Global Information Assurance Certification Paper

Common Data Security Architecture – (CDSA)
For
SANS GIAC
Level One Security Essentials
GSEC Practical Assignment

Document Version	**1.0**
Document Version Date	**1st March, 2001**
Document By	**Michael Yeung**

TABLE OF CONTENTS

Introduction

Common Data Security Architecture (CDSA) is an open and extensible software framework that addresses security requirements of applications such as e-commerce, communication, and digital content distribution.

This paper aims to provide a general understanding of CDSA and to describe how CDSA addresses the three basic concerns of Internet security:

1) *Authenticity*
   The ability to verify the parties on both ends of a communication link
2) *Integrity*
   The ability to verify that all data transmitted and received has not been tampered with or changed
3) *Confidentiality*
   The ability to ensure that all transmitted data over a communication link cannot be read by unintended parties

In order to achieve these objectives, this paper will address the following questions:

- What is CDSA?
- How is CDSA being used? (two scenarios)
- How CDSA addresses Authenticity, Integrity, and Confidentiality?
- What are the possible future trends of CDSA?

1.0 What is CDSA?

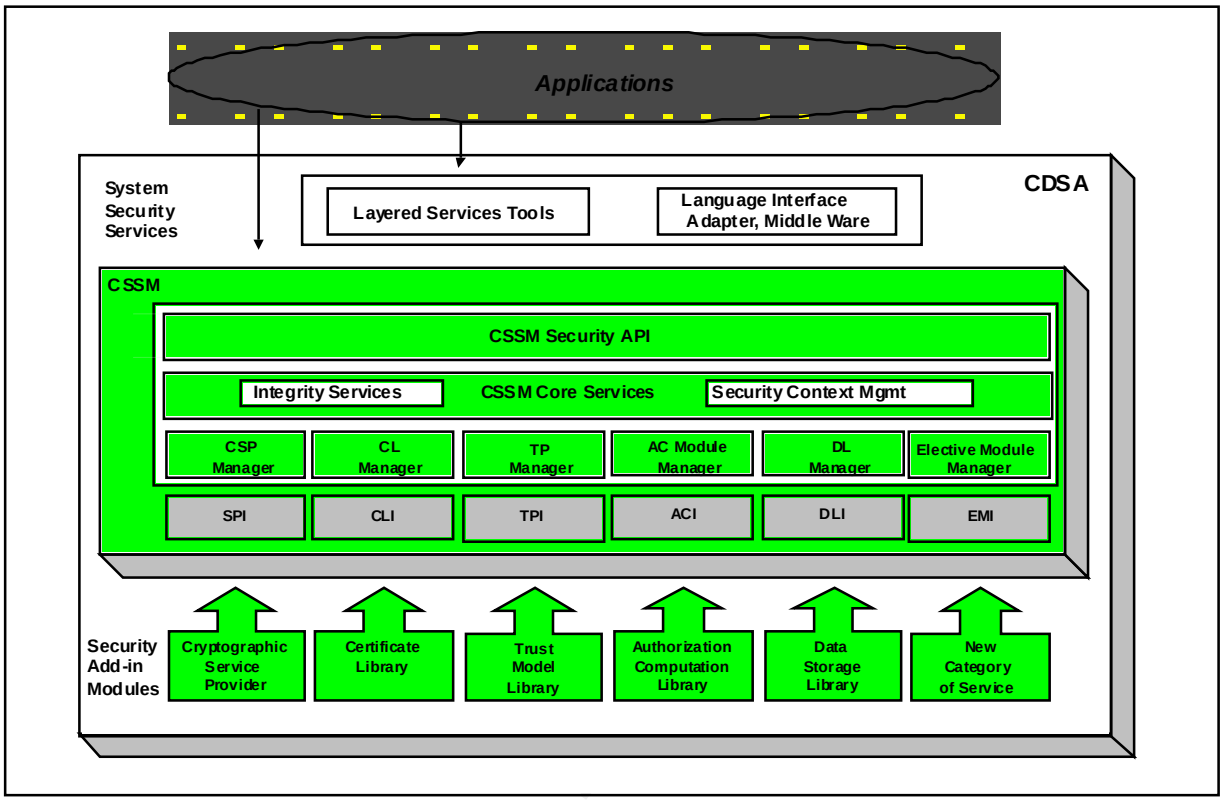
CDSA is an open and extensible software framework and API specification that address communication and data security requirements. CDSA was originally developed by Intel Architecture Lab (IAL). The specification was refined through the Open Group standards process with companies such as Hewlett-Packard, IBM, JP Morgan, Motorola, Netscape, Trusted Information Systems, and Shell Companies. CDSA was adopted by the Open Group in 1977 as an open specification. The objectives of CDSA are:

