID, String tpName) throws CSSMException

Constructs a TPRegistry object with the specified TP's global unique ID (GUID) and logical name. Saves the TP's GUID and name in the object. Sets TPHandle to NULL.

Parameters:

tpGUID- The global unique ID of the TP.

tpName - The logical name of the TP.

Variables:

private String TPName - TP's logical name.

private byte[ ] guid - TP's GUID.

private int TPHandle - The handle to the CSSM-cached data structure that corresponds to this TP object.

Methods:

protected int getHandle ( )

Gets the handle to the CSSM-cached data structure that corresponds to this TP object. The handle may be obtained only after it is created using the attach method.

Returns:

The TP handle.

public static TPRegistry[ ] getList ( )

Gets a list of TPs that have registered with CSSM TPs must be registered and installed using OS tools on the platform or by installation utilities implemented in the C language. The TP List cannot be set from a Java application/applet.

Returns:
A list of TP objects. Return NULL if no TP is found.

public int getMajorVersion ( )

Gets the major version number of this TP object.

Returns:
Major version number.

public int getMinorVersion ( )

Gets the minor version number of this TP object.

Returns:
Minor version number.

public String getName ( )

Returns:
A string contains the name of this TP.

public byte[ ] getGUID ( )

Returns:
A string contains the GUID of this TP.

public void attach (int majorVersion, int minorVersion) throws CSSMException

Loads the specified TP for execution and creates a handle associated with the TP. If the available version of the TP module is compatible with the version level specified, saves the handle in TPHandle variable.

Returns:
majorVersion - Input parameter of the major version number of the TP module that application is compatible with.

minorVersion - Input parameter of the minor version number of the TP module that application is compatible with.

Throws: CSSMException
- Invalid TP

- Incompatible version
- Unable to attach to TP module

public boolean detach ( )

Deletes the TP object invalidating the TP handle, and unloads the TP-executable module. An application must invoke this method when the TP object is no longer needed. For each instance of class TP, CSSM caches a corresponding data structure. Invoking this method allows CSSM to delete the corresponding data structure and invalidate the TP handle.

Returns:

True - Successfully detached.

False - Failed to detach.

public byte[ ] passThrough (CLRegistry cl, DLRegistry dl, DataStore ds, SecurityContext context, int passThroughId, byte[ ][ ] inputParams) throws CSSMException

Accepts as input an operation ID and a set of arbitrary input parameters. The operation ID may specify any type of operation the TP wishes to export for use by an application.

Parameters:

cl - Handle of the certificate library.

dl - Handle of the data storage library. This parameter is optional and should be specified only if the trust policy library requires it.

ds - Handle of the particular data store to be used to perform this operation. This parameter is optional and should be specified only if the trust policy library requires it and a dl handle is also specified. If a data store handle is specified the data store should be open and ready for processing.

context - The security context. This parameter is optional. It is used to pass a security context (standard or custom) to the TP for use in cryptographic operations.

passThroughId - The operation ID.

InputParams - The input parameters.

Returns:

Result data from the pass through operation.

Throws: CSSMException

- Invalid TP
- Invalid CL
- Invalid DL
- Invalid DataStore
- Invalid security context
- Invalid pass through ID
- Memory error
- Unable to perform pass through

protected void finalize ( ) throws Throwable

Free the resource allocated by CSSM, if it not yet freed.

4.11 TrustPolicy

The primary purpose of a Trust Policy (TP) module is to answer the question*, Is this certificate authorized for this action?* Different trust policies define different actions that may be requested by an application. There are also a few basic actions that should be common to every trust policy. These actions are operations on the basic objects used by all trust models. The basic objects common to all trust models are certificates and certificate revocation lists. The basic operations on these objects are sign, verify, and revoke. A trust policy module may also choose to implement additional API calls. Applications gain access to those functions using the provided passThrough method in TPRegistry class.

This class provides methods to access the basic actions provided by TPs.

public final class TrustPolicy
extends Object

Constructors:

public TrustPolicy (TPRegistry tp)

Parameters:

tp - The Trust policy module that will determine the trust in the certificate.

Variables:

private int TPHandle - The handle of the TP.

Methods:

protected int getTPHandle ( )

Gets the handle to the handle of the TP.

Returns:

The TP handle.

