

**SECURITY SERVICE API:
CRYPTOGRAPHIC API RECOMMENDATION
UPDATED AND ABRIDGED EDITION**

NSA Cross Organization CAPI Team
The National Security Agency
Ft. Meade, Maryland
July 25, 1997

ABSTRACT

Until recently, the integration of cryptographic functionality into application software has required that developers tightly couple the application to the cryptographic module. This approach forces each new combination of application and cryptography to be treated as a distinct development effort, and does not provide the modularity and maintainability expected of commercial products. An approach that can provide flexibility and cost savings is the use of a standardized Cryptographic Application Program Interface (CAPI) suite. In this paper, the CAPI evaluation criteria is presented as well as the resulting suite of recommended CAPIs. This paper is an updated and abridged edition of the original CAPI papers [1,2].

INTRODUCTION

As application developers become aware of the need for cryptographic protection, they are adding “hooks” to access the cryptographic functionality developed by others. Those “hooks” are known as the Cryptographic Application Programming Interface (CAPI). As CAPIs mature and gain sophistication, the benefit of a standard interface increases. Applications that utilize a standard CAPI can access multiple cryptographic implementations through a single interface. This decreases life cycle implementation efforts. Likewise, cryptographic modules that are built to a standard CAPI can be accessed by a greater number of applications, increasing portability.

There are numerous efforts currently under way to create CAPI standards. They range in scope from very generic security support like that found in Generic Security Services Application Programming Interface (GSS-API) to an interface more involved in the direct control of the cryptographic token like that found in PKCS #11. A number of these CAPI efforts are receiving a great deal of support as applications and cryptographic modules are being written to use them. While we would ideally like to select a single CAPI standard for all applications, multiple CAPIs are required to support the broadest range of applications and cryptographic modules.

Increases in internal development costs, along with increased availability of commercial cryptography, have changed how implementers should integrate security and cryptography into applications. While the development of secure applications by industry reduces the need to develop unique application software, it replaces it with a new problem. Our programmers must now create software that maps the CAPIs used by our applications to the cryptographic modules. To accomplish the mapping from security-enabled commercial, off-the-shelf (COTS) applications to a standard suite of CAPI mechanisms, we must use a subset of the CAPI standards, encouraging other CAPI efforts to align with one of the standards in this subset (i.e., the recommended CAPI suite).

The recommended CAPI suite is comprised of the following Security Services APIs (SSAPIs) to include CAPIs:

- Generic Security Services API & Independent Data Unit Protection GSS-API
- Common Security Services Manager API
- CryptoAPI
- PKCS #11 - Cryptoki

SCOPE

This document recommends a suite of SSAPIs and CAPIs to be used with COTS applications for use in systems integration work. These APIs insulate application developers from the variety of cryptographic libraries, hardware tokens, and software tokens, giving the user maximum flexibility in selecting cryptographic services for inclusion in an application. These APIs also greatly simplify the process of incorporating various cryptographic algorithms to be used by an application.

The concept of cryptographic awareness has been used by several SSAPI/CAPI developers to describe the level of security service that an application needs to access. This concept can be extended to include the level of security service access that the operating system allows. The goal is for general-purpose applications (e.g., spreadsheets, document processors, E-mail, etc.) to be cryptographically

unaware, utilizing only a minimum number of high-level security calls without having to know about the underlying cryptography and security support (i.e., certificate management, key management, data isolation). Ideally, these high-level calls would initiate security contexts, protect data, and terminate security contexts. These calls would require no knowledge of specific cryptographic algorithms or modules. At most, the application might indicate a desired “quality of protection.”

Special-purpose applications, like certificate authority applications, are cryptographically aware and require an extensive suite of calls that permits precise control of the cryptographic token. These calls require extensive cryptographic knowledge from the implementers of the cryptographic application. In this case, the application may select a specific cryptographic algorithm or mode of operation. Care must be taken when using a low-level interface since a mistake by the caller may compromise the security of the cryptographic system.

