Common
Data Security Architecture
Specification

Draft 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 conformant 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 its 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

List of Figures

1. Introduction

The Common Data Security Architecture (CDSA) is a set of layered security services that address communications and data security problems in the emerging Internet and Intranet application space. Intel Architecture Labs (IAL) defined the CDSA to:

- Encourage open, interoperable, horizontal security standards
- Offer essential components of security capability to the industry at large

The motivation for a robust, broadly diffused, multi-platform, industry-standard security infrastructure is clear. The definition of such an infrastructure, however, must accommodate the emerging Internet and Intranet business opportunities and address the requirements unique to the most popular client systems, namely personal computers (PCs) and networked application servers.

CDSA focuses on security in peer-to-peer distributed systems with homogeneous and heterogeneous platform environments. The architecture also applies to the components of a client-server application. The CDSA addresses security issues in a broad range of applications, including:

- Electronic commerce in business-to-business and home-to-business applications. This implies a selectable range of security solutions.
- Content distribution of software, reference information, educational material, or entertainment content requiring new algorithms and protocols.
- Metering of content, service, or both, and the requirement for secure storage of state and value.
- Securing business or personal activity for private email, home banking, and monetary transactions where the value, and thus the threat, may be quite varied.

The architecture addresses the security requirements of this broad range of applications by:

- Providing layered security mechanisms (not policies)
- Supporting application-specific policies by providing an extensibility mechanism that manages add-in (policy-specific) modules
- Exposing flexible service provider interfaces that may accommodate a broad range of certificate and document formats, such as X.509, SPKI, SDSI and EDI
- Supporting secure protocols such as SSL and SET

1.1 Audience

This document provides on overview of the CDSA for Independent Software Vendors (ISVs) and Independent Hardware Vendors (IHVs), who develop security products as complete applications or plug-ins to extensible platforms. These ISVs and IHVs include:

- Experienced software and hardware designers
- Security architects who work in high-end cryptography
- Advanced programmers
- Sophisticated integrators familiar with numerous forms of network computing

This audience understands their requirements for a ubiquitous security infrastructure upon which they can build security-aware application products or through which they can offer their security toolkit products.

The primary infrastructure component of the CDSA is the Common Security Services Manager (CSSM). Companion documents define interface requirements for application developers who use CSSM services and tool developers who provide add-in functionality to CSSM. These documents include:

API *CSSM Application Programming Interface Specification* — defines the interface that applications developers employ to access security services provided by CSSM and add-in security modules.

SPI *CSSM Cryptographic Service Provider Interface Specification* — defines the interface to which cryptographic service providers must conform in order to be accessible via CSSM.

TPI *CSSM Trust Policy Interface Specification* — defines the interface to which policy makers, such as Certificate Authorities (CAs), Certificate Issuers, and policy-making application developers must conform in order to extend CSSM with model or application-specific policies.

CLI *CSSM Certificate Library Interface Specification* — defines the interface to which library developers must conform to provide format-specific certificate manipulation services to numerous applications and trust policy modules.

DLI *CSSM Data Storage Library Interface Specification* — defines the interface to which library developers must conform to provide format-specific or format-independent persistent storage of certificates.

CSSM Java *CSSM Java* Application Programming Interface Specification* — defines a set of Java classes and methods that Java applets and Java applications must use to access CSSM-managed security services.

1.2 Document Organization

This document has nine sections:

- Introductory material about CDSA
- Common Security Services Manager (CSSM)
- Cryptographic Services Manager
- Trust Policy Services Manager
- Certificate Library Services Manager
- Data Storage Library Services Manager
- Integrity Services Manager
- System Security Services
- Glossary

1.3 Status of This Specification

This document describes the CDSA specification. We welcome constructive comments and feedback for this document. Please direct your suggestions via email to:

CDSA@ibeam.intel.com

1.4 Other References

BSAFE*	*BSAFE Cryptographic Toolkit*, RSA Data Security, Inc., Redwood City, CA
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)
Cryptography	*Applied Cryptography, Second Edition Protocols, Algorithms, and Source Code in C*, Bruce Schneier: John Wiley & Sons, Inc., 1996

1.5 Common Data Security Architecture

The Common Data Security Architecture defines the infrastructure for a comprehensive set of security services. Cryptography is the computational base used to build security protocols and security systems. CDSA is an extensible architecture that provides mechanisms to dynamically manage add-in security modules. Figure 1 shows the three 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 add-in security modules. Four types of modules are defined:

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

Add-in security modules may be provided by independent software and hardware vendors as competitive products. Applications may direct their requests to modules from specific vendors or to any module that performs the required services. Add-in modules augment the set of available security services.

CDSA's extensible architecture allows new module types to be included which accommodate prudent division of labor. Signing services and key management services can be added at the System Security Services Layer and the Security Add-in Modules layer in CDSA. An appropriate degree of visibility of lower layers may be reflected at higher layers such that a complete security profile can be managed uniformly. Independent software and hardware vendors may specialize in their chosen area of expertise. For example, hardware-specific cryptographic device vendors may also provide tamper-resistant storage facilities.

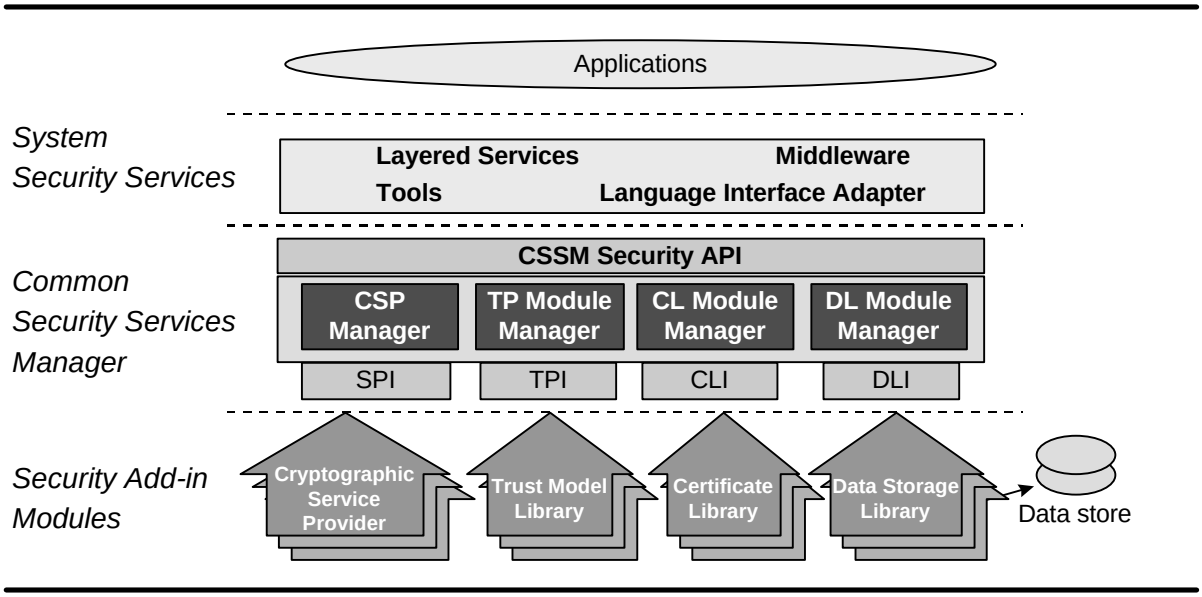

Figure 1. The Common Data Security Architecture for all platforms.

1.5.1 Architectural Assumptions

The CDSA design follows five architectural principles:

- **A layered service provider model** CDSA is built up from a set of horizontal layers, each providing services to the layer above it. This approach is portable, adaptable, modular, and open.
- **Open architecture**. The CDSA is fully disclosed for peer review by the industry.
- **Modularity and extensibility** Components of each layer can be chosen as separate modules. An extensible framework supports inserting domain-specific functions. Extensibility fosters industry growth by encouraging development of incremental functionality and performance-competitive implementations of each add-in module.
- **Value in managing the details** The CDSA can manage security details, so individual applications do not need to be concerned with security-related details.
- **Emerging paradigms** The CDSA is built on emerging data security paradigms.

Regarding the emerging data security paradigms, the architecture is built on two fundamental premises:

Portable digital tokens are used as a person's digital persona for commerce, communications, and access control. These digital tokens are encryption modules, with some amount of encrypted storage. They can be software or hardware, depending on the application's security needs. They come in various form factors, and may have multiple functions aggregated into a single device, such as a digital wallet.

Digital certificates can be used to embody trust. These certificates do not create any new trust models or relationships. They are the digital form for current trust models. A person may have a certificate for each trust relationship (such as multiple credit cards, checkbooks, employer ID).

The ability of client platforms to accommodate these two new technologies is critical to the success of such platforms for digital commerce and information management as the Intranet extends seamlessly into the Internet.

1.6 The Threat Model

Malicious observation and manipulation of computer systems can be classified into three categories, based on the origins of threats. The origins of threats are expressed in terms of the security perimeter that's been breached in order to effect the malicious act:

Category I - The malicious threat originates outside of the computer system. The perpetrator breaches communications access controls, but still operates under the constraints of the communications protocols. This is the standard hacker attack.