Certificate trustedSignCert (SecurityContext context, CLRegistry cl, DLRegistry dl, DataStore ds, Certificate subjectCert, Certificate signerCert , byte[ ][ ] scopeByteOIDs, byte[ ][ ] scopeByteValues, byte[ ][ ] scopeNumberOIDs, int [ ] scopeNumberValues, byte[ ] scopeKeyOID, Key scopeKeyValue) throw CSSMException

Checks to see if the signer's certificate is trusted to perform the signing operation. If so, signs the certificate using the private key associated with the public key found in the signer's certificate.

Parameters:

context - The security context that describes the environment and parameters under which the cryptographic signing operations should be performed.

cl - Handle of the certificate library that will determine trust in the signer's certificate. This parameter is required.

dl - Handle of the data storage library. This parameter is optional and should be specified only if the trust policy library requires it.

ds - Handle of the particular data store that will contain the signed certificate. This parameter is optional and should be specified only if the trust policy library requires it. A DL handle is also specified. If a data store handle is specified, the data store should be open and ready for processing.

SubjectCert - Subject's certificate.

signerCert - Signer's certificate.

scopeByteOIDs - An array of field IDs that identify which byte array fields of the certificate should be signed. This parameter is optional.

scopeByteValues - An array of byte array field values corresponding to the scopeByteOIDs.

scopeNumberOIDs - An array of field IDs that identify which int fields of the certificate should be signed. This parameter is optional.

scopeNumberValues - An array of int field values corresponding to the scopeNumberOIDs.

scopeKeyOID - Field ID that identify key field of the certificate should be signed. This parameter is optional.

scopeKeyValue - Key value corresponding to the scopeKeyOID.

Returns:

A certificate containing the signer's signature.

Throws: CSSMException

- Invalid security context
- Invalid TP
- Invalid CL
- Invalid DataStore
- Invalid DL
- Unable to sign the certificate
- Unable to allocate memory
- Signer certificate can't sign subject
- Invalid subject certificate

public boolean trustedVerifyCert (SecurityContext context, CLRegistry cl, DLRegistry dl, DataStore ds, Certificate subjectCert, Certificate signerCert, byte[ ][ ] scopeByteOIDs, byte[ ][ ] scopeByteValues, byte[ ][ ] scopeNumberOIDs, int [ ] scopeNumberValues, byte[ ] scopeKeyOID, Key scopeKeyValue) throws CSSMException

Checks to see if the signer's certificate is trusted to perform the verifying operation. If so, verifies the certificate using the public key contained in the signer's certificate. This checks for certificate tampering.

Parameters:

context - The security context that describes the environment and parameters under which the cryptographic verification operation should be performed.

cl - Handle of the certificate library that will determine trust in the signer's certificate. This parameter is required.

dl - Handle of the data storage library. This parameter is optional and should be specified only if the trust policy library requires it.

ds - Handle of the particular data store that will contain the verified certificate. This parameter is optional and should be specified only if the trust policy library requires it. A DL handle is also specified. If a data store handle is specified the data store should be open and ready for processing.

subjectCert - The subject's certificate.

signerCert - The signer's certificate.

scopeByteOIDs - An array of field IDs that identify which byte array fields of the certificate should be verified. This parameter is optional.

scopeByteValues - An array of byte array field values corresponding to the scopeByteOIDs.

scopeNumberOIDs - An array of field IDs that identify which int fields of the certificate should be verified. This parameter is optional.

scopeNumberValues - An array of int field values corresponding to the scopeNumberOIDs.

scopeKeyOID - Field ID that identify key field of the certificate should be verified. This parameter is optional.

scopeKeyValue - Key value corresponding to the scopeKeyOID.

Returns:

True - Verify successfully.

False - The signature associated with the certificate is not verified.

Throws: CSSMException

- The certificate is not trusted for the verification operation
- Cannot retrieve the public key from the signer's certificate
- Invalid security context
- Invalid TP
- Invalid CL
- Invalid DataStore
- Invalid DL
- Invalid certificate
- Signature can't be trusted
- Unable to verify certificate

public CRL trustedRevokeCert (SecurityContext context, CLRegistry cl, DLRegistry dl, DataStore ds, CRL oldCrl, Certificate subjectCert, Certificate revokerCert, int reason) throws CSSMException

Checks to see if the revoker's certificate is trusted to perform/sign the revocation and if so, to carry out the operation by adding a new revocation record to the CRL.

