Common Security Services Manager

Trust Policy Interface (TPI) Specification

Draft for Release 1.2
March 1997

Subject to Change Without Notice

Specification Disclaimer and Limited Use License

This specification is for release version 1.2, March 1997.

You are licensed under Intel's copyrights in the CDSA Specifications to download the specifications and to develop, distribute and/or use a conferment software implementation of the specifications. A software implementation of the CDSA Specifications can be tested for conformance via use of the CDSA Conformance Test Suite that accompanies the specifications, and you are licensed to use the conformance test suite for that purpose.

ALL INFORMATION AND OTHER MATERIALS TO BE PROVIDED BY INTEL HEREUNDER ARE PROVIDED "AS IS," AND INTEL MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AND EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS, AND FITNESS FOR A PARTICULAR PURPOSE.

Intel grants no other license under any of it's intellectual property other than as expressly granted above. If you desire any broader rights under Intel intellectual property, please contact Intel directly.

.

Intel Corporation, 5200 N.E. Elam Young Parkway, Hillsboro, OR 97124-6497

Table of Contents

List of Figures

1. Introduction

1.1 CDSA Overview

The Common Data Security Architecture (CDSA) defines the infrastructure for a comprehensive set of security services. CDSA is an extensible architecture that provides mechanisms to manage add-in security modules. These modules use cryptography as a computational base to build security protocols and security systems. Figure 1 shows the four basic layers of the Common Data Security Architecture: Applications, System Security Services, the Common Security Services Manager, and Security Add-in Modules. The Common Security Services Manager (CSSM) is the core of CDSA. It provides a way for applications to access security services directly through the CSSM security API, or to access security services indirectly via layered security services and tools implemented over the CSSM API. CSSM manages the add-in security modules and directs application calls through the CSSM API to the selected add-in module servicing the request. Add-in modules perform various methods of security services, including:

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

Cryptographic Service Providers (CSPs) are add-in modules that 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) or MasterCard* (as an institution). Each Trust Policy module embodies the semantics of a trust model based on using digital certificates as credentials. Applications may 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.

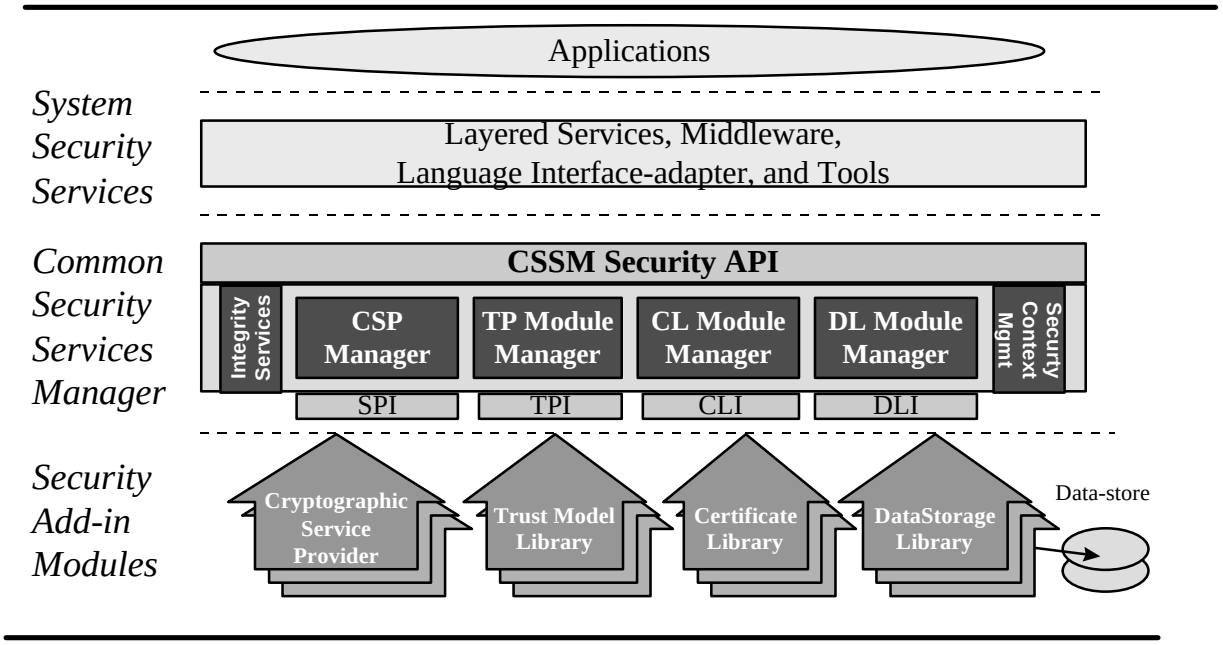

Figure 1. **The Common Data Security Architecture for all platforms.**

Applications directly or indirectly select the modules used to provide security services to the application. Independent software and hardware vendors provide these add-in modules. The functionality of the add-in module may be extended beyond the services defined by the CSSM API, by exporting additional services to applications via the CSSM PassThrough mechanism.

The API calls defined for add-in modules are categorized as service operations, module management operations, and module-specific operations. Service operations include functions that perform a security operation such as encrypting data, inserting a certificate revocation list into a data source, or verifying that a certificate is trusted. Module management functions support module installation, registration of module features and attributes, and queries to retrieve information on module availability and features. Module-specific operations are enabled in the API through pass-through functions whose behavior and use is defined by the add-in module developer.

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.

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 its digital certificate.

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 runtime context of a user application
- call-back functions for disk, screen, and keyboard I/O supported by the operating system
- test-and-check function to ensure CSSM integrity
- management of concurrent security operations

1.2 Trust Policy Overview

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

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

The CSSM Trust Policy API defines the generic operations that each TP module supports. Each module may choose to implement the required subset of these operations for the policy it serves.

The CSSM API defines a pass-through function, which allows each module to provide additional functions, along with those defined by the CSSM Trust Policy API. When a TP function determines the trustworthiness of performing an action, it may invoke Certificate Library functions and Data storage Library functions to carry out the mechanics of the approved action. TP modules must be installed and registered with the CSSM Trust Policy Services Manager. Applications may query the Services Manager to retrieve properties of the TP module, as defined during installation.

1.2.1 Application Interaction

An application determines the availability of a Trust Policy module by querying the CSSM Registry. When a new TP is installed on a system, it must be registered with CSSM. When a client requests that CSSM attach to a TP, CSSM returns a TP handle to the application which uniquely identifies the pairing of the application thread to the TP module instance. The application uses this handle to identify the TP in future function calls.

CSSM uses the TP's function table to pass function calls from an application to a Trust Policy module. The function table consists of pointers to the subset of trust functions from the CSSM API, which are supported by the TP. When an application causes CSSM to attach the TP, the Trust Policy registers its function table with CSSM using *CSSM_TP_RegisterServices*. During future function calls from the application, CSSM uses these function pointers to direct the call appropriately.

The calling application is the responsible for the allocation and de-allocation of all memory that it passes into or out of the Trust Policy module. The application must register memory allocation and de-allocation upcalls with CSSM when it requests a TP attach. These upcalls and the handle identifying the

application/TP pairing are passed to the TP when CSSM calls its *TP_Initialize* function. These functions must be used whenever a Trust Policy allocates or de-allocates memory that belongs to or will belong to the application.