Category I attacks are best defended by correctly designed and implemented access-control protocols and mechanisms, and proper system administration, rather than by the use of tamper-resistant software. Frequently, the goal of a Category I attack is to mount a Category II attack.

Category II - The malicious attack originates as software running on the platform. The perpetrator introduces malicious code onto the platform and the operating system executes it. The attack moves inside the communications perimeter, but remains bounded by the operating system and BIOS, using their interfaces. The malicious software may have been introduced with or without the user's consent. This is the common virus attack.

Examples include viruses, Trojan horses, and software used to discover secrets stored in other software (such as another user's access control information). Category II attacks tend to attack classes of software. Viruses are a good example. Viruses must assume certain coding characteristics to be constant among its target population, such as the format of the execution image. It's the consistency of software across individual computer systems and even across platforms that enables Category II attacks.

Category III - The perpetrator completely controls the platform, may substitute hardware or system software, and may observe any communications channel (such as using a bus analyzer). This attack faces no security perimeter and is limited only by technical expertise and financial resources.

In an absolute sense, Category III attacks are impossible to prevent on the computer system. Defense against a Category III attack merely raises a technological bar to a height sufficient to deter a perpetrator by providing a poor return on investment. That investment might be measured in terms of the tools necessary, or the skills required to observe and modify the software's behavior. The technological bar, from low to high would be:

- No special analysis tools required (such as debuggers and system diagnostic tools)
- Specialized software analysis tools (such as SoftIce*)
- Specialized hardware analysis tools (such as processor emulators and bus-logic analyzers)

The CSSM's goal is to defend against Category II and Category III attacks, to the level of specialized hardware analysis tools. This provides a reasonable compromise. As threat follows value, this level of tamper resistance is adequate for low-to-medium-value applications and high-value applications where the user is unlikely to be a willing perpetrator (such as applications involving the user's personal property).

1.7 Architectural Overview

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

These layers start with fundamental components such as cryptographic algorithms, random numbers, and unique identification information, and build up to digital certificates, key management mechanisms and secure transaction protocols in higher layers. CDSA is intended to be the multi platform security architecture that's horizontally broad and vertically robust.

As Figure 1 shows, the CDSA defines three basic layers:

- System Security Services
- The Common Security Services Manager (CSSM)
- Security Add-in Modules (cryptographic service providers, trust policy modules, certificate library modules, and data storage library modules)

1.7.1 System Security Services Layer

The System Security Services layer is between applications and basic CSSM security services. Software at this layer may

- Define high-level security abstractions (such as secure electronic mail services)
- Provide transparent security services (such as secure file systems or private communication)
- Make CSSM security services accessible to applications developed in languages other than the C language
- Provide tools to manage the security infrastructure

Some layered services allow applications to use security services without explicit calls to the basic CSSM security APIs. This approach allows applications to implicitly use security services (without changing the application) by using these standard system level services. Examples include

- SHTTP and SSL for secured network communications
- PGP* for secured files

New security-related layered services can also be developed and explicitly used by applications that have only a high-level conceptual awareness of security. Examples include:

- JEPI and SET* protocols for secure electronic commerce
- PGP for secure and private electronic mail

Another category of layered service is the language interface adapter. A language adapter extends the CSSM API calls (defined in C language) to other programming languages and programming environments. These language-specific wrappers may export the CSSM C language API calls directly to another language, or may abstract the CSSM concepts and present them through the target language. For example, a CSSM-Java package defines object-oriented classes and methods by which Java applications and Java applets can use security functionality provided by and through CSSM.

CSSM accommodates many new and existing standards as layered services. The broad spectrum of layered security services is easier to implement by virtue of CSSM's modularity.

1.7.2 Common Security Services Manager Layer

The second level of CDSA is the Common Security Services Manager (CSSM). CSSM, the essential component in the CDSA model integrates and manages all the security services. It enables tight integration of individual services, while allowing those services to be provided by interoperable modules. The CSSM has a rich API to support developing secure applications, and system services and SPI supporting plug-in security modules that implement building blocks for secure operations.

The CSSM employs multilateral integrity verification protocol between add-in modules, applications and CSSM.

Bilateral authentication, integrity verification and authorization are done during dynamic binding. Candidate modules present a certificate and engage in integrity checks. Accompanying authentication certificates are signed credentials describing the privileges and capabilities of the authenticating module. The credential signature chain identifies entities trusted to issue credentials according to regulations of interested governing bodies.

Credentials enables the CSSM to accomodate and enforce varying levels of cryptography. Certain classes of applications may make less restrictive use of cryptographic than generic applications. For example financial applications may use strong cryptography to protect overseas transactions. Even more selective than classes of applications, specific applications may be described in a credential enabling highly specialized uses of strong cryptography.

CSSM does not prescribe or implement certificate-based security policy. Application-specific security policies are defined and implemented by add-in modules and layered services. The CSSM integrates and manages the fundamental components of security and defines a common API for accessing the services provided by add-in modules. Similarly, the CSSM contains no cryptographic algorithms. CSSM maps API calls to appropriate add-in modules through the Service Providers Interface (SPI). It's commonly referred to as "crypto with a hole."

CSSM provides security context services to assist applications in managing the numerous parameters required for cryptographic operations during application execution sessions. CSSM also provides essential integrity services, such as self-checking the digital signature of the CSSM executable to detect tampering.

By managing security, the CSSM removes this burden from users and applications. CSSM management services are partitioned among four managers:

- Cryptographic Services Manager
- Trust Policy Services Manager
- Certificate Services Manager
- Data Store Services Manager

The **Cryptographic Services Manager** administers the Cryptographic Service Providers that may be installed on the local system. It defines a common API for accessing all of the Cryptographic Service Providers that may be installed beneath it. All cryptography functions are implemented by the CSPs. The manager administers a registry of local CSPs that can be queried. The registry lists the locally accessible CSPs and their cryptographic services (and algorithms).

The **Trust Policy Services Manager** administers the trust policy modules that may be installed on the local system. It defines a common API for these libraries. The API allows applications to request security services that require "policy review and approval" as the first step in performing the operation.

Operations defined in the API include verifying trust in

- A certificate for signing or revoking another certificate

- A user or user-agent to perform an application-specific action
- The issuer of a certificate revocation list

All policy-specific tests and decisions are implemented by the add-in trust policy module. Application-invoked calls are dispatched to the appropriate module. The CSSM trust policy services manager administers a queryable registry of locally-accessible modules.

The **Certificate Services Manager** administers the Certificate Libraries that may be installed on the local system. It defines a common API for these libraries. The API allows applications to manipulate memory-resident certificates and certificate revocation lists.

Operations defined in the API include create, sign, verify, and extract field values. All certificate operations are implemented by the add-in certificate libraries. Application-invoked calls are dispatched to the appropriate library module. Each library incorporates knowledge of certificate data formats and how to manipulate that format. The CSSM Certificate Services Manager administers a registry of local libraries that can be queried. The registry enumerates the locally-accessible libraries and attributes of those libraries, such as the certificate type manipulated by each registered library.

The **Data Store Services Manager** defines an API for secure, persistent storage of certificates, certificate revocation lists (CRLs) and other security objects. The API allows applications to search and select stored data objects, and to query meta-information about each data store (such as its name, date of last modification, size of the data store, etc.) Data store operations are implemented by add-in data storage library modules.

These modules may be drivers or gateways to traditional, full-featured Database Management Systems (DBMS), customized services layered over a file system, or access to other forms of stable storage. A data storage module may execute and store its data locally or remotely.

Future versions of CSSM could add additional service manager types to the four described here. The security services managers are logical partitions of functionality. As the spectrum of security services enlarges, new codification of security functionality may emerge. CSSM can naturally accommodate such eventualities by introducing new manager interfaces.

1.7.3 Security Add-in Modules Layer

CDSA envisions four logical types of Security add-in modules:

- Cryptographic Service Providers (CSPs)
- Trust Policy modules (TPs)
- Certificate Library modules (CLs)
- Data Storage Library Modules (DLs)

Add-in module implementers may register interfaces in multiple categories. For example, a hardware cryptographic token vender may register CSP and DL interfaces which may capitalize on the vendor's tamper resistant persistent storage technology. Other vendors may find synergy in supporting both TP and CL modules. Registration of services with CSSM is distinct for each logical interface, allowing add-in providers the flexibility of specializing in a particular class of add-ins.

1.7.3.1 Cryptographic Service Providers (CSPs)

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

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

A CSP may be instanciated in software, hardware, or both. Installing a CSP with the CSSM Cryptographic Services Manager makes that CSP accessible for use on the local system. The installation process records, in a persistent registry, the CSP's identifying name, a list of cryptographic services provided by the CSP, and the information required to dynamically load the CSP. Applications may query the registry and select one or more CSPs based on their capabilities.