Parameters:

context - The security context that describes the environment and parameters under which the cryptographic verification operation should be performed.

cl - Handle of the certificate library that will determine trust in the signer's certificate. This parameter is required.

dl - Handle of the data storage library. This parameter is optional and should be specified only if the trust policy library requires it.

ds - Handle of the particular data store that should/will contain the revoked CRL. This parameter is optional and should be specified only if the trust policy library requires it and a dl handle is also specified. If a data store handle is specified the data store should be open and ready for processing.

oldCRL - The certificate revocation list.

subjectCert - The subject's certificate.

revokerCert - The revoker's certificate.

reason - The reason for revoking the certificate. The possible values are listed in Table 5.

Returns:

The new CRL containing the old CRL and a new record corresponding to the newly-revoked certificate.

Throws: CSSMException

- Invalid security context
- Invalid certificate
- Revoker certificate can't revoke subject
- Invalid TP
- Invalid CL
- Invalid DL
- Invalid DataStore
- No CRL was specified
- Unable to revoke certificate
- Error in allocating memory

<u>public void trustedAction (SecurityContext context, CLRegistry cl, DLRegistry dl, DataStore ds, int action, Certificate cert, byte[ ] data) throws CSSMException</u>

Performs a domain-specific action. The trust policy library must determine whether or not the certificate is trusted to perform the domain-specific action.

Parameters:

context - The security context that describes the environment and parameters under which the cryptographic operation should be performed.

cl - Handle of the certificate library that will determine trust in the signer's certificate. This parameter is required.

dl - Handle of the data storage library. This parameter is optional and should be specified only if the trust policy library requires it.

ds - Handle of the particular data store to be used to perform this operation. This parameter is optional and should be specified only if the trust policy library

requires it and a DL handle is also specified. If a data store handle is specified the data store should be open and ready for processing.

action - The domain-specific action.

cert - The certificate to perform the domain-specific action.

data - A byte array that the action is going to be performed on.

Throws: CSSMException

- Invalid TP
- Invalid CL
- Invalid DL
- Invalid DataStore
- Invalid security context
- Invalid certificate
- Invalid action
- Certificate not trusted for action
- Unable to determine trust for action

public CRL trustedSignCrl (SecurityContext context, CLRegistry cl, DLRegistry dl, DataStore ds, CRL crl, Certificate signerCert, byte[ ][ ], scopeByteOIDs, byte[ ][ ] scopeByteValues, byte[ ][ ] scopeNumberOIDs, int [ ] scopeNumberValues)
throws CSSMException

Checks to see if the signer's certificate is trusted to perform the signing operation. If so, signs the CRL using the private key associated with the public key found in the signer's certificate.

Note: This method verifies trust in the signer to sign the entire CRL, not just a single record in the CRL.

Parameters:

context - The security context that describes the environment and parameters under which the cryptographic signing operation should be performed.

cl - Handle of the certificate library that will determine trust in the signer's certificate. This parameter is required.

dl - Handle of the data storage library. This parameter is optional and should be specified only if the trust policy library requires it.

ds - Handle of the particular data store to be used to perform this operation. This parameter is optional and should be specified only if the trust policy library requires it and a DL handle is also specified. If a data store handle is specified the data store should be open and ready for processing.

crl - The certificate revocation list to be signed.

signerCert - Signer's certificate.

scopeByteOIDs - An array of field IDs that identify which byte array fields of the CRL should be signed. This parameter is optional.

scopeByteValues - An array of byte array field values corresponding to the scopeByteOIDs.

scopeNumberOIDs - An array of field IDs that identify which int fields of the CRL should be signed. This parameter is optional.

scopeNumberValues - An array of int field values corresponding to the scopeNumberOIDs.

Returns:

A CRL containing the signer's signature.