1. Encourage interoperable, horizontal security standards
2. Offer essential components of security capability to the industry at large

In order to achieve these objectives, CDSA was designed with the following architectural principals:

1. *A layered service provider model*
   CDSA consists of a set of horizontal layers of services. Each layer builds on services provided by the layer below it.
2. *Open Architecture*
   CDSA is fully disclosed for peer review and went through the Open Group Standards Process
3. *Modularity and Extensibility*
   Each layer is made up of individual modules. New modules can be added as needed and therefore support extensibility

The overview of CDSA is illustrated as follows:



The diagram shows that CDSA is made up of three basic layers:

- System Security Services
- The Common Security Services Manager (CSSM)
- Security Add-in Modules

1.1 System Security Services

System Security Services are between applications and CSSM services. Software at this layer provides a high-level abstraction of security services such as secure e-mail, secure file systems, or secure communications. Applications can invoke the CSSM APIs directly, or use these layered services to access security services on a platform.

1.2 The Common Security Service Manager (CSSM)

CSSM provides a set of core services that are common to all categories of security services. CSSM defines five basic categories of services:

Cryptographic Service Provider (CSP) modules
CSPs perform cryptographic operations such as bulk encrypting, digesting, and digital signatures.

Trust Policy (TP) modules
TPs implement policies defined by authorities and institutions and set the level of trust required to carry out specific actions (such as issuing a check or gaining access to confidential intellectual property).

Certificate Library (CL) modules
CLs manage certificates and revocation lists, and access to remote signing capabilities such as Certification Authorities (CA).

Data Storage Library (DL) modules
DLs provide stable storage for security-related data objects, including certificates cryptographic keys and policy objects.

Authorization Computation (AC) modules
ACs define a general authorisation evaluation service that computes whether a set of credentials and samples are authorized to perform a specific operation on a specific object.

Elective (EM) Modules
EMs add new and compelling security features not encompassed by the current set of service modules. For example, one new feature that vendors might add to CDSA is biometrics authentication.

In addition, CSSM provides two additional core services:

Integrity Services
The integrity services are used by CSSM itself to verify and guarantee the integrity of all the other components within the CSSM environment

Security Context Management
CSSM provides context management functions (such as session information) to facilitate applications to utilise the security services

1.3 Security Add-in Modules

This layer is made up of service provider modules that offer basic components — cryptographic algorithms, base certificate manipulation facilities, and storage etc.

2.0 How is CDSA being used?

This section attempts to give readers a better understanding of CDSA by providing two perspectives of how CDSA is being used. The first perspective is from a programming point of view – what is sequence of events for an application within CDSA to invoke CSP security services? The second perspective is from a system security services point of view – How can CDSA be used to implement OpenSSL (Open Source Secure Socket Layer)?

2.1 Interaction between CSP and Applications

Each CSSM module (including CSSM itself and add-ins) must be installed on the system before applications can access it. A typical session of interaction between an application and CSP is outlined as follows:

1) The application selects a CSP and requests CSSM to attach to it.
2) The CSSM returns a CSP handle to the application that uniquely identifies the pairing of the application thread to the CSP module instance.
3) The application establishes a "session," a framework in which the CSP will perform cryptographic operations.
4) The application creates an operation "context," which must exist prior to starting CSP operations and is deleted as soon as possible upon completion of the operation.
5) When creating the context, the application specifies an algorithm and may also initialize a session key, pass an initialization vector and/or pass padding information to complete the description of the session.