All CSPs must enable encrypted storage for private keys and variables. CSPs or an independent module can also deliver key management services, such as key escrow or key recovery. As a minimum, CSPs do not reveal key material unless it's been wrapped, but they must support importing, exporting, and key generation.

The key-generation module of a CSP should be made tamper resistant. The CSSM Integrity Services provides mechanisms that may be used by a CSP to make itself more tamper resistant.

1.7.3.2 Trust Policy modules (TPs)

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 should be supported by every TP module. Each module may choose to implement the subset of these operations that are required for its policy.

The CSSM API defines a pass-through function. This mechanism allows each module to provide additional functions along with those defined by the CSSM Trust Policy API. When a TP function has determined the trustworthiness of performing an action, the TP function 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.

1.7.3.3 Certificate Library Modules (CLs)

Certificate library modules implement syntactic manipulation of memory-resident certificates and certificate revocation lists. The CSSM Certificate API defines the generic operations that should be supported by every CL module. Each module may choose to implement only those operations required to manipulate a specific certificate data format, such as X.509, SDSI, etc.

The implementation of these operations should be semantic-free. Semantic interpretation of certificate values should be implemented in TP modules, layered services, and applications. A pass-through function

is defined in the CL API. This mechanism allows each CL to provide additional functions to manipulate certificate and certificate revocation lists in memory.

The CSSM architecture makes manipulation of certificates and certificate revocation lists orthogonal to persistence of those objects. Hence, it is not recommended that CL modules invoke the services of data storage library modules. Decisions regarding persistence should be made by TP modules, layered security services, and applications.

CL modules must be installed and registered with the CSSM Certificate Services Manager. Applications may query the Services Manager to retrieve properties of the CL module as defined during installation.

CL modules may implement remote signing capabilities such as X.509 CertificateRequestMessage and X.9 DelegationRequestMessages. Remote signing requests may use standard message protocols such as PKCS#10 and may be transport independent. Such capabilities are registered with CSSM at install time.

1.7.3.4 Data Storage Library Modules (DLs)

A Data storage library module provides stable storage for security-related data objects. These objects can be certificates, certificate revocation lists (CRLs), cryptographic keys, policy objects or application-specific objects. Stable storage could be provided by a

- Commercially-available database management system product
- Native file system
- Custom hardware-based storage devices
- Interface to remote storage services
- In-memory storage

Each DL module may choose to implement only those operations required to provide persistence under its selected model of service.

The implementation of DL operations should be semantic-free. Semantic interpretation of stored objects such as certificate values, CRL values, key values, and policy should be interpreted by layered services, TP modules and applications. An extensible function interface is defined in the DL API. This mechanism allows each DL to provide additional functions to store and retrieve security objects, such as performance-enhancing retrieval functions or unique administrative functions required by the nature of the implementation.

1.8 Multi-platform Industry Architecture

The CDSA represents an accumulation of security requirements and solutions gathered across the multi-platform industry. It's being offered to the industry as the overall framework for data security on PCs, workstations, servers, and other computing platforms. A reference implementation of CDSA runs on the Windows* 95 and Windows NT* platforms. It may also be feasible to implement CDSA on UNIX* or other platforms. The CDSA reference implementation includes the core CSSM infrastructure, one or more add-ins of each type, SSL secure communications service layer, and several example applications. Other operating system environments and extended platforms, such as Java*, are also being addressed.

2. Common Security Services Manager

This section provides details on the main infrastructure component of the CDSA, the Common Security Services Manager (CSSM).

2.1 Overview

The Common Security Services Manager integrates the security functions required by applications to use 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 details of the CSSM.

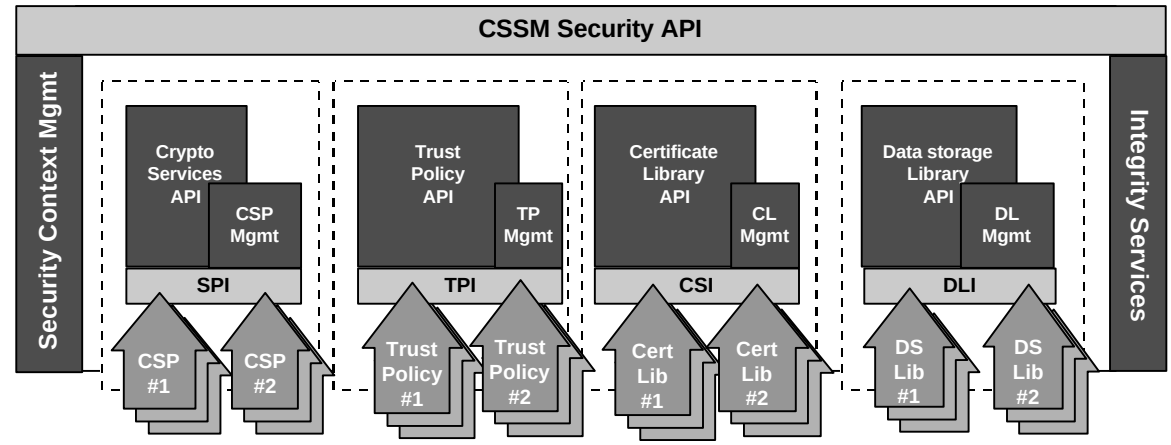

Figure 2. Design of the Common Security Services Manager

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 is partitioned among four service roles: cryptography, trust policy enforcement, certificate manipulations, and persistent storage. The usage model for each partition varies according to intrinsic operations.

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

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 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, certificate revocation lists and other types of objects. Certificate Libraries and Data Storage Libraries make the existence and manipulation of security objects 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.

Modules register meta-information about their capabilities, services, and operating modes. CSSM retains this information in secured persistent storage used only by CSSM. Applications query the CSSM to discover capabilities of installed modules. Applications may use meta-information to aid in selecting appropriate add-in modules.

Add-in modules may be provided by independent software and hardware vendors. Applications directly or indirectly select the modules that will be used to provide security services. Add-in modules may invoke other add-in modules to implement portions of their operations.

The API calls defined for each add-in module is 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.

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. Add-in modules also have integrity checking facilities that not only check themselves before attaching to CSSM but also checks the integrity of CSSM. The integrity checking facilities are constructed in such a way that class attacks are extremely unlikely. Cross-checking of all CSSM components further diminishes an attacker's ability to subvert the system. The CSSM integrity manager provides a built-in audit capability, which logs the occurrence of a selectable set of security operations.

Security-context management provides run-time 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. Figure 3 shows how calls to the CSSM security API are managed by the CSSM dispatch mechanism. The dispatcher forwards function calls to selected

- Trust policy modules
- Certificate library modules
- Data storage library modules
- Cryptographic service provider modules

A unique handle is used to identify a session between the attached add-in module and the caller. A single identifier is used repeatedly for different function calls between the same caller and target module. A caller uses the handle to specify its target add-in module, and the CSSM dispatcher forwards the call to the selected module. CSSM's use of handles enables add-in modules to be re-entrant.

In Figure 3, the application invokes func1 in the cryptographic module identified by the handle CSP1. The CSSM dispatcher forwards the function call to func1 in the CSP1 module. The application also invokes func7 in the trust policy module identified by the handle TP2. Again the CSSM dispatcher forwards the function call to func7 in the TP2 module. The implementation of func7 in the TP2 modules uses functions

implemented by a certificate library module. The TP2 module must invoke the certificate library functions via the CSSM dispatcher. To accomplish this, the TP2 module attaches the certificate library module, obtaining the handle CL1, and invokes func13 in the certificate library identified by the handle CL1. The CSSM dispatcher forwards the function call to func13 in the CL1 module.

Each attach call results in a unique attach handle. CSSM maintains separate context state for each attach operation. This enables non-cooperating threads of execution the freedom to maintain their independence, even though they may share the same process space.

Calls to the CSSM security API can originate in an application, in another add-in security module, or in CSSM itself. The CSSM dispatcher forwards all calls uniformly, regardless of their origin. CSSM ensures access to CSSM internal structures is serialized through thread synchronization primitives. If CSSM is implemented as a shared library then process synchronization primitives are also employed. Add-in modules need not have multi-threaded implementations to inter-operate with CSSM. Multi-threaded capabilities are registered with CSSM at module install time. Access to non-multithreaded add-ins is serialized by CSSM.

A common aspect of security services usage is the need for a convenient memory usage model. Often cryptographic operations and operations on certificates make pre-calculation of memory block sizes difficult and inefficient. CSSM rectifies this problem through registration of application memory allocation callback functions. CSSM and attached add-in modules use the applications memory functions to create complex or opaque objects. The CSSM_DATA buffer structure is used to track memory blocks allocated by CSSM components. Memory blocks returned to the application can be freed using the applications chosen free routine by freeing the CSSM_DATA buffer contents.