Throws: CSSMException

- Invalid security context
- Invalid TP
- Invalid CL
- Invalid DL
- Invalid DataStore
- Invalid certificate
- Signer certificate can't sign CRL
- Signing scope is invalid
- Not enough memory
- Unable to sign CRL

public boolean trustedVerifyCrl (SecurityContext context, CLRegistry cl, DLRegistry dl, DataStore ds, CRL crl, Certificate signerCert, scopeByteOIDs, byte[ ][ ] scopeByteValues, byte[ ][ ] scopeNumberOIDs, int [ ] scopeNumberValues) throws CSSMException

Verifies trust in the signature that applies to the entire CRL. Verification uses the public key specified in the CA's certificate. It does not verify trust in individual signatures associated with each record in the CRL. This method should be used to detect tampering of the entire CRL, and to authenticate the CA that constructed the CRL.

Parameters:

context - The security context that describes the environment and parameters under which the cryptographic verification operation should be performed.

cl - Handle of the certificate library that will determine trust in the signer's certificate. This parameter is required.

dl - Handle of the data storage library. This parameter is optional and should be specified only if the trust policy library requires it.

ds - Handle of the particular data store that should/will contain the verified certificate. This parameter is optional and should be specified only if the trust policy library requires it and a DL handle is also specified. If a data store handle is specified the data store should be open and ready for processing.

crl - The certificate revocation list to be verified.

signerCert - The signer's certificate.

scopeByteOIDs - An array of field IDs that identify which byte array fields of the CRL should be verified. This parameter is optional.

scopeByteValues - An array of byte array field values corresponding to the scopeByteOIDs.

scopeNumberOIDs - An array of field IDs that identify which int fields of the CRL should be verified. This parameter is optional.

scopeNumberValues - An array of int field values corresponding to the scopeNumberOIDs

Returns:

True - Verify successfully.

False - The signature associated with the certificate is not verified.

Throws: CSSMException

- Invalid certificate
- Signer certificate is not signer of CRL
- CRL can't be trusted
- Verify scope is invalid
- Unable to verify CRL
- Invalid TP
- Invalid CL
- Invalid DL
- Invalid DataStore

public void trustedToDS (CLRegistry cl, DLRegistry dl, DataStore ds, CRL crl) throws CSSMException

Verifies trust in the signature that applies to the entire CRL, and approves it for persistence in the specified data store.

Parameters:

cl - Handle of the certificate library that will determine trust in the signer's certificate. This parameter is required.

dl - Handle of the data storage library. This parameter is optional and should be specified only if the trust policy library requires it.

ds - Handle of the particular data store that this CRL will be stored in. This parameter is optional and should be specified only if the trust policy library requires it and a DL handle is also specified. If a data store handle is specified the data store should be open and ready for processing.

crl - The certificate revocation list.

Throws: CSSMException

- Invalid TP
- Invalid DL
- Invalid DataStore
- Invalid CL
- Invalid CRL
- CRL cannot be trusted
- Unable to apply CRL to data store

Constants:

Table 5. Reason for revoking a certificate. (X.509 amendment 1.)

Value	Description

CSSM_REVOKE_CUSTOM	Custom reason
CSSM_REVOKE_UNSPECIFIC	Reason unspecified
CSSM_REVOKE_KEYCOMPROMISE	Private key of certificate has been compromised
CSSM_REVOKE_CACOMPROMISE	CA has been compromised
CSSM_REVOKE_SUPERCEDED	Overwrite the revocation
CSSM_REVOKE_ AFFILIATIONCHANGE	Affiliation of CA/issuer has been changed
CSSM_REVOKE_ CESSATIONOFOPERATION	CA/Issuer has stopped operation
CSSM_REVOKE _CERTIFICATEHOLD	Certificate is on hold
CSSM_REVOKE _CERTIFICATEHOLDRELEASE	Hold on certificate has been released
CSSM_REVOKE _REMOVEFROMCRL	Remove certificate from CRL

4.12 CSSMException

Signals that a CSSM exception has occurred.

public final class CSSMException
extends Exception

Constructors:

public CSSMException ( )

Constructs a CSSMException with no detail message.

public CSSMException (String msg)

Constructs a CSSMException with the specified detail message. A detail message is a String that describes this particular exception.

Parameters:

msg - The detail message.

public CSSMException (String msg, int code)

Constructs a CSSMException with the specified detail message and error code.

Parameters:

msg - The detail message.

code - The error code returned by CSSM.

Variables:

private int errorCode - The error code returned by CSSM.

Methods:

public int getErrorCode ( )

Gets the error code.

Returns:

The error code.