1.3 CSSM Trust Policy Interface Specification

1.3.1 Intended Audience

This document should be used by Independent Software Vendors (ISVs) or certificate authorities (CAs) who want to develop their own Trust Policy module. These developers should be highly experienced software architects, advanced programmers, or sophisticated users, who are experts in the security and authorization policies of their application area. They also should be familiar with high-end cryptography and digital certificates. We assume that this audience is familiar with the basic capabilities and features of the protocols they are considering.

1.3.2 Document Organization

This document is divided into the following sections:
Section 2, Trust Policy Interface, describes the functions that a Trust Policy module makes available to applications via the CSSM.

Section 3, Trust Policy Structure and Management, describes important considerations in developing a Trust Policy module. It also gives examples of how several trust policy functions might be implemented.

1.4 References

PKCS*	*The Public-Key Cryptography Standards*, RSA Laboratories, Redwood City, CA: RSA Data Security, Inc.
X.509	*CCITT. Recommendation X.509: The Directory – Authentication Framework*, 1988. CCITT stands for Comite Consultatif Internationale Telegraphique et Telphonique (International Telegraph and Telephone Consultative Committee)
SPKI	*Simple Public Key Infrastructure*
SDSI	*SDSI - A Simple Distributed Security Infrastructure*, R. Rivest and B. Lampson, 1996
CDSA	*Common Data Security Architecture Specification*, Intel Architecture Labs, 1996
CSSM API	*CSSM Application Programming Interface*, Intel Architecture Labs, 1996
CSSM SPI	*CSSM Cryptographic Service Provider Interface Specification*, Intel Architecture Labs, 1996
CSSM CLI	*CSSM Certificate Library Interface Specification*, Intel Architecture Labs, 1996
CSSM DLI	*CSSM Data storage Library Interface Specification*, Intel Architecture Labs, 1996
CSSM Java*	*CSSM Java Application Programming Interface Specification*, Intel Architecture Labs, 1996

2. Trust Policy Interface

2.1 Overview

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

The use of digital certificates is the foundation on which the CDSA is designed. The CDSA assumes the concept of digital certificates in its broadest sense. Applications use the credential for:

- identification
- authentication
- authorization

The applications interpret and manipulate the contents of certificates to achieve these ends, based on the real-world trust model they chose as their model for trust and security. The primary purpose of a Trust Policy (TP) module is to answer the question, "Is this certificate trusted for this action?" The CSSM Trust Policy API determines the generic operations that should be defined for certificate-based trust in every application domain. The specific semantics of each operation is defined by the :

- application domain
- trust model
- policy statement for a domain
- certificate type
- real-world operation the user is trying to perform within the application domain

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

Different trust policies define different actions that an application may request. Some of these actions are common to every trust policy, and are operations on objects all trust models use. The objects common to all trust models are certificates and certificate revocation lists. The basic operations on these objects are sign, verify, and revoke.

Based on this analysis, CSSM defines two categories of API calls that should be implemented by TP modules. The first category allows the TP module to define and expose actions specific to the trust domain (such as requesting authorization to make a $200 charge on a credit card certificate, and requesting access to the locked project lab). The second category specifies basic operations (for example, sign, verify, and revoke) on certificates and certificate revocation lists.

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

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

Implementing a Trust Policy module may or may not be tightly coupled with one or more Certificate Library modules or one or more Data Storage Library modules. The trust policy embodies the semantics of the domain. The certificate library and the data storage library embody the syntax of a certificate format and operations on that format. A trust policy can be completely independent of certificate format, or it may be defined to operate with one or a small number of certificate formats. A trust policy implementation may invoke a certificate library module and/or a data storage library module to manipulate certificates.

The Trust Policy API defines two categories of operation:

- module installation and management
- trust-based services

2.1.1 Trust Policy Registration and Management

The Trust Policy Module Manager defines API calls for installing and registering TP modules. CSSM manages a trust policy registry that records each trust policy's logical name and the information required to locate and dynamically initiate the module. An application uses an *attach* operation to load and initiate a module. The module executable may be local or remote.

When a policy module is loaded, it must register its services with the CSSM before an application can use it. A TP module registers a set of callback functions with the CSSM. There is one callback function for each CSSM-defined trust policy API call. The module may or may not implement all trust policy calls defined by CSSM. Non-implemented functions are registered as NULL.

The Trust Policy module may implement additional functions outside of the CSSM-defined API calls. This set of extended functions is available through a single callback function, which the module registers with the CSSM trust policy services manager. Applications access these functions through the CSSM *pass-through* function. The Trust Policy module must document the features and services these functions provided. CSSM does not require or enforce the availability of run-time query support for extended functions.

The Trust Policy Module Manager API allows an application to query the registry of installed Trust Policy modules to determine their availability. Trust Policy modules may be detached but the application should not invoke this operation unless all requests to the target module have been completed. Trust Policy modules may also be de-installed. This operation removes the trust policy's logical name and its associated attributes from the CSSM's registry. De-install must be performed before a new version of a Trust Policy module is installed in the CSSM registry.

2.1.2 Trust Policy Services API

CSSM defines nine API calls that all trust policies should at a minimum, implement. Six functions define operations on the fundamental CSSM object types of certificate and certificate revocation lists. Two functions are used by the TP module to extend the semantics of its policy to the application. The remaining function is used for version checking.

Signing Certificates and Certificate Revocation Lists Every system should be capable of being a Certificate Authority (CA), if so authorized. CAs are applications that issue and validate certificates and certificate revocation lists (CRLs). Issuing certificates and CRLs include initializing their attributes and digitally signing the result using the private key of the issuing authority. The private key used for signing is associated with the signer's certificate. The Trust Policy module must evaluate the trustworthiness of the signer's certificate before performing this operation. Some policies may require that multiple authorities sign a newly-issued certificate. If the TP trusts the signer's certificate, then the TP module may perform the cryptographic signaturing algorithm by invoking the signing function in a Certificate library module, or by directly invoking the data signing function in a CSP module. The certificate library functions that can be used to carry out some of the TP operations are documented in *CSSM Certificate Library Interface Specification*

Verifying Certificates and Certificate Revocation Lists The TP module determines the general trustworthiness of a certificate. This is a general verification of trust in a certificate. The TP modules must also determine the trustworthiness of a certificate revocation list received from a remote system. The test focuses on the trustworthiness of the agent who signed the CRL. The TP module may need to perform operations on the certificate or CRL to determine trustworthiness. If these operations depend on the data format of the certificate or CRL, the TP module uses the services of a certificate library module to perform these checks.

Revoking Certificates When revoking a certificate, the identity of the revoking agent is presented in the form of another certificate. The TP module must determine trustworthiness of the revoking agent's certificate to perform revocation. If the requesting agent's certificate is trustworthy, the TP module carries out the operation directly by invoking a certificate library module to add a new revocation record to a CRL, marking the certificate as revoked. The CSSM API also defines a reason parameter that is passed to the TP module. The TP may use this parameter as part of its trust evaluation.