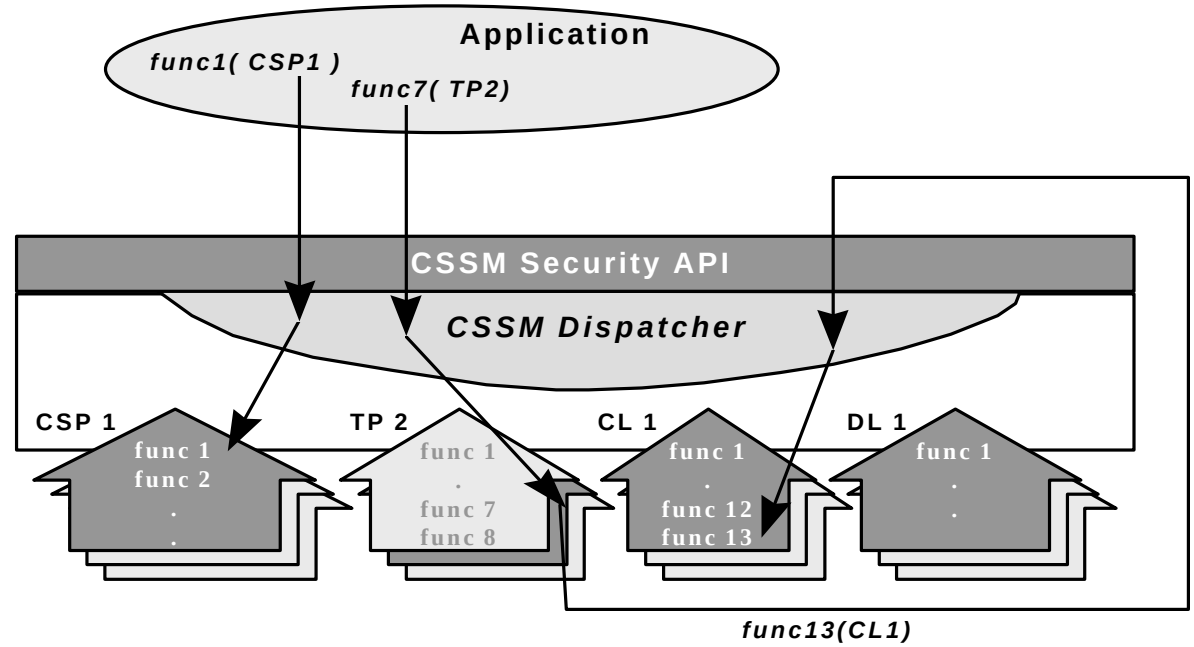

Figure 3. CSSM dispatches call to selected add-in security modules.

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
- Call-back functions for disk, screen, and keyboard I/O supported by the operating system
- A test-and-check function to ensure CSSM integrity
- Additional integrity services to test add-in module integrity and application authentication
- Management of concurrent security operations
- Auditable security functions

3. Cryptographic Services Manager

The CSSM infrastructure doesn't implement general purpose cryptography. It has been termed "crypto with a hole." The Cryptographic Services Manager provides applications with access to cryptographic functions that are implemented by Cryptographic Service Provider (CSP) modules. This centralizes all the cryptography into exchangeable modules.

The Cryptographic Services Manager defines two categories of service:

- Module management — installation, feature registration, and query of CSP features
- Selection, initialization, and use of cryptographic operations, implemented by a CSP

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

A CSP can perform one or more of these cryptographic functions:

- Bulk encryption algorithm
- Digital signature algorithm
- Cryptographic hash algorithm
- Unique identification number
- Random number generator
- Encrypted storage

The Cryptographic Services Manager doesn't assume any particular form factor for a CSP. CSPs can be instantiated in hardware, software or both. Operationally, the distinction must be transparent. The two visible distinctions between hardware and software implementations are the degree of trust the application receives by using a given CSP, and the cost of developing that CSP. A hardware implementation should be more tamper-resistant than a software implementation. Hence a higher level of trust is achieved by the application.

Cryptographic service providers, whose capabilities may change after installation, may make dynamic requests to update CSSM registration information. The dynamic nature of removable and software-loadable cryptographic service providers is supported by CDSA.

Software CSPs are convenient and portable. Software CSPs can be carried as an executable file on common forms of removable media. The components that implement a CSP must be digitally signed (to authenticate their origin and integrity), and they are highly tamper-resistant. This requirement extends to composite implementations involving both software and hardware. Multiple CSPs may be loaded and active within the CSSM at any time. A single application may use multiple CSPs concurrently. Interpreting the resulting level of trust and security is the responsibility of the application or the trust policy module used by the application.

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

The Cryptographic Services Manager defines a high-level, certificate-based API for cryptographic services to support application development. The Cryptographic Services Manager defines a lower-level Service

Provider Interface (SPI) that more closely resembles typical CSP APIs, and provides CSP developers with a single interface to support. A CSP may or may not support multi-threaded applications.

Intel has defined a CSP Service Provider Interface. The specification is available in the companion document titled *Common Security Services Manager Service Provider Interface Specification*

Acknowledging legacy CSPs, the CSSM architecture defines an optional adaptation layer between the Cryptographic Services Manager and a CSP. The adaptation layer allows the CSP vendor to implement a shim to map the CSSM SPI to the CSP's existing API and to implement any additional management functions that are required for the CSP to function as an add-in module in the extensible CSSM architecture. New CSPs may support the CSSM SPI directly (without the aid of an adaptation layer).

3.1 Cryptographic Service Provider Registration and Management

The Cryptographic Services Manager defines API calls for installation and registration of CSPs. CSSM manages a CSP registry that records each CSP's logical name, the information required to locate and dynamically initiate the CSP, and some minimal meta-data describing the algorithms implemented by the CSP. An application uses an *attach* operation to load and initiate a CSP.

When a CSP is loaded and initiated, it must present a digitally-signed credential, such as a certificate to identify its author and publisher. The signature represents the CSP provider's attestation of ownership and a guarantee that the CSP, with the CSP adaptation layer, conforms to the CSSM SPI specification. CSSM checks the authenticity of every CSP that is loaded on the local system.

When a CSP is loaded, it must register its services with the CSSM before its cryptographic services can be used by an application or add-in module. A CSP, or a proxy for the CSP (such as its Adaptation Layer), registers a set of callback functions with the CSSM. There is one callback function for each CSSM-defined cryptographic API call. The CSP may or may not implement all cryptographic calls defined by CSSM. Unimplemented functions are registered as null. The CSP may implement additional functions outside of the CSSM-defined API calls. The CSP may register a single callback function, and instruct applications and modules developers (via documentation) to activate these functions through the message-based, CSSM *pass-through* function.

When registering services, a CSP must present information describing the algorithms it implements and its secure key storage mechanisms, if any are provided. The Cryptographic Services Manager maintains this descriptive information in the CSP registration database. This infrastructure supports extensible cryptographic services. Any CSP implementing a subset of the CSSM Cryptographic API may make its services available to applications by installing its logical name and registering its functionality with the CSSM Cryptographic Services Manager. CSPs with dynamic capability may register with this flag set. CSSM will refresh information about the CSP whenever an application queries the CSPs registry. In this fashion an application can poll a CSP to become informed of a change in its status. It is anticipated that the form factor of some add-in modules will span SPI functional boundaries. For example, a smart card may also register as a data storage module that contains certificates and credentials in tamper resistant storage.

The Cryptographic Services Manager API allows an application to query the registry of installed (known) CSPs to determine the availability of cryptographic algorithms. When requesting cryptographic operations, the caller must select which CSP should be used. In some scenarios, the CSSM will restrict the set of CSP capabilities based on the credentials an application can present to the CSSM. Applications may request cryptographic operations directly through the CSSM-defined API for CSPs, and indirectly through the certificate-based services of another add-in module (such as a trust policy).

The indirection provided by CSSM enables applications designers to program to complex execution models, while CSP implementations may lag behind using much simpler execution models. For example, an application could attach to the same CSP multiple times with different threads of execution each time. Each thread would get the appearance of having exclusive access to the CSP. Meanwhile the CSP may be implemented according to a single threaded model. Additionally, the CSP may be managing multiple installed cards or multiple portable card slots on the system. An application may attach to the same CSP once for each card, as it looks like a different CSP, even though there is a single instance of the CSP attached to the CSSM.

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

CSPs may be detached. An application should not invoke this operation unless all requests to the target CSP have been completed. CSPs may also be uninstalled. This operation removes the CSP name and its associated attributes from the CSSM's CSP registry. Uninstall must be performed before a new version of the CSP is installed in the CSSM registry.

3.2 Cryptographic Services API

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

- SignData
- VerifyData
- DigestData
- EncryptData
- DecryptData
- GenerateKeyPair
- GenerateRandom
- WrapKey

Security-conscious applications use these high-level concepts to provide authentication, data integrity, data and communication privacy, and nonrepudiation of messages to the end-users.