6) A successful return value from the create function to the application indicates the desired CSP is available.
7) All cryptographic services requested by applications can now be channeled to the CSP via the CSSM.
8) Cryptographic operations might take place:
   - as a single call to perform an operation and obtain a result.
   - as a sequence of calls, starting with an initialization call, followed by one or more update calls, and ending with a completion (final) call. Usually, the result is available after the final function completes its execution.
9) When security services are no longer needed, the application calls CSSM_DeleteContext to delete the session's context information and thus terminates the session.

2.2 Integrating CDSA into OpenSSL

While CDSA provides the building blocks for basic security functions, an application may require use of a higher-level set of protocols (among them, SSL, S/MIME, PGP, or SET). These protocols not only provide a higher level structuring for basic security functions, but also ensure cryptographic interoperability among different platforms. Thus, when Internet application developers field their products, they might employ the SSL or S/MIME protocols to ensure interoperability. Broadly speaking, these high-level security protocols safeguard the transmission of e-commerce information and serve as de facto standards of secure communication.

One of these higher-level protocols is OpenSSL. OpenSSL is a security library that implements the SSL protocol and related functionality for use by applications such as Secure Web Servers. The OpenSSL project is a collaborative effort to develop an Open Source toolkit that implements the Secure Socket Layer (SSL), the Transport Layer Security (TLS v1) protocol, and a full strength general-purpose cryptography library. OpenSSL is being maintained by a worldwide community of volunteers.

Through CDSA, OpenSSL would gain access to a more powerful and complete set of security services; including cryptography, certificate management, trust policies, and secure data storage. In addition, OpenSSL would gain interpretability and extensibility that it does not have now.

As examples how CDSA is being used, the next two sections will outline two approaches that can be used to integrate CDSA with OpenSSL.

Option 1: Integrate CDSA into the SSL crypto library
With this approach, the SSL crypto library API implementation is modified to invoke functionally equivalent CDSA APIs. This approach is transparent to the applications, i.e. the application would call the same SSL crypto library APIs with the same parameters and return types. However, within the SSL crypto API, calls are routed through CDSA CSP APIs. In short, this approach would involve porting of the appropriate SSL crypto primitives and place CDSA calls in each of these crypto primitives.

Options 2: Replace SSL crypto library with CDSA
With this approach, all the SSL crypto library calls will be replaced with functionally equivalent CDSA APIs. This approach is not transparent to the applications, i.e. the application source would need to be modify to call CDSA APIs instead of SSL crypto library APIs to perform crypto operations. In short, this approach would bypass the existing SSL crypto library all together.

3.0 How CDSA addresses Authenticity, Integrity, and Confidentiality?

Although all the CSSM modules contribute to authenticity, integrity and confidentiality, the principal modules that address these are the Cryptography Service Provider (CSP) module and Certificate Library Services (CL) module.

The objective of this section is that by going into more details about how these two modules supports authenticity, integrity and confidentiality that the readers would gain yet another level of understanding of CDSA.

3.1 Cryptography Service Provider

A CSP (whether software and/or hardware -based) provides data encryption/decryption, digital signatures, cryptographic hash ing, key generation, random -number generation services. A CSP supports confidentiality, integrity and authenticity by providing a set of comprehensive APIs.

3.1.1 APIs that support Data confidentiality:

1. Data encryption APIs
   CSP_EncryptData ()
   This functio n encrypts all data contained in the set of input buffers using information in the context.
   CSP_EncryptDataInit ()
   This function initializes the staged encrypt function.
   CSP_EncryptDataUpdate ()
   This function continues the staged encryption process over all data in the set of input buffers.
   CSP_EncryptDataFinal ()
   This function finalizes the staged encryption process by returning any remaining cipher text not returned in the previous staged encryption call.