Verify Action The TP module must determine if the certificate presented is trusted to perform the domain-specific action defined by the TP module. An action for a TP module might be an employee's access to a lab housing a critical project. The question of whether to allow the employee into the lab is asked through this function.

Pass-through Function For operations not defined in the TPI, the pass-through function allows the TP module to provide support for these services to clients. These private services are identified by operation identifiers. TP module developers must provide documentation of these services.

Version Checking As part of the attach process, version information is supplied by the client. The TP must determine if it is compatible to the version the client had requested. If the TP decides that it is incompatible, the CSSM will not complete the attaching of the TP module to the client.

2.1.3 Trust Operations

CSSM_BOOL CSSMTPI TP_CertVerify ( ) - Determines whether the certificate is trustworthy.

CSSM_DATA_PTR CSSMTPI TP_CertSign ( ) Determines whether the signer's certificate is authorized to perform the signing operation. If so, The TP module carries out the operation. The*scope* of a signature may be used to identify which certificate field should be signed. An example is the case of multiple signatures on a certificate. Should signatures be applied to just the certificate, or to the certificate and all currently-existing signatures, as a notary public would do.

CSSM_ DATA_PTR CSSMTPI TP_CertRevoke ( ) Determines whether the revoker's certificate is trusted to perform/sign the revocation. If so, the TP module carries out the operation by adding a new revocation record to the CRL.

CSSM_BOOL CSSMTPI TP_CrlVerify ( ) Determines whether the CRL is trusted. This test may include verifying the correctness of the signature associated with the CRL, determining whether the CRL has been tampered with, and determining if the agent who signed the CRL was trusted to do so.

CSSM_ DATA_PTR CSSMTPI TP_CrlSign ( ) Determines whether the certificate is trusted to sign the CRL. If so, the TP module carries out the operation.

CSSM_RETURN CSSMTPI TP_ApplyCrlToDb ( ) Determines whether the memory-resident CRL is trusted and should be applied to a persistent database, which could result in designating certificates as revoked.

2.1.4 Extensibility Functions

CSSM_BOOL CSSMTPI TP_VerifyAction ( ) Determines whether or not the certificate is trusted to perform a domain-specific action. Certificates can be used to request authorizations in an application domain.

CSSM_RETURN CSSMTPI TP_PassThrough ( ) Executes TP module custom operations. This function accepts as input an operation ID and an arbitrary set of input parameters. The operation ID may specify any type of operation the TP wishes to export. Such operations may include queries or services specific to the domain represented by the TP module.

2.1.5 Module Management Functions

CSSM_RETURN CSSMTPI TP_Initialize ( ) This function checks whether the version of the attached TP module is compatible with the input version number and performs TP module setup activities. It is called by the CSSM Core as part of the *CSSM_TP_Attach*routine. It is called immediately after the TP module's function table is registered with CSSM. If the versions are incompatible, the TP module is detached, a CSSM_INCOMPATIBLE_VERSION error is set, and a NULL handle is returned to the calling application.

CSSM_RETURN CSSMTPI TP_Uninitialize ( ) This function checks performs TP module cleanup activities. It is called by the CSSM Core as part of the *CSSM_TP_Detach*routine. It is called immediately prior to the detach of the TP module.

2.2 Data Structures

```
typedef uint32 CSSM_TP_HANDLE    /* Trust Policy Handle */
typedef uint32 CSSM_TP_ACTION
tydefef CSSMAPI CSSMTPI
```

2.2.1 CSSM_DATA

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

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

Definition:

Length - The length, in bytes, of the memory block pointed to by *Data*.

Data - A pointer to a contiguous block of memory.

2.2.2 CSSM_OID

This structure stores object identifier for describing the data.

```
typedef CSSM_DATA CSSM_OID, *CSSM_OID_PTR;
```

2.2.3 CSSM_FIELD

This structure contains the tag/data pair for a single field of a certificate or CRL.

```
typedef struct cssm_field {
    CSSM_OID FieldOid;
    CSSM_DATA FieldValue;
}CSSM_FIELD, *CSSM_FIELD_PTR
```

Definition:

FieldOid - The object identifier which uniquely identifies this certificate or CRL field.

FieldValue - The data contained in this certificate or CRL field.

2.2.4 CSSM_REVOKE_REASON

This structure represents the reason a certificate is being revoked.

```
typedef enum cssm_revoke_reason {
    CSSM_REVOKE_CUSTOM,
    CSSM_REVOKE_UNSPECIFIC,
    CSSM_REVOKE_KEYCOMPROMISE,
    CSSM_REVOKE_CACOMPROMISE,
    CSSM_REVOKE_AFFILIATIONCHANGED,
    CSSM_REVOKE_SUPERCEDED,
    CSSM_REVOKE_CESSATIONOFOPERATION,
    CSSM_REVOKE_CERTIFICATEHOLD,
    CSSM_REVOKE_CERTIFICATEHOLDRELEASE,
    CSSM_REVOKE_REMOVEFROMCRL
} CSSM_REVOKE_REASON
```

2.3 Trust Policy Operations

2.3.1 TP_CertVerify

CSSM_BOOL CSSMTPI TP_CertVerify (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR SubjectCert,
const CSSM_DATA_PTR SignerCert,
const CSSM_FIELD_PTR VerifyScope,
uint32 ScopeSize)

This function determines whether the certificate is trusted.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DBHandle (input)
The handle that describes the data storage used to perform this function.

CCHandle (input)
The handle that describes the context of the cryptographic operation.

SubjectCert (input)
A pointer to the CSSM_DATA structure containing the subject certificate.

SignerCert (input)
A pointer to the CSSM_DATA structure containing the certificate used to sign the subject certificate.

VerifyScope (input)
A pointer to the CSSM_FIELD array containing the tags of the fields to be verified.
A null input verifies a default set of fields in the certificate.

ScopeSize (input)
The number of entries in the verify scope list.

Return Value

A CSSM_TRUE return value signifies that the certificate can be trusted. When CSSM_FALSE is returned, either the certificate cannot be trusted or an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_TP_INVALID_CERTIFICATE	Invalid certificate
CSSM_TP_NOT_SIGNER	Signer certificate is not signer of subject
CSSM_TP_NOT_TRUSTED	Signature can't be trusted
CSSM_TP_CERT_VERIFY_FAIL	Unable to verify certificate
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

See Also

CSSM_TP_CertSign, CSSM_CL_CertVerify

2.3.2 TP_CertSign

CSSM_DATA_PTR CSSMTPI TP_CertSign (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR SubjectCert,
const CSSM_DATA_PTR SignerCert,
const CSSM_FIELD_PTR SignScope,
uint32 ScopeSize)

The TP module decides first whether the signer certificate is trusted to sign the subject certificate. Once the trust is established, the TP signs the certificate when given the signer's certificate and the *scope* of the signing process.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DBHandle (input)
The handle that describes the data storage used to perform this function.

CCHandle (input)
The handle that describes the context of the cryptographic operation.

SubjectCert (input)
A pointer to the CSSM_DATA structure containing the subject certificate.