Varying levels of cryptographic awareness provide a metric that allows layering amongst the recommended CAPIs. Using this metric, along with the level of abstraction with respect to our recommended APIs, produces the layering scheme illustrated in Figure 1.

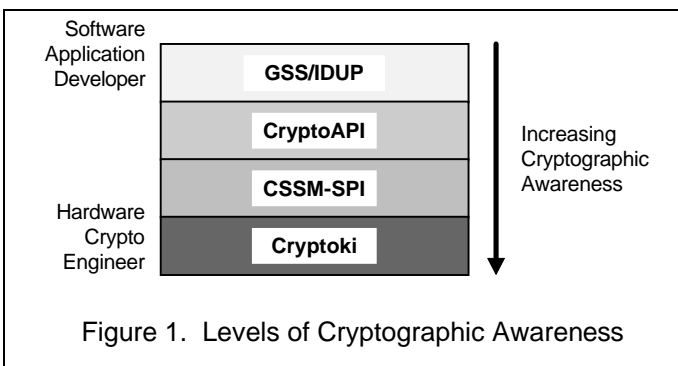


Figure 1. Levels of Cryptographic Awareness

BACKGROUND

The following section gives background information on each API reviewed and subsequently recommended. Within this section, the terminology will mirror that used by the API being discussed. This approach was chosen to allow the reader to easily apply our recommendations while referencing a particular API specification.

The Generic Security Services API (GSS-API) [3] and the extensions for Independent Data Unit Protection (IDUP-GSS-API) [4] support cryptographically unaware applications. These SS APIs provide a high-level interface to authentication, integrity, confidentiality, and nonrepudiation (IDUP only) services. The application merely indi-

cates the required security services and optionally the quality of protection (QOP) for the per-message services. GSS-API was designed to protect session-style communications (e.g., ftp) between entities. IDUP-GSS-API does not assume real-time communications between sender and recipient, and protects each data unit (e.g., files or messages) independently of all others. IDUP-GSS-API is therefore suitable for protecting data in store-and-forward applications. These specifications were developed within the Common Authentication Technology (CAT) group within the Internet Engineering Task Force (IETF).

The Common Security Services Manager API (CSSM-API) [5] is the heart of Intel’s Common Data Security Architecture (CDSA) [6]. CSSM-API¹ offers a robust set of security services to include: cryptography, certificate management, trust policy, data storage, and optionally key recovery. CSSM-API also has the capability to provide integrity services via the Embedded Integrity Services Library (EISL) and to support auditing services. CSSM-API was developed at Intel Architecture Labs and is currently being “Fast-Tracked” as a standard within the Security Program Group (SPG) of the Open Group (the result of the X/OPEN and the Open Software Foundation merger). While CSSM services like certificate management, trust policy, and data storage fit logically at the middle level, the actual CAPI calls (their Cryptographic Service Provider Interface - SPI) are more low-level like Cryptoki. For instance, CSSM-SPI supports user authentication and administrative control of tokens.

The Microsoft CryptoAPI [7] supports cryptographically aware applications. As a service suite provided by the Windows NT operating system, CryptoAPI provides extensive facilities for utilizing both hardware and software cryptographic modules, called Cryptographic Service Providers (CSPs). CryptoAPI was developed by Microsoft and therefore has not been subjected to any formal standards process. The authors did, however, consult with various government and corporate customers while developing the CryptoAPI specification. Applications using CryptoAPI can take advantage of default features of the interface to reduce their cryptographic awareness requirements, or they can exert full control over algorithms, keys, and modes of operation.

PKCS #11 - Cryptoki [8] is an abstract token interface that defines the arguments and results of various algorithms. Cryptoki also specifies certain objects and data structures which the token makes available to the applica-

¹ CSSM-API is replacing the Generic Cryptographic Services API (GCS-API) that was deprecated by the Open Group in March 1997.

tion. Cryptoki interfaces directly to cryptographic tokens, and is thus the logical place for functions that allow user authentication (e.g., logon or PIN entry) and administrative control of the token. Cryptoki is appropriate for use by developers of cryptographic devices and libraries. Cryptoki was developed by RSA Labs and is a member of their family of Public Key Cryptography Standards (PKCS). Continuing development of Cryptoki is accomplished by the PKCS #11 workshops sponsored by RSA Labs and held annually for all interested parties.