The CSP may implement any algorithm. For example, CSPs may provide one or more of the following algorithms, in one or more modes:

- Bulk encryption algorithm: DES, Triple DES, IDEA, RC2, RC4, RC5, Blowfish, CAST
- Digital signature algorithm: RSA, DSS
- Key negotiation algorithm: Diffie/Helman
- Cryptographic hash algorithm: MD4, MD5, SHA
- Unique identification number: hard-coded or random-generated

- Random number generator: attended and unattended
- Encrypted storage: symmetric-keys, private-keys

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

- The algorithm to be used (such as RC5)
- Algorithm-specific parameters (such as key length)
- The object upon which the operation is conducted (such as filename)
- The cryptographic variables (such as the key)

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

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

The pass-through feature allows a highly knowledgeable application to call low-level CSP functions that are not available through to the common Cryptographic API. The Cryptographic Services Manager will either reject the call or pass it through to the selected CSP. The Cryptographic Services Manager will not alter the result of the request, or generate other side effects based on the request. The philosophy of CDSA and the numerous services provided by CSSM is to reduce the need for applications to work at this low level.

3.3 Additional CSP Services

Unique services. Application processes may use the unique cryptographic services provided by a CSP via a pass-through capability in the Cryptographic Services API. The parameters to the pass-through interface include a security context, a CSP-specific function name, and the arguments to the function. After determining the authorization for the call (based on the invoking process, the CSP selected by the security context, and the specific function requested), the call is passed through to the specified CSP. The application process is responsible for the correctness of the arguments supplied to the call. Invalid calls are trapped by the CSSM and cause a failed process.

Key management. Every CSP is responsible for implementing its own secure, persistent storage and management of private keys. To support chains of trust across application domains, CSPs must support importing and exporting both public and private keys. This means transferring keys among remote and possibly foreign systems. The ability to transfer keys assumes the ability to convert one key format into any other key format, and to secure the transfer of private and symmetric keys.

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

Note that each CSP will create and manage its own private-key database. If an application requires that more than one CSP perform operations using the same private key, then that key must be exported from

some source and imported to all CSPs needing to use it. Wrapping keys as key blobs manages the problem of different key formats among different CSPs. This assumes that the key length is acceptable to all CSPs using the same key.

Each CSP defines and implements its own key-management functions. Recent CSP implementations, such as Microsoft's Cryptographic API*, define internal storage formats and key-blob wrappers for exporting keys outside of the CSP. CSPs will exchange private keys through secured communication protocols (such as wrappers), rather than through access to a shared database for private keys.

The Cryptographic Services Manager API defines how private keys will be passed up and down through the layers of the CDSA, but it does not specify how private keys will be stored within the CSP.

CSPs that are used with key escrow protocols are not restricted by CDSA from communicating escrowed keys to escrow agents. Though some CSPs may not have escrow capabilities, a CSP under CDSA is not required to accommodate key escrow. Rather the exportation of private keys and other capabilities relevant to key escrow can be managed by CDSA. CDSA capabilities registration facilities permit escrow services to be implemented using CDSA.

CDSA CSPs may have native support for key recovery or non-native key recovery can be augmented through the adaptation layer at the service provider interface. CDSA is flexible enough to support multiple key recovery techniques. Implementations of key recovery may involve a service component above the CSSM layer as well as extending existing modules or creating new modules beneath the CSSM layer.

3.4 Security Context Manager

The Security Context Manager creates, initializes, and maintains concurrent security contexts. A security context is a run-time structure containing the secrets and the security-related execution parameters of an application process or thread.

Once cryptographic contexts have been created the application may freely use those contexts without CSSM-imposed security checks. Security contexts may contain actual encryption keys, passphrases and passphrase functions. Applications are responsible for protecting these secrets. Applications desiring maximal protection should use passphrase callback functions that limit the duration in which the passphrase is present in the system.

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

A knowledgeable application initializes the security context structure with values obtained by querying the Cryptographic Services Manager, as described in Section 3. Using the information returned by these queries, an application may use the Security Context Manager to create and initialize one or more contexts.

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

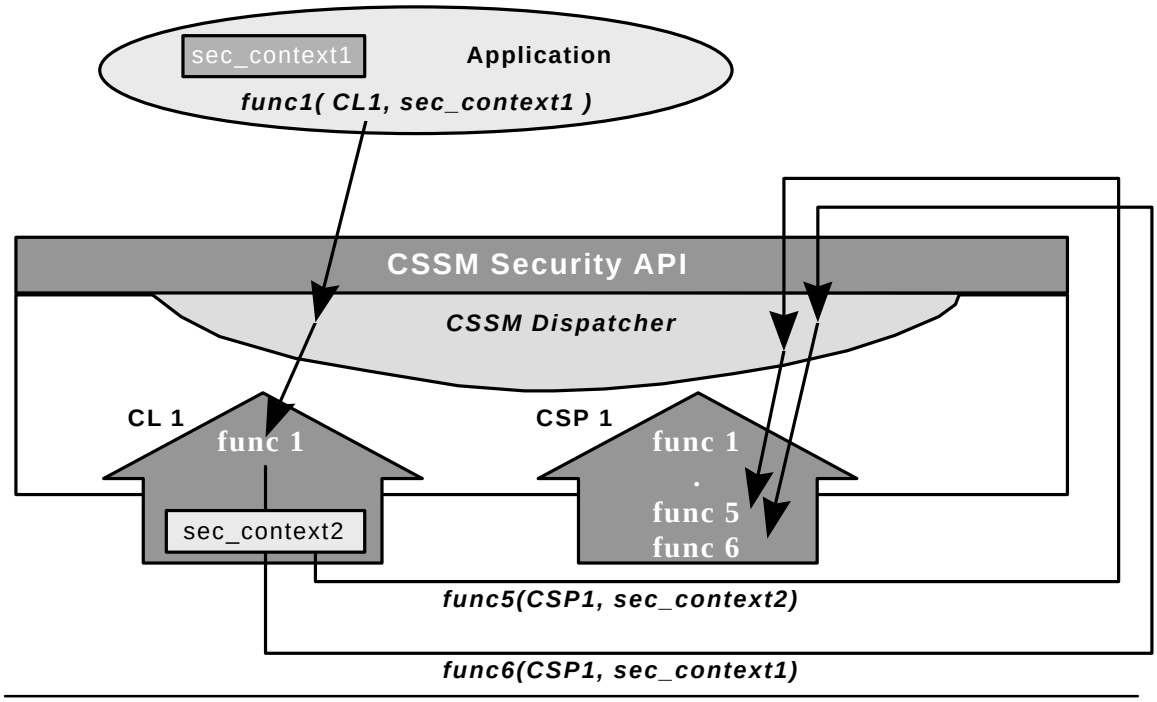

Figure 4. Indirect creation of a security context.

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

Security context management provides mechanisms that:

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

4. Trust Policy Services Manager

A digital certificate binds an identification in a particular domain to a public key. When a certificate is issued (created and signed) by the owner and authority of a domain, the binding between key and identity is attested by the digital signature on the certificate. The issuing authority also associates a level of trust with the certificate. The actions of the user, whose identity is bound to the certificate, are constrained by the 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

How applications interpret and manipulate the contents of certificates to achieve these ends is defined by the real world trust model the application has chosen as its model for trust and security.

The primary purpose of a Trust Policy (TP) module is to answer the question "Is this certificate trusted for this action?" The CSSM Trust Policy API defines the generic operations that should be defined for certificate-based trust in every application domain. The specific semantics of each operation is defined by the

- Application domain
- Trust model
- Policy statement for a domain
- Certificate type
- Real-world operation the user requests within the application domain

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

As an infrastructure, 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.

Policy describing the intended use of security objects such as private keys, credentials and certificates requires flexible mechanisms. . The CDSA trust policy module can support trust policy interpreters such as PolicyMakers AKWARD, safe-JAVA and safe-TLC. The implementation of a policy interpreter my rely heavily on the CL and DL modules for certificate parsing and policy object storage.

Trust policy statements can also be expressed concisely as action tags which assert actions a principal is authorized to perform. Fixed trust policy action assertions enable simpler expression of trust policy which makes policy inspections straightforward. These actions are primary operations on the basic objects common to most all trust models. These include certificates, credentials and certificate revocation lists. The basic operations on certificates 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 only needs 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 sure that applications using their module(s) will 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 and 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 Services Manager API defines two categories of operation:

- Installation, dynamic loading, attribute registration, and attribute query of TPs
- Selection and use of TPs by applications

4.1 Trust Policy Registration and Management

The Trust Policy Services 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 module is loaded and initiated, it must present a digitally-signed credential, such as a certificate to identify its author and publisher. The signature represents the module provider's attestation of ownership and a guarantee that the trust policy module conforms to the CSSM Trust Policy Interface (TPI) specification. CSSM will check the authenticity of each TP code that is loaded on the local system.

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. Unimplemented functions are registered as null. The CSSM will not restrict access to TP functionality based on caller credentials. It is understood that TP modules facilitate reference monitor applications in making policy based decisions. They do not attempt to enforce policy upon the caller. 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 provided by these functions. CSSM does not require nor enforce the availability of run-time query support for extended functions.

The Trust Policy Services Manager API allows an application to query the registry of installed (known) trust policy modules to determine their availability.

Trust policy modules may be detached. An application should not invoke this operation unless all requests to the target module have been completed. Trust policy modules may also be uninstalled. This operation removes the trust policy's logical name and its associated attributes from the CSSM's trust policy registry. Uninstall must be performed before a new version of a trust policy module is installed in the CSSM registry.