SignerCert (input)
A pointer to the CSSM_DATA structure containing the certificate to use to sign the subject certificate.

SignScope (input)
A pointer to the CSSM_FIELD array containing the tags of the fields to be signed. A NULL input signs a default set of fields in the certificate.

ScopeSize (input)
The number of entries in the sign scope list.

Return Value

A pointer to the CSSM_DATA structure containing the signed certificate. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_TP_INVALID_CERTIFICATE	Invalid certificate
CSSM_TP_CERTIFICATE_CANT_OPERATE	Signer certificate can't sign subject
CSSM_TP_MEMORY_ERROR	Error in allocating memory
CSSM_TP_CERT_SIGN_FAIL	Unable to sign certificate
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

See Also

CSSM_TP_CertVerify, CSSM_CL_CertSign

2.3.3 TP_CertRevoke

CSSM_DATA_PTR CSSMTPI TP_CertRevoke (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR OldCrl,
const CSSM_DATA_PTR SubjectCert,
const CSSM_DATA_PTR RevokerCert,
CSSM_REVOKE_REASON Reason)

The TP module determines whether the revoking certificate can revoke the subject certificate. Once the trust is established, the TP revokes the subject certificate by adding it to the certificate revocation list.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DBHandle (input)
The handle that describes the data storage used to perform this function.

CCHandle (input)
The handle that describes the context of the cryptographic operation.

OldCrl (input)
A pointer to the CSSM_DATA structure containing an existing certificate revocation list. If this input is NULL, a new list is created.

SubjectCert (input)
A pointer to the CSSM_DATA structure containing the subject certificate.

RevokerCert (input)
A pointer to the CSSM_DATA structure containing the certificate under whose authority the subject certificate is revoked.

Reason (input)
The reason for revoking the subject certificate.

Return Value

A pointer to the CSSM_DATA structure containing the updated certificate revocation list. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_TP_INVALID_CRL	Invalid CRL
CSSM_TP_INVALID_CERTIFICATE	Invalid certificate
CSSM_TP_CERTIFICATE_CANT_OPERATE	Revoker certificate can't revoke subject
CSSM_TP_MEMORY_ERROR	Error in allocating memory
CSSM_TP_CERT_REVOKE_FAIL	Unable to revoke certificate
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

See Also

CSSM_CL_CrlAddCert

2.3.4 TP_CrlVerify

CSSM_BOOL CSSMTPI TP_CrlVerify (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR SubjectCrl,
const CSSM_DATA_PTR SignerCert,
const CSSM_FIELD_PTR VerifyScope,
uint32 ScopeSize)

The TP module determines whether the certificate revocation list is trusted.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DBHandle (input)
The handle that describes the data storage used to perform this function.

CCHandle (input)
The handle that describes the context of the cryptographic operation.

SubjectCrl (input)
A pointer to the CSSM_DATA structure containing the certificate revocation list.

SignerCert (input)
A pointer to the CSSM_DATA structure containing the certificate used to sign the certificate revocation list.

VerifyScope (input)
A pointer to the CSSM_FIELD array containing the tags of the fields to be verified. A null input verifies a default set of fields in the certificate revocation list.

ScopeSize (input)
The number of entries in the verify scope list.

Return Value

A CSSM_TRUE return value means the certificate revocation list can be trusted. If CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_TP_INVALID_CERTIFICATE	Invalid certificate
CSSM_TP_NOT_SIGNER	Signer certificate is not signer of CRL
CSSM_TP_NOT_TRUSTED	Certificate revocation list can't be trusted
CSSM_TP_CRL_VERIFY_FAIL	Unable to verify certificate
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

See Also

CSSM_CL_CrlVerify

2.3.5 TP_CrlSign

CSSM_DATA_PTR CSSMTPI TP_CrlSign (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
const CSSM_DATA_PTR SubjectCrl,
const CSSM_DATA_PTR SignerCert,
const CSSM_FIELD_PTR SignScope,
uint32 ScopeSize)

The TP module first decides whether the signer certificate is trusted to sign the subject certificate revocation list. Once the trust is established, the TP signs the certificate revocation list.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DBHandle (input)
The handle that describes the data storage used to perform this function.

CCHandle (input)
The handle that describes the context of the cryptographic operation.

SubjectCrl (input)
A pointer to the CSSM_DATA structure containing the certificate revocation list.

SignerCert (input)
A pointer to the CSSM_DATA structure containing the certificate to use to sign the certificate revocation list.

SignScope (input)
A pointer to the CSSM_FIELD array containing the tags of the fields to be signed. A NULL input signs a default set of fields in the certificate revocation list.

ScopeSize (input)
The number of entries in the sign scope list.

Return Value

A pointer to the CSSM_DATA structure containing the signed certificate revocation list. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_TP_INVALID_CERTIFICATE	Invalid certificate
CSSM_TP_CERTIFICATE_CANT_OPERATE	Signer certificate can't sign certificate revocation list
CSSM_TP_MEMORY_ERROR	Error in allocating memory
CSSM_TP_CRL_SIGN_FAIL	Unable to sign certificate revocation list
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

See Also

CSSM_CL_CrlSign

2.3.6 TP_ApplyCrlToDb

CSSM_RETURN CSSMTPI TP_ApplyCrlToDb (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
const CSSM_DATA_PTR Crl)

The TP module first determines whether the memory-resident CRL is trusted, and if it should be applied to a persistent database. Once the trust is established, the TP updates the persistent storage to reflect entries in the certificate revocation list. This results in designating persistent certificates as revoked.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DBHandle (input)
The handle that describes the data storage used to perform this function.

Crl (input)
A pointer to the CSSM_DATA structure containing the certificate revocation list.

Return Value

A CSSM_TRUE return value means the certificate revocation list has been used to update the revocation status of certificates in the specified database. If CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_TP_INVALID_CRL	Invalid certificate revocation list
CSSM_TP_NOT_TRUSTED	Certificate revocation list can't be trusted
CSSM_TP_APPLY_CRL_TO_DB_FAIL	Unable to apply certificate revocation list on database
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

See Also

CSSM_CL_CrlGetFirstItem, CSSM_CL_CrlGetNextItem, CSSM_DL_CertRevoke

2.4 Extensibility Functions

2.4.1 TP_VerifyAction

CSSM_BOOL CSSMTPI TP_VerifyAction (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
CSSM_TP_ACTION Action,
const CSSM_DATA_PTR Data,
const CSSM_DATA_PTR Cert)

The TP module determines whether the given certificate is trusted to perform the module-specific action.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DBHandle (input)
The handle that describes the data storage used to perform this function.

CCHandle (input)
The handle that describes the context of the cryptographic operation.

Action (input)
An action to be performed under the authority of the input certificate.

Data (input)
A pointer to the CSSM_DATA structure containing the module-specific data required to authorize or to perform the requested action.

Cert (input)
A pointer to the CSSM_DATA structure containing the certificate.