CRITERIA

The following criteria were used to analyze the CAPIs.

- **Algorithm Independence:** *Algorithm Independence* is defined as the property that the CAPI does not specify use of a particular algorithm to provide cryptographic service. The CAPI must accommodate a broad range of choices of current and future cryptographic algorithms. This property gives an application access to any cryptographic algorithm supported by the underlying cryptomodule, enhancing interoperability. An example of algorithm independence is a reference to an “encryption algorithm” rather than the “DES algorithm.”
- **Application Independence:** *Application Independence* is defined as the property that the CAPI is equally suitable for designing any application. The CAPI must provide cryptographic services to the wide variety of applications being written today as well as future applications. A CAPI that has the property of application independence will enable the programmer to write a wide variety of applications. This will give the CAPI widespread use and longevity. A well-designed CAPI should be able to support all applications such as store-and-forward applications, like E-mail, as well as connection-oriented applications, like file transfer.
- **Cryptomodule Independence:** *Cryptomodule Independence* is the property that the CAPI, in providing its cryptographic service, can use a particular cryptomodule as easily as any other. The application should not need to know the specifics of the underlying cryptographic implementation. For example, the application need not know whether the cryptography is provided in hardware or software. Cryptomodule independence provides a solid foundation for the use of multiple cryptomodule implementations.
- **Degree of Cryptographic Awareness:** *Cryptographic Awareness* refers to the amount of cryptographic knowledge required of the application developer. The goal for the majority of applications is to require only a minimal degree of cryptographic knowledge from the developer.

Some applications, for example those in key management, require a higher degree of cryptographic knowledge from the developer. CAPI specifications can fall anywhere in this spectrum from requiring little knowledge to requiring a great degree of expertise in cryptography. Therefore a CAPI suite must be used in order to support both cryptographically aware and cryptographically unaware applications.

- **Modular Design and Auxiliary Services:** *Modular Design* is the division of the CAPI into units that work together to provide the complete security service, each of which has a focused purpose. *Auxiliary Services* are support services used by the goal cryptographic service. These can include: key life-cycle management, cryptomodule verification, user authentication, certificate management, query capability, and (user-to-CAPI) session set-up/tear-down capability. The CAPI architecture must also be functionally complete. In simpler architectures, there may be only two modules: cryptographic service and key management. It is expected that auxiliary services that are not fully developed and modularized today will become separate modules. In many current standards, the lack of APIs providing access to these security support services has caused the inclusion of that functionality within the CAPI. While this meets our needs for now, continuing development will most likely result in numerous other auxiliary service APIs.

As a separate auxiliary service, the CAPI must support, preferably include, a function to verify its underlying cryptosubsystem. It is envisioned, for example, that cryptographic token verification functionality could filter up through the CAPI suite from a lower level CAPI.

- **MISSI² Support:** *MISSI Support* is defined as the ability to be extended to support current and anticipated MISSI cryptography. It is vital that no CAPIs preclude utilizing current cryptographic products supported by MISSI.
- **Safe Programming:** *Safe Programming* is the term used for describing steps taken to guard against inadvertent security mishaps by the programmer. Three aspects that contribute to safe programming include consistent naming conventions, information hiding, and ease of use. By having consistent naming conventions, the possibility of a

² Multilevel Information System Security Initiative (MISSI) provides evolutionary, interoperable security solutions for the Defense Information Infrastructure (DII) constituent programs via an integrated, cohesive security architecture composed of compatible building block products and a common Security Management Infrastructure.

programmer misunderstanding the purpose and use of each procedure and its parameters decreases. Information hiding is provided by the use of opaque handles and pointers, to prevent the inadvertent exposure of sensitive information. Ease of use results in the CAPI mechanism doing many of the detailed tasks of parameterizing and sequencing of cryptographic operations, and is likely to decrease the probability of programmer error. This is especially important for CAPI specifications aimed at the cryptographically unaware programmer.