4.2 Trust Policy Services API

CSSM defines eight API calls that all trust policies should implement at a minimum. Six functions define operations on the fundamental CSSM object types of certificate and certificate revocation list. The other two functions are used by the TP module to extend the semantics of its policy to the application. The extending functions are:

- CSSM_TP_CertVerifyForAction ( ) — Accepts as input from the application a certificate and a domain-specific action. The TP module must determine whether or not the certificate is trusted to perform the domain-specific action.
- CSSM_TP_PassThrough ( ) — 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. Such operations may include queries or services (outside of verify) that are specific to the domain represented by the TP module. A results parameter of arbitrary type is used to return function results.

The six remaining functions define the three operations, sign, verify, and revoke on certificates and certificate revocation lists. In general, the TP module must determine whether or not the presented certificate is trusted to perform the action of sign, verify, or revoke on another certificate or on a certificate revocation list. The following descriptions present general, recommended semantics. The specific semantics implemented by the TP module is defined by the specific trust model it represents.

Signing Certificates and Certificate Revocation Lists. Every system should be capable of being a Certificate Authority (CA), if authorized to do so. CAs are applications that issue and validate certificates and certificate revocation lists (CRLs). Issuing certificates and CRLs includes 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 for performing this operation. Some policies may require that a newly-issued certificate be signed by multiple authorities.

The trustworthiness of each signer's certificate must be evaluated and accepted or rejected by the trust policy module. If the TP trusts the signer's certificate, then the TP module may perform the cryptographic signing algorithm by invoking the signing function in a certificate library module, or by directly invoking the data signing function in a CSP module.

Verifying Certificates and Certificate Revocation Lists. The TP module must determine the general trustworthiness of a certificate. This evaluation is performed without regard to the operation that might be performed under the auspices of the certificate. This is a general verification of trust in a certificate. The TP modules must also determine the trustworthiness of a certificate revocation list that was 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, then the TP module should use 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 as 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, then the TP module should carry out the operation directly by invoking a certificate library module to add a new revocation record to a CRL, mark the certificate as revoked, or both. The TP API also defines a reason parameter (in the spirit of X.509 revocation reasons) that is passed to the TP module. The TP may use the presented reason as part of its trust evaluation.

5. Certificate Library Services Manager

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

As new standard formats are defined and accepted by the industry, certificate library modules will be defined and implemented by industry members and used directly and indirectly by many applications. Certificate library modules perform syntactic manipulations of certificate and CRL data objects. The semantic interpretation of certificate and revocation security objects are considered policy transformations are implemented as trust policy modules.

Certificate library modules manipulate memory-based objects only. The persistence of these objects is an independent property. It is the responsibility of the application and/or the trust policy module to use data storage add-in modules to make these objects persistent (if appropriate). The storage mechanism used by a data storage module may be independent of other modules.

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

The Certificate Library Services Manager API defines two categories of operation:

- Installation, dynamic loading, attribute registration, and attribute query of CLs
- Selection and use of CLs by applications

5.1 Certificate Library Registration and Management

The Certificate Library Services Manager defines API calls for installing and registering CL modules. CSSM manages a certificate library registry that records each certificate module'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 module is loaded and initiated, it must present a digitally-signed credential, such as a certificate to identify its author and publisher. The signature represents the module provider's attestation of ownership and a guarantee that the certificate library module conforms to the CSSM Certificate Library Interface (CLI) specification. CSSM will check the authenticity of each CL code that is loaded on the local system.

When a certificate library module is loaded, it must register its services with the CSSM before it can be used by an application. A CL module registers a set of callback functions with the CSSM. There is one callback function for each CSSM-defined certificate library API call. The module may or may not implement all certificate library calls defined by CSSM. Unimplemented functions are registered as null.

The certificate library 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 Certificate Library Services Manager. Applications access these functions through the CSSM *pass-through* function. The certificate library module must document the features and services provided by these functions. CSSM does not require, nor enforce, the availability of run-time query support for extended functions.

The Certificate Library Services Manager API allows an application to query the registry of installed (known) certificate library modules to determine their availability.

Certificate library modules may be detached. An application should not invoke this operation unless all requests to the target module have been completed. Certificate library modules may also be uninstalled. This operation removes the certificate module's logical name and its associated attributes from the CSSM's certificate library registry. Uninstall must be performed before a new version of a certificate library module is installed in the CSSM registry.

5.2 Certificate Library Services API

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

- Creating new certificates and new CRLs
- Signing existing certificates and existing CRLs
- Viewing certificates
- Verifying certificates and CRLs
- Extracting values (for example, public keys) from certificates
- Importing and exporting certificates of other data formats
- Revoking certificates
- Reinstating revoked certificates
- Searching certificate revocation lists
- Pass-through for unique, format-specific certificate and CRL operations

Every certificate library (CL) module should implement most if not all of these functions. A *pass-through* function is also defined by the certificate library API. This allows CLs to provide additional functionality if required to manipulate the certificate data format and CRL data format supported by the module.

The following is a brief description of the CSSM-recommended semantics of certificate library functions. The data format dependent manipulation of certificates and CRLs is implemented by the CL module developer.

Creating new Certificates and new CRLs. The CL creates a memory-resident certificate containing values specified by the caller. The new certificate is not an official certificate; it is not signed as a result of using this operation. The signing function provided by the CL module can be used to sign a memory-resident certificate. The CL creates an empty, memory-resident CRL. Revocation records can be added to the CRL using the CL module's revocation function.

Signing Certificates and CRLs. The CL computes the digital signature over a certificate or a CRL and includes the newly-generated signature in the memory-resident copy of the certificate or CRL. CL modules may forward signing requests to external signing authorities or perform them locally. The CL module may use the services of a CSP add-in module to calculate the signature. Many certificate formats define fields that must be excluded from the signature calculation.

For example, management fields whose state must change over time without invalidating the certificate, cannot be included in the signature calculation. The CL module uses its knowledge of the certificate data format to include only those certificate fields that must not change during the life of the certificate. The signing function may be called repeatedly on a single certificate if the certificate data format supports multiple signatures. Signing memory-resident CRLs is performed in the same manner.

Viewing Certificates. Certificates are the user-recognized credentials for secure and authorized operations performed by a client process on the user's behalf or by a server process in response to the user's request. For this reason, having a viewable representation for a certificate is essential to system usability. Also, an electronic identification card, which is a specific type of credential, should always include a photograph of the owner and be viewable. Certificates must be able to serve all the current functions of traditional paper-based credentials.

The process of viewing a certificate is data format-specific. If appropriate, the CL module can take control of the viewing device (assuming a real display exists). Otherwise the CL module can extract a copy of all viewable fields from a memory-resident certificate and return those fields to the caller with a template describing the format of the returned data fields. The caller may then present the returned values in the most appropriate manner, such as creating a graphical representation for an end-user system or generating text to a report file for a server system.

Verifying Certificates and CRLs. Mechanically verifying one or more signatures associated with a certificate or CRL depends on the format of the signed objects. The CL module embodies knowledge of which subset of the object's fields were included in the signing process. Regardless of data format, every CL module must test the integrity of the signature. This means that the object associated with that signature has not been modified since the signature was calculated.

Typically the CL module will invoke a CSP to recalculate the signature and then compare that result with the signature under verification. Depending on the fields stored in the certificate or the CRL, additional checks may be required to complete syntactic verification.

Extracting Values (such as Public Keys) from Certificates. Applications and trust policy modules may need selected values from a certificate. Extracting field values depends on the certificate data format. The CL module must extract and return, to the caller, any requested field value.

Importing and Exporting Certificates. When presenting a certificate to an application or sending a certificate from one system to another, a data format translation is often required. A full-service CL module should provide format translation functions that import certificates of foreign format to the library's particular format. Also the reverse translation should be provided.

These import and export functions allow applications to more easily accept certificates in one of several distinct formats by converting the foreign format to the format typically processed by the application. CRLs are typically stored only in the CL module's native format. It is assumed that CRLs are exchanged only among systems that support the same CRL format.

Revoking and Reinstating Certificates. A certificate may be permanently or temporarily revoked for a number of reasons. Some certificate formats include one or more revocation status fields. In this case, the CL module will mark the certificate as revoked. When revoking a certificate, the CL module will typically add a revocation record to the supplied CRL.

To ensure the integrity of the revocation record, it should be digitally signed, using the private key associated with the revoking agent's certificate. To reinstate a temporarily revoked certificate, the revocation record must be removed from the CRL. If fields in the certificate itself were modified to indicate the revoked state, these certificate values must also be updated.

Searching Certificate Revocation Lists. Certificate revocations must be reported to all systems that may receive that certificate as a security credential. To avoid constant online revocation checks, CRLs are distributed periodically to all systems that need to verify certificates that may be contained in the revocation list. When a user presents a certificate to an application, the application must verify that certificate.

Certificate verification (by a CL module) includes a check to ensure the certificate in not revoked. This test requires a search of all CRLs that may contain the certificate in question. A CL module must support searches over a CRL, selecting revocation records based on selection criteria appropriate to the CRL data format.