Return Value

A CSSM_TRUE return value means the certificate can be trusted. If CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_TP_INVALID_CERTIFICATE	Invalid certificate
CSSM_TP_INVALID_ACTION	Invalid action
CSSM_TP_NOT_TRUSTED	Certificate not trusted for action
CSSM_TP_VERIFY_ACTION_FAIL	Unable to determine trust for action
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

2.4.2 TP_PassThrough

CSSM_DATA_PTR CSSMTPI TP_PassThrough (CSSM_TP_HANDLE TPHandle,
CSSM_CL_HANDLE CLHandle,
CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CC_HANDLE CCHandle,
uint32 PassThroughId,
const CSSM_DATA_PTR InputParams)

The TP module allows clients to call Trust Policy module-specific operations that have been exported. Such operations may include queries or services specific to the domain represented by the TP module.

Parameters

TPHandle (input)
The handle that describes the add-in trust policy module used to perform this function.

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

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DBHandle (input)
The handle that describes the data storage used to perform this function.

CCHandle (input)
The handle that describes the context of the cryptographic operation.

PassThroughId (input)
An identifier assigned by the TP module to indicate the exported function to perform.

InputParams (input)
A pointer to the CSSM_DATA structure containing parameters to be interpreted in a function-specific manner by the TP module.

Return Value

A pointer to the CSSM_DATA structure containing the output from the pass-through function. The output data must be interpreted by the calling application based on externally-available information. If the pointer is NULL, an error has occurred.

Error Codes

Value	Description
CSSM_TP_INVALID_TP_HANDLE	Invalid handle
CSSM_TP_INVALID_CL_HANDLE	Invalid handle
CSSM_TP_INVALID_DL_HANDLE	Invalid handle
CSSM_TP_INVALID_DB_HANDLE	Invalid handle
CSSM_TP_INVALID_CC_HANDLE	Invalid handle
CSSM_TP_INVALID_DATA_POINTER	Invalid pointer for input data
CSSM_TP_INVALID_ID	Invalid pass through ID
CSSM_TP_MEMORY_ERROR	Error in allocating memory
CSSM_TP_PASS_THROUGH_FAIL	Unable to perform pass-through
CSSM_FUNCTION_NOT_IMPLEMENTED	Function not implemented

2.5 Module Management Functions

2.5.1 TP_Initialize

CSSM_RETURN CSSMTPI TP_Initialize (uint32 VerMajor,
uint32 VerMinor)

This function checks whether the current version of the TP module is compatible with the input version and performs any module-specific setup activities. Memory management upcalls are also passed to the TP through this call.

Parameters

VerMajor (input)
The major version number of the TP module expected by the calling application.

VerMinor (input)
The minor version number of the TP module expected by the calling application.

Return Value

A CSSM_OK return value signifies that the current version of the TP module is compatible with the input version numbers and all setup operations were successfully performed. When CSSM_FAIL is returned, either the current TP module is incompatible with the requested TP module version or an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_TP_INITIALIZE_FAIL	Unable to perform module initialization

See Also

TP_Uninitialize

2.5.2 TP_Uninitialize

CSSM_RETURN CSSMTPI TP_Uninitialize (void)

This function performs any module-specific cleanup activities.

Parameters

None

Return Value

A CSSM_OK return value signifies that all cleanup operations were successfully performed. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_TP_UNINITIALIZE_FAIL	Unable to perform module cleanup

See Also

TP_Initialize

3. Trust Policy Structure and Management

3.1 Introduction

This section clarifies key aspects of the structure and management of Trust Policy modules. It describes the composition of a TP module, installation of a TP module, the expected behavior of a TP on attach, and basic services expected of TP functions. This section also includes example code fragments for TP functions.

3.2 Trust Policy Module Composition

A Trust Policy module is a dynamically-linkable library, composed of functions that implement some or all of the CSSM TPI described in Section 2. When the TP module is loaded, it is responsible for registering a function table with CSSM, accepting the memory management upcalls, and performing any module-specific setup. During TP unload, it is responsible for any cleanup the module requires. The remaining functions consist of some subset of the TPI determined by the TP developer.

The Trust Policy module composition can be broadly classified into the following categories:

- Registration with CSSM
- Memory Management
- Trust/Policy on certificates
- Trust/Policy on CRL
- Trust/Policy on actions
- PassThrough Operation Support

3.3 Trust Policy Module Installation

Before an application can use a Trust Policy module , the module's name, location, and module characteristics must be registered with CSSM by an installation application. The name of a Trust Policy module is both a logical name and a globally-unique identifier (GUID). The logical name is a readable string chosen by the Trust Policy developer to describe the TP module. The GUID is a structure used to differentiate between library modules in the CSSM registry. GUIDs are discussed in more detail below. The location of the TP module is required on installation so that CSSM can locate the module when an application requests an attach. The module characteristics are registered with CSSM at install time so that an application can query for TP module availability and features.

3.3.1 Global Unique Identifiers (GUIDs)

Each Trust Policy module must have a globally-unique identifier (GUID) which is used by CSSM, applications, and layered services to uniquely identify a TP. The TP GUID is used by the CSSM registry to expose add-in modules to applications. The TP module uses its GUID to identify itself when it sets an error.

A GUID is defined as:

```
typedef struct cssm_guid {
    uint32 Data1;
    uint16 Data2;
    uint16 Data3;
    uint8 Data4[8];
} CSSM_GUID, *CSSM_GUID_PTR;
```

GUID generators are publicly available for Windows* 95, Windows NT*, and on many UNIX* platforms.

3.3.2 Module characteristics

Version information must be supplied to the CSSM during the installation of the TP module. Applications use the information to determine the compatibility of the installed TP with its required TP. If the compatible versions are unknown to the application, it can pass the version number that it understands to the TP at attach time. At that time, the TP checks for compatibility and either attaches or fails accordingly.

3.4 Attaching a Trust Policy Module

Before an application can use the functions of a specific TP, it must attach the TP to CSSM using *CSSM_TP_Attach*. On attach, the Trust Policy module uses *CSSM_TP_RegisterServices* provided by the CSSM to register its function table. CSSM uses the TP module's function table to direct calls from the application to the correct function in the TP module. During the attach process, the TP's *TP_Initialize* function is called. At this time version compatibility is confirmed and a table of memory function upcalls is passed to the TP. The TP module uses the memory management upcalls to allocate any memory which will be returned to the calling application, and to free any memory which it received from the calling application.

When CSSM attaches to or detaches from a Trust Policy module, it initiates a function in the TP that performs the necessary setup and cleanup operations. The attach and detach functions vary depending on the target operating system for the Trust Policy module. For example, DllMain would be used to implement these functions in a TP targeted to Windows NT; _init and _fini are used in a TP targeted to SunOS*.

3.4.1 The TP module function table

The function table for a Trust Policy module contains pointers to the TP module's implementation of the functions specified in the Trust Policy Interface. This structure is specified as part of the CSSM header file, cssm.h, and contains all the prototypes of all the functions supported by the Trust Policy Interface. If a TP does not support some function in the TPI, the pointer to that function is set to NULL. See Section 4.2.5 for a definition of the structure.

3.4.2 Memory management upcalls

The calling application is responsible for all memory allocation and de-allocation for data passed between the application and the TP module via CSSM. The application provides memory management upcalls, which the TP module uses to return data to the application.