2. Data decryption APIs (with similar functional description as CSP_EncryptData*)
   CSP_DecryptData ()
   CSP_DecryptDataInit ()
   CSP_DecryptDataUpdate ()
   CSP_DecryptDataFinal ()

3.1.2 APIs that support Data integrity:

1. Sign Data APIs
   CSP_SignData ()
   This function signs all data contained in the set of input b uffers using the private key specified in the context.
   CSP_SignDataInit ()
   This function initializes the staged sign data function. For staged operations, a combination operation selecting both a digesting algorithm and a signing algorithm must be specifie d.
   CSP_SignDataUpdate ()
   This function continues the staged signing process over all data contained in the set of input buffers. Signing is performed using the private key specified in the context.
   CSP_SignDataFinal ()
   This function completes the final sta ge of the sign data function.

2. Verify Data APIs (with similar functional description as CSP_SignData*)
   CSP_VerifyData ()
   CSP_VerifyDataInit ()
   CSP_VerifyDataUpdate ()
   CSP_VerifyDataFinal ()

3. Digest Data APIs (A message digest is the result of a hash operat ion on an arbitrary series of bits)
   CSP_DigestData ()
   CSP_DigestDataInit ()
   CSP_DigestDataUpdate ()
   CSP_DigestDataFinal ()
   CSP_DigestDataClone ()
   This function clones a given staged message digest context with its cryptographic attributes and intermediate result.

4. APIs that support MAC (Message Authentication Code – A function that produces fixed length output from variable-length input and a key. Might be hash-based, cipher-based, or stream-cipher based)
   CSP_GenerateMac ()
   CSP_GenerateMacInit ()
   CSP_GenerateMacUpdate ()
   CSP_GenerateMacFinal ()
   CSP_VerifyMac ()
   CSP_VerifyMacInit ()
   CSP_VerifyMacUpdate ()
   CSP_VerifyMacFinal ()

3.1.3 APIs that facilitate authentication via public/private key management:

CSP_GenerateKey ()
: This function generates a symmetric key.

CSP_GenerateKeyPair ()
: This function generates an asymmetric key pair. (One public, one private)

CSP_WrapKey ()
: This function wraps the supplied key using the context. (Key wrapping/unwrapping means Encryption/decryption of a key. The key can be a symmetric key or private key of a public/private key pair)

CSP_UnwrapKey ()
: This function unwraps the supplied key using the context.

CSP_DeriveKey ()
: This function derives a new symmetric key using the context and/or information from the base key in the context.

CSP_QueryKeySizeInBits ()
: This function gets the logical and effective size of a key. The logical size of the key is the number of bits in the key data for a symmetric key, or the size of the key component commonly used to reference the size of an asymmetric key. The effective size of a key is the number of bits in a key that are actually used in the cryptographic operation.

Since more than 90% of Cryptographic Operations APIs were listed above with simplified descriptions, the reader should now have a pretty good idea about the functionality of the CSP module.

3.2 Certificate Library Services

Authenticity can be supported via the certificate infrastructure. A certificate is a mechanism for establishing identity. The primary purpose of a Certificate Library (CL) module is to perform syntactic manipulations on a specific certificate format, and its associated certificate revocation list (CRL) format. These manipulations cane be grouped into 2 categories:

- Certification Operations
- Certificate Revocation List Operations.

3.2.1 APIs that support Certificate Operations

CL_CertSign ()
: This function sign a certificate using the private key and signing algorithm specified in a context Handle.

CL_CertVerify ()
: This function verifies that the signed certificate has not been altered since it was signed by the designated signer.

CL_CertCreateTemplate ()

This function allocates and initializes memory for an encoded certificate template output. The initialization process includes encoding all certificate field values according to the certificate type and certificate encoding supported by the certificate library module.