6. Data Storage Library Services Manager

The primary purpose of a Data Storage Library (DL) module is to provide secure, persistent storage, retrieval, and recovery of security-relevant data objects such as certificates, certificate revocation lists (CRLs), keys and policy assertions. The persistence of these generic trust objects is independent of the memory-based manipulations performed by certificate library modules. DL modules may be invoked by applications, trust policy modules, or certificate library modules that make decisions about the persistence of these objects.

A single DL module may be semantically coupled to a Certificate Library (CL) module or may independently manage the persistence of opaque objects. A data storage library that is coupled with a certificate library module may use information obtained from the CL in the implementation of a physical data storage and retrieval model. For example, a DL might interrogate a certificate to construct/use indexes to speed data retrieval.

Each DL module can manage any number of independent, physical data stores. Each data store must have a logical name used by callers to refer to the persistent data store. Implementation of the DL module may use local file system facilities, commercial database management systems, custom stable storage devices and remote storage facilities. Properties and capabilities of each storage device are managed by the DL manager. Applications may hold multiple references to a single storage device. The DL manages open references as context handles to internal data structures and physical media. DL modules are not restricted from using caching or other performance optimization techniques. Processes and threads may access common physical storage devices, however device handles are unique and non-specific.

A DL module is responsible for the integrity of the objects it stores. If the DL module uses an underlying commercial database management system (DBMS), it may choose to further secure the data store by leveraging integrity services provided by the DBMS. DL module designers must choose which mechanisms best address the availability, integrity, privacy and performance needs of the perceived customer. For example tamper-resistant storage devices and encryption could be used to protect secret objects. Local and remote redundant storage devices could facilitate integrity and availability, while caching could improve performance.

6.1 Data Storage Library Registration and Management

The Data Storage Library Services Manager defines API calls for installing and registering of DL modules. CSSM manages a data storage library registry that records each data storage module's logical name and the information required to locate and dynamically initiate the module. (The logical name of the data storage library must not be confused with the logical name for each data store managed by the DL module.) Meta-information about the DL module capabilities and traits enable applications to select a DL module appropriate for their needs. For example, a DL module built on top of an X.500 directory service may indicate different naming and usage semantics than for file system-based storage. Other capabilities such as structured query language support, removable media and latent operation, are also traits registered with the DL module.

An application uses an *attach* operation to load and initiate a DL module. The module executable may be local or remote.

When a module is loaded and initiated, it must present a digitally-signed credential, such as a certificate to identify its author and publisher. The signature represents the module provider's attestation of ownership and a guarantee that the data storage library module conforms to the CSSM Data Storage Library Interface (DLI) specification. CSSM will check the authenticity of each DL code that is loaded on

the local system. The integrity checking protocol reduces the likelihood that virus infected modules will be loaded into the CSSM system.

When a data storage library module is loaded, it must register its services with the CSSM before it can be used by an application. A DL module registers a set of callback functions with the CSSM. There is one callback function for each CSSM-defined data storage library API call. The module may or may not implement all data storage library calls defined by CSSM. Unimplemented functions are registered as null.

The data storage library 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 Data Storage Library Services Manager. Applications access these functions through the CSSM *pass-through* function. The data storage library vendors must document the features and services provided by the module.

DL modules may have capabilities that change after installation and registration. For example remote or removable data stores' capabilities may change during regular operation of the DL module. In this case, the DL module cannot know what functions it supports until the application requests capabilities of a particular data store. In this fashion applications can poll add-in module capabilities.

The Data Storage Library Services Manager API allows an application to query the registry of installed (known) data storage library modules to determine their availability and capabilities. The DL modules themselves must support queries requesting the logical names of all data stores managed by the DL module. This list is dynamic (as data stores are created and deleted).

DL modules have associated with them semantic typing information specific to an object class. Semantic information is used to describe to CSSM and applications the object's intended use. For example certificate objects whose corresponding private key is known to CSSM would receive the semantic label of "owned". Other certificates may be self-signed and therefore endpoints of a signature chain or mesh and would receive the label of "root". Applications, CL and TP modules may use the semantic information when manipulating and evaluating objects that are semantically related.

Data storage library modules may be detached. An application should not invoke this operation unless all requests to the target module have been completed. Data storage library modules may also be uninstalled. This operation removes the data storage module's logical name and its associated attributes from the CSSM's data storage library registry. Uninstall must be performed before a new version of a data storage library module is installed in the CSSM registry.

6.2 Data Storage Library Services API

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

- Data store management functions
- Persistence operations

The data store management functions operate on a data store as a single unit. These operations include:

- Opening and closing data stores
- Creating and deleting data stores
- Importing and exporting data stores

The persistence operations on data stores include:

- Inserting
- Updating
- Deleting
- Retrieving
- Module-specific operations

A data store may contain a single object type or multiple object types. DL modules will register which object types the DL is capable of storing at installation time or whenever an application polls for capabilities information.

Creating and Deleting Data Stores. The DL module creates a new, empty data store and opens it for future access by the caller. An existing data store may be deleted. Deletion discards all data contained in the data store. Deletion will not occur if there are outstanding open references to a data store. Data store creation also involves specifying an access schema and setting other configuration information. This includes describing indexed fields and fields requiring unique database keys.

Opening and Closing Data Stores. The DL module manages the mapping of logical data store names to physical storage mechanisms. The caller uses logical names to reference persistent data stores. The open operation initializes physical storage mechanisms and associates a context to the logical storage facility. The close operation terminates current access to the data store and cleans up any temporal state created during initialization and operation.

Importing and Exporting Data Stores. Local data stores may be moved from one system to another or from one storage medium to another storage medium. The import and export operations support the transfer of an entire data store. The export operation prepares a snapshot of a data store. (Export does not delete the data store it snapshots.)

The import operation accepts a snapshot (generated by the export operation) and includes it as a new data store managed by the DL module. The imported data store is assigned a new logical name, which is then known by the local DL module.

The physical format type identifier of the exported image is included in the DL module capabilities information. Applications may choose the export format from among the registered export format types. For import, the application may select from the set of supported import format types. A format type identifier of “CUSTOM” is used to indicate a proprietary format defined by the DL implementer. Exclusion of any registered format identifier indicates the DL module does not support import/export capabilities.

The following is a brief description of the CSSM-recommended semantics of data storage library functions. The persistence mechanisms are implemented by the DL module developer.

Inserting Objects. The DL module adds a persistent copy of the supplied object to an open data store. This operation may include updating index entries or other components of the physical data model. The mechanisms used to store and retrieve persistent objects is specific to the implementation of the DL module; transparent to applications.

Updating Objects. The DL module updates objects when the inserted object already exists in the data store and the meta-data indicates unique key space. In the case of non-unique key spaces, the inserted object is appended to the data store. It is anticipated that applications will store and retrieve multiple objects using the same key. For example an application may associate several identity certificates, several policy objects and possibly a key object with a user name. A query for the user name would result in all

the user's objects being returned. Updates to these objects must be done by deleting and then re-adding to the data store.

Deleting Objects. The DL module removes the specified objects from the data store. If the objects are not found, then the operation returns an appropriate result condition.

Retrieving Objects. Applications and add-in security modules need to search persistent data stores for objects specified in the query. The DL module must provide a search mechanism for selectively retrieving a copy of these persistent objects. Selection is based on the selection criteria of a selection predicate. Selection predicates may be expressed as a string of structured language or as data structures of a query tree. As part of DL module registration the capability for processing structured language query predicates will be indicated. The CSSM enumerates the class of languages supportable by DL modules.

7. Integrity Services Manager

The Integrity Services Manager provides test-and-check services for the binary components of CSSM. The CSSM software components are digitally signed off-line by a companion signing tool. Signature verification will authenticate the manufacturer as the author and publisher of the binary object and determine whether or not the CSSM object was modified after it was signed. Periodic, runtime re-checks may be performed to verify constancy of the CSSM code segment. If tampering is detected in any component of CSSM, execution is interrupted.

7.1 Add-in Module Integrity Verification

CDSA checks the integrity of modules as they are dynamically bound into the system. Modules are signed by manufacturers. A manifest of the module and signature verification operations are reliably bound to add-in modules. At add-in installation, the manifest is presented to CSSM for verification and CSSM presents its manifest to the add-in module for reciprocal verification. Integrity of CSSM and add-in module binary images are cross-checked dynamically in this fashion.

Roles can be enforced during dynamic binding. The add-in module requesting installation will include with its manifest, signed credentials from the manufacturer indicating that the CSP enforces constraints described by the roles mechanism. The CSSM verifies these credentials and registers capabilities appropriate for the role.

7.2 Audit Log

An audit trail of executed security operations is a key contributor to the integrity of any security system. CSSM defines an audit facility. The audit mechanism must persistently store all relevant information concerning a security event. Dynamic/run-time selection of what information to capture must be supported.