• **Security Perimeter:** *Security Perimeter* is defined as the boundary that prevents sensitive security-related information from leaking out of the trusted computing base into untrusted applications. In trusted systems, architectures with a security perimeter will contain the cryptography within the perimeter while the application is outside. The CAPI provides the access between the untrusted components and the trusted components. The CAPI must not be used as a portal to violate the security perimeter. It must restrict access to sensitive cryptographic data (e.g., unprotected keys) and must not propagate such information beyond the CAPI interface.

CONCLUSION

Applying the criteria to the APIs in the recommend CAPI suite produces the results in Table 1.

Criteria	GSS/IDUP	CSSM	CryptoAPI	PKCS #11
Algorithm Independence	Yes	Yes	Yes	Yes
Application Independence	Yes _[1]	Yes _[2]	Yes _[2]	Yes _[2]
Cryptomodule Independence	Yes	Yes	Yes	Yes
Cryptographic Awareness	No	Yes _[3]	Yes _[3]	Yes _[3]
Auxiliary Services	—	—	—	—
Cryptomodule Verification	No	Yes	Yes	No
User Authentication	Yes _[4]	Yes _[5]	Yes _[5]	Yes _[5]
Certificate Management	Some	Yes	No _[6]	No
Query Capability	No	Yes	Yes	Yes
Set-up & Tear-Down	Yes	Yes	Yes	Yes
MISSI Support	Yes	Yes	Yes	Yes
Safe Programming _[7]	5	2	2	2

Security Perimeter	Yes	Yes	Yes _[8]	Yes
--------------------	-----	-----	--------------------	-----

Table 1: CAPI Analysis Summary

[1] GSS and IDUP each handle the different application paradigms.

[2] CSSM, CryptoAPI, and Cryptoki are low-level to be independent of the application.

[3] CSSM, CryptoAPI, and Cryptoki are intended for the cryptographically aware programmer.

[4] GSS and IDUP provide authentication services defined within their corresponding mechanism specifications.

[5] CSSM, CryptoAPI, and Cryptoki support user authentication to the cryptomodules via PIN capabilities.

[6] Currently CryptoAPI version 1.0 does not support any certificate management capabilities; however, CryptoAPI version 2.0 will provide support for certificate storage, and manipulation.

[7] Safe Programming is weighted from 1 through 5, with 5 being the most safe.

[8] Microsoft CryptoAPI specifies a programming interface that supports this function. The degree of support is CSP specific.

All of the APIs in the recommended CAPI suite provide cryptomodule independence and can be extended to support MISSI cryptography. The main difference between the CAPIs is in the areas of the amount of cryptographic knowledge required by the application developer, and the amount of sensitive information that can be recovered through the CAPI. GSS-API and IDUP-GSS-API (GSS/IDUP) provide the safest interface, but the most limited capability to manipulate the cryptography. CSSM-API, CryptoAPI, and Cryptoki provide applications with more capabilities to manipulate the cryptography, increasing the risk that the application may misuse the interface. Since the majority of applications will be cryptographically unaware, the recommendation is for application developers to use GSS/IDUP. CSSM-API, CryptoAPI, and Cryptoki should be used only when developing cryptographically aware applications. Besides being directly called by cryptographically aware applications, the three lower level APIs (i.e., CSSM-API, CryptoAPI, and Cryptoki) could also be used to build the underlying GSS/IDUP mechanisms. Although any of the three low-level CAPIs could be layered to provide the detailed cryptographic functions for one of the others (i.e., CryptoAPI over CSSM-SPI, or CSSM-SPI over Cryptoki), this would be redundant and is discouraged. This now provides cryptographic developers a choice of three low-level CAPIs to choose from for their cryptographic modules.

This provides us with the recommended CAPI suite illustrated in Figure 2.