CL_CertGetFirstFieldValue ()
This function returns the value of the certificate's first field (designated by the data structure CSSM_OID *CertField*)

CL_CertGetNextFieldValue ()
This function returns the value of a certificate field, when that field occurs multiple times in a certificate.

CL_CertAbortQuery ()
This function terminates a results handle used to access multiple certificate fields.

CL_CertGetKeyInfo ()
This function returns the public key and integral information about the key from the specified certificate.

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

CL_CertDescribeFormat ()
This function returns a list of the values this certificate library module uses to name and reference fields of a certificate.

3.2.2 APIs that support Certificate Revocation List Operations

CL_CrlCreateTemplate ()
This function creates an unsigned, memory-resident CRL. Fields in the CRL are initialized.

CL_CrlSetFields ()
This function will set the fields of the input CRL to the new values, specified by the input.

CL_CrlAddCert ()
This function revokes the input certificate by adding a record representing the certificate to the CRL.

CL_CrlRemoveCert ()
This function reinstates a certificate by removing it from the specified CRL.

CL_CrlSign ()
This function signs a CRL using the private key and signing algorithm. The result is a signed, encoded certificate revocation list.

CL_CrlVerify ()
This function verifies that the signed CRL has not been altered since it was signed by the designated signer.

CL_IsCertInCrl ()
This function searches the CRL for a record corresponding to the certificate.

CL_CrlGetFirstFieldValue ()
This function returns the value of the CRL's first field (designated by the CSSM_OID *CrlField*).

CL_CrlGetNextFieldValue ()
This function returns the value of a CRL field, when that field occurs multiple times in a CRL.

CL_CrlAbortQuery ()
This function terminates the query initiated

CL_CrlDescribeFormat ()
This function returns a list of the values this certificate library module uses to name and reference fields of a CRL.

As with the CSP module, the simplified API descriptions listed above should give the reader a deeper understanding of the CL module functionality.

4.0 What are the possible future trends of CDSA?

Currently, CDSA Version 2 specification has been adopted by The Open Group. Intel developed a reference implementation of the CDSA Version 2 specification for Windows* 95, Windows 98, and Windows NT* 4.0 and Linux. Various vendors such as Hewlett-Packard, IBM have already implemented and committed to CDSA.

A stream of development to be expected is that more and more high-level security services/protocols such as S/MIME, SET, and PGP will be integrated with CDSA.

Currently, the CDSA APIs are specified and implemented in the C language. Given the widely adopted JAVA platform, it is not unreasonable to expect a JAVA package that defines the JAVA classes and interfaces to CSSM-defined services.

Another stream of development to be expected is the integration or collaboration of CDSA with other technologies such as LDAP. For example, the Data Storage Library of CSSM can use LDAP as the supporting infrastructure for storing data objects. In addition, the Trust Policy module can also use LDAP to centrally store and manage policies.

In conclusion, CDSA has gained wide acceptance as an open standard. It is expected that CDSA will continue to be a framework for developing security applications and services for the foreseeable future.

References

Common Security: CDSA and CSSM, Version 2 (with corrigenda)
http://www.opengroup.org/security/cdsa/index.htm

Intel Common Data Security Architecture – Technical Overview
http://www.intel.com/ial/security/tech.htm

Intel Common Data Security Architecture – Frequently Asked Questions
http://www.intel.com/ial/security/faq.htm

Integrating CDSA into OpenSSL (CDSA Engineering Release 3.1.2)
http://developer.intel.com/ial/security/documentation.htm

A Comparison of CDSA to Cryptoki
http://csrc.nist.gov/nissc/1999/program/rd11.htm

"Common Data Security Architecture (CDSA)"
"Common Data Security Architecture (CDSA) Tutorial",
"Integrity Services"
"CSSM Core Services"
"SSL and CDSA"
http://developer.intel.com/ial/security/documentation.htm

Common Data Security Architecture (CDSA) White Paper
http://docs.hp.com/hpux/internet/index.html

Virtual Private Network (VPN) Security
http://www.sans.org/infosecFAQ/encryption/VPN_sec.htm