An audit trail of security-related events helps track how security functions are used on a system, what security functions are not used, if and when a system is attacked, and whether or not the attacker was successful. All security-relevant events are auditable, including:

- CSSM API calls
- Loading an add-in security module
- Inserting or removing a hardware CSP

Many platforms and operating systems provide high-performance audit utilities for creating, managing, and profiling audit logs. It is desirable to use these facilities on each respective platform, but often these facilities were not developed with security as a requirement. Each platform’s native audit mechanism must therefore be evaluated with respect to function and security of the audit information stored.

It is also highly desirable that the audit mechanism provide user-level, post-audit, analysis tools that summarize and display the audit records in a usable form. If such an application is not provided as part of the native audit mechanism it should be possible to develop one.

8. System Security Services

The System Security Services layer is the appropriate architectural layer for defining and implementing sophisticated security protocols, based on the security services of the CSSM and its add-in modules. These services and protocols may include:

- Secure and private file systems (such as PFP secured files)
- Protocols for secure electronic commerce (such as JEPI and SET)
- Protocols for private communication (such as SHTTP, SSL, PGP and S/MIME)
- Multi language access to the CSSM API (such as CSSM-Java API)
- CSSM management tools (such as a CSSM installation and configuration tool)

9. Glossary

Asymmetric algorithms	Cryptographic algorithms where one key is used to encrypt, and a second key is used to decrypt. They are often called public-key algorithms. One key is called the public key, and the other is called the private key or secret key. RSA (Rivest-Shamir-Adelman) is the most commonly used public-key algorithm. It can be used for encryption and for signing.
Certificate Authority (CA)	An entity that guarantees or sponsors a certificate. For example, a credit card company signs a cardholder's certificate to assure that the cardholder is who he or she claims to be. The credit card company is a certificate authority. Certificate authorities issue, verify, and revoke certificates.
Certificate	See Digital certificate.
Certificate chain	The hierarchical chain of all the other certificates used to sign the current certificate. This includes the Certificate Authority (CA) who signs the certificate, the CA who signed that CA's certificate, and so on. There is no limit to the depth of the certificate chain.
Certificate signing	The Certificate Authority (CA) can sign certificates it issues or cosign certificates issued by another CA. In a general signing model, an object signs an arbitrary set of one or more objects. Hence, any number of signers can attest to an arbitrary set of objects. The arbitrary objects could be, for example, pieces of a document for libraries of executable code.
Certificate validity date	A start date and a stop date for the validity of the certificate. If a certificate expires, the Certificate Authority (CA) may issue a new certificate.
Cryptographic Service Providers (CSPs)	Modules that provide secure key storage and cryptographic functions. The modules may be software only or hardware with software drivers. The cryptographic functions provided may include: • Bulk encryption and decryption • Digital signing • Cryptographic hash • Random number generation • Key exchange
Common Data Security Architecture (CDSA)	A set of layered security services that address communications and data security problems in the emerging PC business space. The CDSA consists of three basic layers: • A set of system security services • The Common Security Services Manager (CSSM) • Add-in Security Modules (CSPs, TPs, CLs, DLs)

Common Security Services Manager (CSSM)	The central layer of the Common Data Security Architecture (CDSA) that defines six key service components: • Cryptographic Services Manager • Trust Policy Services Manager • Certificate Library Services Manager • Data Storage Library Services Manager • Integrity Services Manager • Security Context Manager The CSSM binds together all the security services required by PC applications. In particular, it facilitates linking digital certificates to cryptographic actions and trust protocols.
Cryptographic algorithm	A method or defined mathematical process for implementing a cryptography operation. A cryptographic algorithm may specify the procedure for encrypting and decrypting a byte stream, digitally signing an object, computing the hash of an object, generating a random number, etc. CSSM accommodates DES, RC2, RC4, IDEA and other encryption algorithms.
Cryptoki	Short for cryptographic token interface. See Token.
Digital certificate	The binding of some identification to a public key in a particular domain, as attested to directly or indirectly by the digital signature of the owner of that domain. A digital certificate is an unforgeable credential in cyberspace. The certificate is issued by a trusted authority, covered by that party's digital signature. The certificate may attest to the certificate holder's identity, or may authorize certain actions by the certificate holder. A certificate may include multiple signatures and may attest to multiple objects or multiple actions.
Digital signature	A data block that was created by applying a cryptographic signing algorithm to some other data using a secret key. Digital signatures may be used to: • Authenticate the source of a message, data, or document • Verify that the contents of a message hasn't been modified since it was signed by the sender • Verify that a public key belongs to a particular person Typical digital signing algorithms include MD5 with RSA encryption, and DSS, the proposed Digital Signature Standard defined as part of the U.S. Government Capstone project.
Hash algorithm	A cryptographic algorithm used to hash a variable-size input stream into a unique, fixed-sized output value. Hashing is typically used in digital signing algorithms. Example hash algorithms include MD and MD2 from RSA Data Security. MD5, also from RSA Data Security, hashes a variable-size input stream into a 128 bit output value. SHA, a Secure Hash Algorithm published by the U.S. Government, produces a 160-bit hash value from a variable-size input stream.

Key Management	(*See* Key Escrow *and* Key Recovery)
Key Escrow	A copy of the user's private key is stored with an escrow agent. Symmetric keys used for bulk data encryption are wrapped with the recipient's public key. If the recipient loses the private key, the escrow copy could be retrieved from an escrow agent.
Key Recovery	One or more trusted key recovery agents are identified by message sender and receiver. The symmetric key(s) used during bulk data transfer are encrypted with the agent's public key. The wrapped symmetric key(s) are embedded in the encrypted message. When the symmetric key needs to be recovered, the message containing the embedded symmetric key(s) is sent to one of the key recovery agents.
Leaf Certificate	The certificate in a certificate chain that has not been used to sign another certificate in that chain. The leaf certificate is signed directly or transitively by all other certificates in the chain.
Meta-information	Descriptive information specified by an add-in service module and stored in the CSSM registry. This information advertises the add-in modules services. CSSM supports application queries for this information. The information my change at runtime.
Message digest	The digital fingerprint of an input stream. A cryptographic hash function is applied to an input message arbitrary length and returns a fixed-size output, which is called the digest value.
Nonce	A sequence of random bits.
Owned certificate	A certificate whose associated secret or private key resides in a local CSP. Digital-signing algorithms require using owned certificates when signing data for purposes of authentication and non-repudiation. A system may use certificates it does not own for purposes other than signing.
Private key	The cryptographic key used to decipher messages in public-key cryptography. This key is kept secret by its owner.
Public key	The cryptographic key used to encrypt messages in public-key cryptography. The public key is available to multiple users (for example, the public).
Random number generators	A function that generates cryptographically strong random numbers that cannot be easily guessed by an attacker. Random numbers are often used to generate session keys.
Root certificate	The prime certificate, such as the official certificate of a corporation or government entity. The root certificate is positioned at the top of the certificate hierarchy in its domain, and it guarantees the other certificates in its certificate chain. Each Certificate Authority has a self-signed root certificate. The root certificate's public key is the foundation of signature verification in its domain.

Secret key	See Private key.
Security Context	A control structure that retains state information shared between a cryptographic service provider and the application agent requesting service from the CSP. Only one context can be active for an application at any given time, but the application is free to switch among contexts at will, or as required. A security context specifies CSP and application-specific values, such as required key length and desired hash functions.
Security-relevant event	An event where a CSP-provided function is performed, an add-in security module is loaded, or a breach of system security is detected.
Session key	A cryptographic key used to encrypt and decrypt data. The key is shared by two or more communicating parties, who use the key to ensure privacy of the exchanged data.
Signature	See Digital signature.
Signature chain	The hierarchical chain of signers, from the root certificate to the leaf certificate, in a certificate chain.
Symmetric algorithms	Cryptographic algorithms that use a single secret key for encryption and decryption. Both the sender and receiver must know the secret key. Well-known symmetric functions include DES (Data Encryption Standard) and IDEA. DES was endorsed by the U.S. Government as a standard in 1977. It's an encryption block cipher that operates on 64-bit blocks with a 56-bit key. It is designed to be implemented in hardware, and works well for bulk encryption. IDEA (International Data Encryption Algorithm), one of the best known public algorithms, uses a 128-bit key.
Token	The logical view of a cryptographic device, as defined by a CSP's interface. A token can be hardware, a physical object, or software. A token contains information about its owner in digital form, and about the services it provides for electronic-commerce and other communication applications. A token is a secure device. It may provide a limited or a broad range of cryptographic functions. Examples of hardware tokens are SmartCards* and PMCIA cards.
Verification	The process of comparing two message digests. One message digest is generated by the message sender and included in the message. The message recipient computes the digest again. If the message digests are exactly the same, it shows or proves there was no tampering of the message contents by a third party (between the sender and the receiver).
Web of trust	A trust network among people who know and communicate with each other. Digital certificates are used to represent entities in the web of trust. Any pair of entities can determine the extent of trust between the two, based on their relationship in the web. Based on the trust level, secret keys may be shared and used to encrypt and decrypt all messages exchanged between the two parties. Encrypted exchanges are private, trusted communications.