Memory management upcalls are pointers to the memory management functions used by the calling application. They are provided to the TP module via CSSM as a structure of function pointers. The functions are the calling application's equivalent of malloc, calloc, free and re-alloc and behave the same as those functions. The function parameters consist of the function's normal parameters. The function return values are interpreted in the standard manner. The TP module is responsible for making the memory management functions available to its internal functions.

3.5 Trust Policy Basic Services

3.5.1 Function implementation

A Trust Policy developer can implement some or all of the functions specified in the TPI. Section 2 describes the behavior of each function.

A Trust Policy developer can leverage the services of another TP, CL or DL module to implement certain functions. To do this, the TP attaches to another module using the appropriate *CSSM_Attach* calls. Subsequent function calls to the first TP call the corresponding function in the other add-ins for some or all of its implementation.

3.5.2 Error handling

If an error occurred, the function in the TP module calls *CSSM_SetError*. This function takes the module's GUID and an error number as inputs. The module's GUID is used to identify the error's location. The error number is used to describe the error.

The error number set by the TP module falls into one of two ranges. The first range of numbers is pre-defined by CSSM. These are errors common to all TP modules implementing a given function. They are described in this document as part of the function definitions in Sections 2.3 to 2.5. They are defined in the header file cssmerr.h, which is distributed as part of CSSM. The second range of error numbers defines module-specific error codes. These module-specific error codes are in the range of CSSM_TP_PRIVATE_ERROR to CSSM_TP_END_ERROR. CSSM_TP_PRIVATE_ERROR and CSSM_TP_END_ERROR are also defined in the header file cssmerr.h. The TP module developer is responsible for making the definitions and interpretation of their module-specific error codes available to applications.

If no error occurs, but the appropriate return value from a function is CSSM_FALSE, the function calls *CSSM_ClearError* before returning. The application is responsible for checking whether an error has occurred by calling *CSSM_GetError*. If the function in the TP module calls *CSSM_ClearError*, the calling application receives CSSM_OK response from *CSSM_GetError*, indicating no error has occurred.

3.6 Attach/Detach Example

The Trust Policy module performs certain operations when CSSM attaches to or detaches from it. TP modules that have been developed for Windows-based systems use the DllMain routine to perform those operations, as shown in the following example.

3.6.1 DllMain

```
#include “cssm.h”
CSSM_GUID tp_guid =
{ 0x83bafc39, 0xfac1, 0x11cf, { 0x81, 0x72, 0x0, 0xaa, 0x0, 0xb1, 0x99, 0xdd }
};

BOOL WINAPI DllMain (HANDLE hInstance, DWORD dwReason, LPVOID lpReserved)
{
    switch (dwReason)
    {
        case DLL_PROCESS_ATTACH:
        {
            CSSM_SPI_TP_FUNCS_PTR FunctionTable;

          /* Allocate TP memory for pointers */
          FunctionTable = (CSSM_SPI_TP_FUNCS_PTR)malloc (sizeof
                                (CSSM_SPI_TP_FUNCS));

          /* Initialize TP callback functions */
          FunctionTable->CertVerify = CertVerify;
          FunctionTable->CertSign = CertSign;
          FunctionTable->CertRevoke = CertRevoke;
          FunctionTable->CrlVerify = CrlVerify;
          FunctionTable->CrlSign = CrlSign;
          FunctionTable->ApplyCrlToDb = ApplyCrlToDb;
          FunctionTable->VerifyAction = VerifyAction;
          FunctionTable->PassThrough = NULL;

          FunctionTable->Initialize = TP_Initialize;
                    FunctionTable->Uninitialize = TP_Uninitialize;
            /* Call CSSM_TP_RegisterServices to register the FunctionTable */
            /* with CSSM and to receive the application’s memory upcall table */
            if (CSSM_TP_RegisterServices (&tp_guid, FunctionTable,
                                    &UpcallMemFunction) != CSSM_OK)
                return FALSE;

            break;
        }
    case DLL_THREAD_ATTACH:
        break;

    case DLL_THREAD_DETACH:
        break;

    case DLL_PROCESS_DETACH:
        if (CSSM_TP_DeregisterServices (&tp_guid) != CSSM_OK)
            return FALSE;
        break;
    }
return TRUE;
}
```

3.7 Trust Policy Operations Example

This section contains an example of an implementation of a function in the Trust Policy Module.

3.7.1 ApplyCrlToDb

```
/*-----------------------------------------------------------------------------
-
 * Name: ApplyCrlToDb
 *
 * Description:
 * This function applies a CRL to the persistent certificate storage.
 *
 * Parameters:
 * TPHandle (input) - Handle to the TP add-in.
 * CLHandle (input) - Handle to the CL add-in.
 * DLHandle (input) - Handle to the DL add-in.
 * DBHandle (input) - Handle to the database.
 * Crl (input) - Pointer to the CRL.
 *
 * Return value:
 * CSSM_FAIL - Unable to update permament store.
 * CSSM_OK - Permanent store has been updated to indicate revoked entries.
 *
 *
*-----------------------------------------------------------------------------*/
CSSM_RETURN CSSMTPI ApplyCrlToDb (CSSM_TP_HANDLE TPHandle,
                                  CSSM_CL_HANDLE CLHandle,
                                  CSSM_DL_HANDLE DLHandle,
                                  CSSM_DB_HANDLE DBHandle,
                                  const CSSM_DATA_PTR Crl)
{
    CSSM_DATA_PTR Cert = NULL;
    CSSM_HANDLE ResultsHandle;
    uint32 NumberOfMatchedCerts;
    CSSM_DB_CONJUNCTIVE Conjunctive = CSSM_NONE;

    /* Get the first certificate in the database */
    if ((Cert = CSSM_DL_CertGetFirst (DLHandle, DBHandle, NULL, 0,
              Conjunctive, &ResultsHandle, &NumberOfMatchedCerts)) != NULL) {

        /* Check to see if this certificate is present in the CRL */
        if (CSSM_CL_IsCertInCrl (CLHandle, Cert, Crl) == CSSM_TRUE)
            /* Revoke certificate when present in the CRL */
            if (CSSM_DL_CertRevoke (DLHandle, DBHandle, Cert) != CSSM_OK) {
                CSSM_DL_CertAbortQuery (DLHandle, DBHandle, ResultsHandle);
                return (CSSM_FAIL);
            }

        /* Cycle through all the certificates in the database */
        while ((Cert = CSSM_DL_CertGetNext (DLHandle, DBHandle,
                  ResultsHandle)) != NULL) {

            /* For each entry check to see if it is present in the CRL */
            if (CSSM_CL_IsCertInCrl (CLHandle, Cert, Crl) == CSSM_TRUE)
                /* Revoke certificate when present in the CRL */
                if (CSSM_DL_CertRevoke (DLHandle, DBHandle, Cert) != CSSM_OK)
{
                    CSSM_DL_CertAbortQuery (DLHandle, DBHandle,
                                ResultsHandle);
                    return (CSSM_FAIL);
                }
        }
```

```
        /* Abort query when there are no more entries */
        CSSM_DL_CertAbortQuery (DLHandle, DBHandle, ResultsHandle);
         return (CSSM_OK);
    } else
        return (CSSM_FAIL);
}
```

4. Appendix A, Relevant CSSM API functions

4.1 Overview

There are several API functions particularly relevant to Trust Policy developers, because they are used by either the application to access the TP module or by the TP module to access CSSM services, such as the CSSM registry or the error-handling routines. They are included in this appendix for quick reference. For additional information, see the *CSSM Application Programming Interface*.