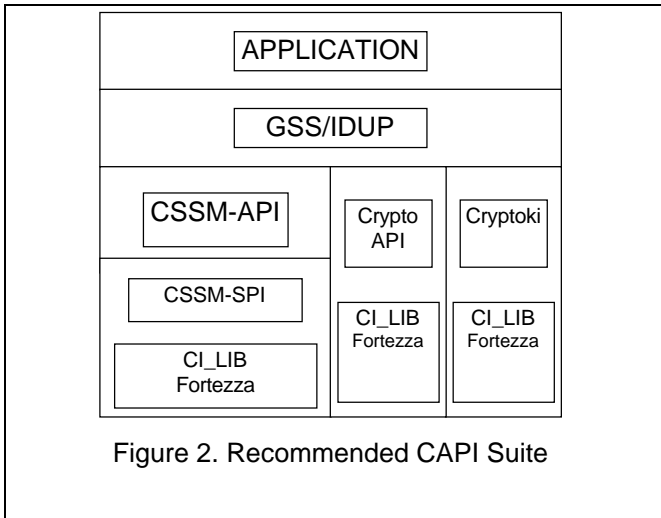


Figure 2. Recommended CAPI Suite

While this presents the most complete set of functionality, this document does not mandate having to incorporate all the recommended CAPIs. In reality, combinations of the recommended CAPIs will more likely be used. Development of “yet-another-CAPI” should be avoided. The appropriateness of a layered CAPI suite depends on the security policy and the intended environment.

During the last two years, our original CAPI recommendations have generated a greater awareness with respect to CAPIs. In addition, the CAPI specifications have gone through several revisions, Microsoft announced CryptoAPI, and Intel has developed CSSM-API. Therefore we are continuing to update our CAPI recommendation with this updated and abridged release. We hope this document will provide application developers and cryptographic developers with enough information to make intelligent choices.

Now that the CAPI recommendation has been updated, the CAPI team is focusing on implementing the recommended CAPI suite. The first prototype, Cryptoki, has been completed. Currently, researchers are completing the prototype of GSS-API over the Simple Public Key Mechanism (SPKM) using CryptoAPI, followed by inclusion of the IDUP extensions. The next step is to integrate and test all three low-level CAPIs: CryptoAPI, CSSM-SPI, and Cryptoki with varying underlying cryptomodules. As the implementation efforts continue, the team will provide feedback to the appropriate standards body or author. In addition, we are planning on publishing implementation guidance.

REFERENCES

- [1] NSA Cross-Organization Team, “Security Service API: Cryptographic API Recommendation,” First Edition, National Security Agency, June 12, 1995.
- [2] NSA Cross-Organization Team, “Security Service API: Cryptographic API Recommendation,” Second Edition, National Security Agency, July 1, 1996, <http://www.tis.com/docs/research/crypto/ice/nsacapi/capi19.htm>
- [3] Linn, J., “Generic Security Service Application Program Interface - version 2” RFC 2078, January, 1997, <ftp://ds.internic.net/rfc/rfc2078.txt>.
- [4] Adams, C., “Independent Data Unit Protection Generic Security Service Application Program Interface (IDUP-GSS-API),” Internet draft 7, March 28, 1997, <http://ftp.ietf.org/internet-drafts/draft-ietf-cat-idup-gss-07.txt>.
- [5] Intel Architecture Labs, “Common Security Services Manager Application Programming Interface (CSSM-API),” Draft 2.0, June 16, 1997, <http://www.opengroup.org/public/tech/security/pki/index.htm>
- [6] Intel Architecture Labs, “Common Data Security Architecture (CDSA),” Draft 2.0, June 16, 1997, <http://www.opengroup.org/public/tech/security/pki/index.htm>.
- [7] Microsoft Corporation, “Application Programmer’s Guide: Microsoft CryptoAPI, Version 1.0, January 1996.
- [8] Kaliski, B., “Cryptoki: A Cryptographic Token Interface, Version 1.0.” RSA Laboratories, April 28, 1995, <http://www.rsa.com/rsalabs/pubs/PKCS/html/pkcs-11.html>