4.2 Data Structures

4.2.1 CSSM_DATA

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

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

Definition:

Length - The length, in bytes, of the memory block pointed to by *Data*.

Data - A pointer to a contiguous block of memory.

4.2.2 CSSM_GUID

A GUID is a globally-unique identifier used to uniquely identify a TP.

```
typedef struct cssm_guid {
    uint32 Data1;
    uint16 Data2;
    uint16 Data3;
    uint8 Data4[8];
} CSSM_GUID, *CSSM_GUID_PTR;
```

4.2.3 CSSM_TPINFO

Trust Policies have certain common characteristics, which are made available to applications. These characteristics are given by the CSSM_TPINFO structure, which is registered with CSSM during installation for an application's query.

```
typedef struct cssm_tpinfo{
    uint32 VerMajor;
    uint32 VerMinor;
}CSSM_TPINFO, *CSSM_TPINFO_PTR
```

Definition:

VerMajor - The major version number of the add-in module.

VerMinor - The minor version number of the add-in module.

4.2.4 CSSM_SPI_MEMORY_FUNC

This data structure contains function pointers to the calling application's memory management routines. The TP module uses these routines to allocate and free any memory returned to the client.

```
typedef struct cssm_spi_memory_func {
    void *(*malloc_func) (CSSM_HANDLE AddInHandle, uint32 size);
    void (*free_func) (CSSM_HANDLE AddInHandle, void *memblock);
    void  *(*realloc_func) (CSSM_HANDLE AddInHandle, void *memblock, uint32
size);
    void *(*calloc_func) (CSSM_HANDLE AddInHandle, uint32 num, uint32 size);
}CSSM_SPI_MEMORY_FUNC, *CSSM_SPI_MEMORY_FUNC_PTR;
```

Definition:

malloc_func - pointer to function that returns a void pointer to the allocated memory block of at least *size* bytes.

free_func - pointer to function that deallocates a previously-allocated memory block (*memblock*).

realloc_func - pointer to function that returns a void pointer to the reallocated memory block(*memblock*) of at least *size* bytes.

calloc_func - pointer to a function that returns a void pointer to an array of *num* elements of length *size* initialized to zero.

4.2.5 CSSM_SPI_TP_FUNCS

This data structure contains function pointers to the routines that a TP module can support. The function prototypes are provided for compiler checking when assigning function pointers to the structure. This structure is used during the registration of the TP's services. For a description of each function, refer to section 2.3, Trust Policy Operations.

```
typedef struct cssm_spi_tp_funcs {
    CSSM_BOOL (CSSMAPI *CertVerify) (CSSM_TP_HANDLE TPHandle,
                                     CSSM_CL_HANDLE CLHandle,
                                     CSSM_DL_HANDLE DLHandle,
                                     CSSM_DB_HANDLE DBHandle,
                                     CSSM_CC_HANDLE CCHandle,
                                     const CSSM_DATA_PTR SubjectCert,
                                     const CSSM_DATA_PTR SignerCert,
                                     const CSSM_FIELD_PTR VerifyScope,
                                     uint32 ScopeSize);
    CSSM_DATA_PTR (CSSMAPI *CertSign) (CSSM_TP_HANDLE TPHandle,
                                       CSSM_CL_HANDLE CLHandle,
                                       CSSM_DL_HANDLE DLHandle,
                                       CSSM_DB_HANDLE DBHandle,
                                       CSSM_CC_HANDLE CCHandle,
                                       const CSSM_DATA_PTR SubjectCert,
                                       const CSSM_DATA_PTR SignerCert,
                                       CSSM_FIELD_PTR SignScope,
                                       uint32 ScopeSize);
    CSSM_DATA_PTR (CSSMAPI *CertRevoke) (CSSM_TP_HANDLE TPHandle,
                                         CSSM_CL_HANDLE CLHandle,
                                         CSSM_DL_HANDLE DLHandle,
                                         CSSM_DB_HANDLE DBHandle,
                                         CSSM_CC_HANDLE CCHandle,
                                         const CSSM_DATA_PTR OldCrl,
```

```
                                        const CSSM_DATA_PTR SubjectCert,
                                        const CSSM_DATA_PTR RevokerCert,
                                        CSSM_REVOKE_REASON Reason);
    CSSM_BOOL (CSSMAPI *CrlVerify) (CSSM_TP_HANDLE TPHandle,
                                    CSSM_CL_HANDLE CLHandle,
                                    CSSM_DL_HANDLE DLHandle,
                                    CSSM_DB_HANDLE DBHandle,
                                    CSSM_CC_HANDLE CCHandle,
                                    const CSSM_DATA_PTR SubjectCrl,
                                    const CSSM_DATA_PTR SignerCert,
                                    const CSSM_FIELD_PTR VerifyScope,
                                    uint32 ScopeSize);
    CSSM_DATA_PTR (CSSMAPI *CrlSign) (CSSM_TP_HANDLE TPHandle,
                                      CSSM_CL_HANDLE CLHandle,
                                      CSSM_DL_HANDLE DLHandle,
                                      CSSM_DB_HANDLE DBHandle,
                                      CSSM_CC_HANDLE CCHandle,
                                      const CSSM_DATA_PTR SubjectCrl,
                                      const CSSM_DATA_PTR SignerCert,
                                      const CSSM_FIELD_PTR SignScope,
                                      uint32 ScopeSize);
    CSSM_RETURN (CSSMAPI *ApplyCrlToDb) (CSSM_TP_HANDLE TPHandle,
                                         CSSM_CL_HANDLE CLHandle,
                                         CSSM_DL_HANDLE DLHandle,
                                         CSSM_DB_HANDLE DBHandle,
                                         const CSSM_DATA_PTR Crl);
    CSSM_RETURN (CSSMAPI *VerifyAction) (CSSM_TP_HANDLE TPHandle,
                                         CSSM_CL_HANDLE CLHandle,
                                         CSSM_DL_HANDLE DLHandle,
                                         CSSM_DB_HANDLE DBHandle,
                                         CSSM_CC_HANDLE CCHandle,
                                         CSSM_TP_ACTION Action,
                                         const CSSM_DATA_PTR Data,
                                         const CSSM_DATA_PTR Cert);
    CSSM_DATA_PTR (CSSMAPI *PassThrough) (CSSM_TP_HANDLE TPHandle,
                                          CSSM_CL_HANDLE CLHandle,
                                          CSSM_DL_HANDLE DLHandle,
                                          CSSM_DB_HANDLE DBHandle,
                                          CSSM_CC_HANDLE CCHandle,
                                          uint32 PassThroughId,
                                          const CSSM_DATA_PTR InputParams);

CSSM_RETURN (CSSMAPI *Initialize)    (uint32 VerMajor,
                                                                    uint32
                                    VerMinor);
CSSM_RETURN (CSSMAPI *Uninitialize)  (void);
} CSSM_SPI_TP_FUNCS, *CSSM_SPI_TP_FUNCS_PTR;
```

4.3 Function Definitions

This section describes the API provided by the CSSM for TP developers to communicate with the CSSM and other add-in modules. For a complete description of the CSSM API refer to the *CSSM Application Programming Interface*.

4.3.1 CSSM_TP_Install

CSSM_RETURN CSSMAPI CSSM_TP_Install (const char *TPName,
const char *TPFileName,
const char *TPPathName,
const CSSM_GUID_PTR GUID,
const CSSM_TPINFO_PTR TPInfo,
const void * Reserved1,
const CSSM_DATA_PTR Reserved2)

This function updates the CSSM-persistent internal information about the TP module.

Parameters

TPName (input)
The name of the Trust Policy module.

TPFileName (input)
The name of the file that implements the Trust Policy.

TPPathName (input)
The path to the file that implements the Trust Policy.

GUID (input)
A pointer to the CSSM_GUID structure containing the global unique identifier for the TP module.

TPInfo (input)
A pointer to the CSSM_TPINFO structure containing information about the TP module.

Reserved1 (input)
Reserve data for the function.

Reserved2 (input)
Reserve data for the function.

Return Value

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

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Error in the registry

See Also

CSSM_TP_Uninstall

4.3.2 CSSM_TP_Uninstall

CSSM_BOOL CSSMAPI CSSM_TP_Uninstall (const CSSM_GUID_PTR GUID)

This function deletes the persistent CSSM internal information about the TP module.

Parameters

GUID (input)
A pointer to the CSSM_GUID structure containing the global unique identifier for the TP module.

Return Value

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

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_INVALID_ GUID	Certificate library was not installed
CSSM_REGISTRY_ERROR	Unable to delete information

See Also

CSSM_TP_Install

4.3.3 CSSM_TP_RegisterServices

CSSM_RETURN CSSMAPI CSSM_TP_RegisterServices
(const CSSM_GUID_PTR GUID,
const CSSM_SPI_TP_FUNCS_PTR FunctionTable,
CSSM_SPI_MEMORY_FUNCS_PTR UpcallTable,
void *Reserved)

A Trust Policy module uses this function to register its function table with CSSM and to receive a memory management upcall table from CSSM.

Parameters

GUID (input)
A pointer to the CSSM_GUID structure containing the global unique identifier for the TP module.

FunctionTable (input)
A structure containing pointers to the Trust Policy Interface functions implemented by the TP module.

UpcallTable (output)
A structure containing pointers to the memory routines used by the TP module to allocate and free memory returning to the calling application.

Reserved (input)
A reserved input.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_INVALID_FUNCTION_TABLE	Invalid function table
CSSM_MEMORY_ERROR	Memory error
CSSM_REGISTRY_ERROR	Unable to register services

See Also

CSSM_TP_DeregisterServices

4.3.4 CSSM_TP_DeregisterServices

CSSM_RETURN CSSMAPI CSSM_TP_DeregisterServices (const CSSM_GUID_PTR GUID)

A Trust Policy module uses this function to deregister its services from the CSSM.

Parameters

GUID (input)
A pointer to the CSSM_GUID structure containing the global unique identifier for the TP module.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer GUID
CSSM_MEMORY_ERROR	Unable to deregister services

See Also

CSSM_TP_RegisterServices

4.3.5 CSSM_TP_Attach

CSSM_TP_HANDLE CSSMAPI CSSM_TP_Attach (const CSSM_GUID_PTR GUID,
uint32 CheckCompatibleVerMajor,
uint32 CheckCompatibleVerMinor,
const void * Reserved)

This function attaches the TP module and verifies that the version of the module expected by the application is compatible with the version on the system.

Parameters

GUID (input)
A pointer to the CSSM_GUID structure containing the global unique identifier for the TP module.

CheckCompatibleVerMajor(input)
The major version number of the TP module that the application is compatible with.

CheckCompatibleVerMinor(input)
The minor version number of the TP module that the application is compatible with.

Reserved (input)
A reserved input.

Return Value

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

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_MEMORY_ERROR	Internal memory error
CSSM_INCOMPATIBLE_VERSION	Incompatible version
CSSM_EXPIRE	Add-in has expired
CSSM_ATTACH_FAIL	Unable to load TP module

See Also

CSSM_TP_Detach

4.3.6 CSSM_TP_Detach

CSSM_RETURN CSSMAPI CSSM_TP_Detach (CSSM_TP_HANDLE TPHandle)

This function detaches the application from the TP module.

Parameters

TPHandle (input)
The handle that describes the TP module.

Return Value

A CSSM_TRUE return value means the application has been detached from the TP module. If CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_ADDIN_HANDLE	Invalid TP handle

See Also

CSSM_TP_Attach

4.3.7 CSSM_Free

void CSSMAPI CSSM_Free (void *MemPtr, CSSM_HANDLE AddInHandle)

This function frees the memory allocated by add-in.

Parameters

MemPtr (input)
A pointer to the memory to be freed.

AddInHandle (input)
The handle to add-in module that needs to free memory

Return Value

None

Error Codes

None

4.3.8 CSSM_GetAPIMemoryFunctions

CSSM_API_MEMORY_FUNCS_PTR CSSMAPI CSSM_GetAPIMemoryFunctions
(CSSM_HANDLE AddInHandle)

This function retrieves the application's memory function table associated with the add-in module.

Parameters

AddInHandle (input)
The handle to add-in module that is associated to memory function table.

Return Value

Non NULL if the function was successful. NULL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_ADDIN_HANDLE	Invalid add-in handle
CSSM_MEMORY_ERROR	Internal memory error

4.3.9 CSSM_GetError

CSSM_ERROR_PTR CSSMAPI CSSM_GetError (void)

This function returns the current error information.

Parameters

None

Return Value

Returns the current error information. If there is currently no valid error, the error number is CSSM_OK. A NULL pointer indicates the CSSM_InitError was not called by the CSSM Core or that CSSM Core made a call to CSSM_DestroyError. No error information is available.

See Also

CSSM_ClearError, CSSM_SetError

4.3.10 CSSM_SetError

CSSM_RETURN CSSMAPI CSSM_SetError (CSSM_GUID_PTR guid,
uint32 error_number)

This function sets the current error information to *error_number* and *guid.*

Parameters

guid (input)
Pointer to the GUID (global unique ID) of the add-in module.

error_number (input)
An error number. It falls within one of the valid CSSM, CL, TP, DL, or CSP error ranges.

Return Value

CSSM_OK if error was successfully set. A return value of CSSM_FAIL indicates the error number passed is not within a valid range, the GUID passed is invalid, CSSM_InitError was not called by the CSSM Core, or the CSSM core called CSSM_DestroyError. No error information is available.

See Also

CSSM_ClearError, CSSM_GetError

4.3.11 CSSM_ClearError

void CSSMAPI CSSM_ClearError (void)

This function sets the current error value to CSSM_OK. This is called if the current error value has been handled and therefore is no longer a valid error.

Parameters

None

See Also

CSSM_SetError, CSSM_GetError