

APPENDIX D: GSS/IDUP Changes

D.1 GSS-API Changes

- More general with respect to the underlying mechanisms residing below the API
- Recommendation text that deals with default behavior
- Security contexts are now opaque objects vice opaque octet strings
- Restructuring of naming conventions
- Combinations of anonymous and mutual authentication states
- New boolean state for Per-Message protection during context establishment to ensure backwards compatibility with GSS version 1
- New section providing recommendations with respect to implementation robustness
- New calls: GSS_Inquire_cred_by_mech, GSS_Inquire_names_for_mech, GSS_Inquire_mechs_for_name, GSS_Canonicalize_name, and GSS_Export_name
- Deleted calls: GSS_Import_name_object and GSS_Export_name_object

D.2 IDUP-GSS-API Changes

- Additional detailed text with respect to the use of QOP
- Additional detailed text with respect to the provision of time
- Object Identifier definitions for protection/unprotection services
- New calls: IDUP_Form_Complete_Evidence
- Deleted calls: IDUP_Process_Receipt
- Addition of environment-level policy parameters
- The number of major status codes has grown substantially
- Introduction of the concept of parameter bundles that provide great detail about the operational conditions of the Per-IDU calls

C.3 DMS: A Government Specific Example

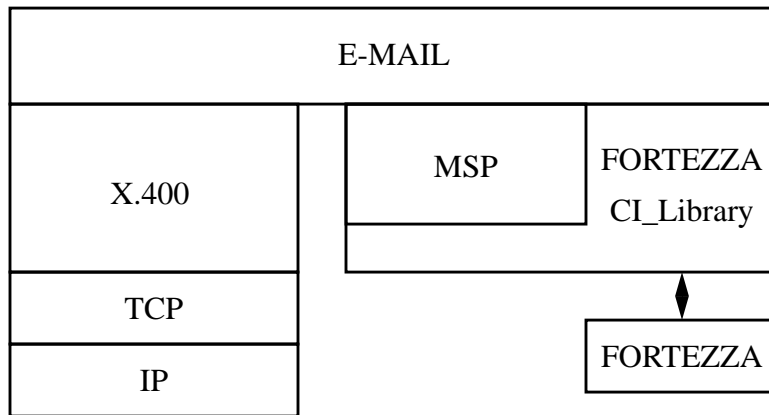


Figure 6: DMS Without CAPIs

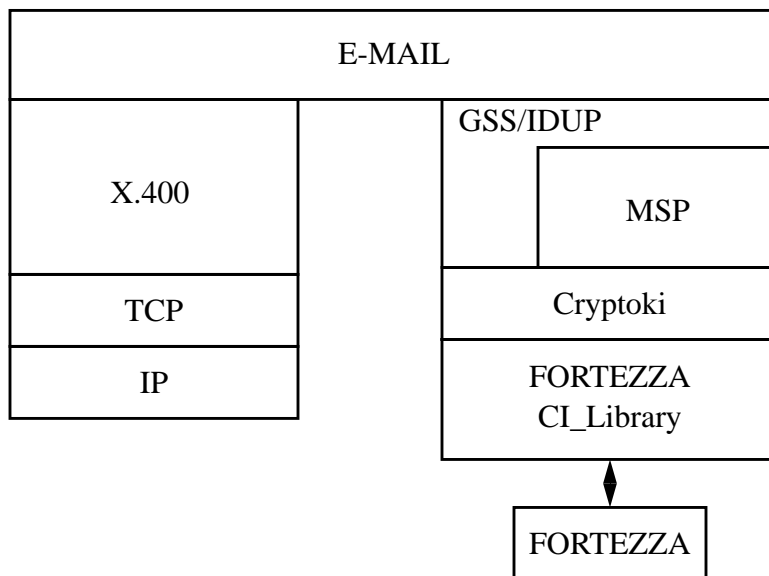


Figure 7: DMS Using CAPIs

The first scenario (**Figure 6**) shows how DMS currently makes use of the FORTEZZA card by directly making CI_Library calls from both the email application and the Message Security Protocol (MSP). The second scenario (**Figure 7**) shows what has to be done in order to incorporate the recommended CAPI solution. The CI_Library calls within the email application are replaced with GSS/IDUP, while the CI_Library calls within MSP would have to be replaced with Cryptoki calls. The benefit of this solution is that it requires a one-time alteration to MSP, and then minimal further alterations to MSP for future FORTEZZA releases or for more advanced cryptographic tokens.

C.2 Complete

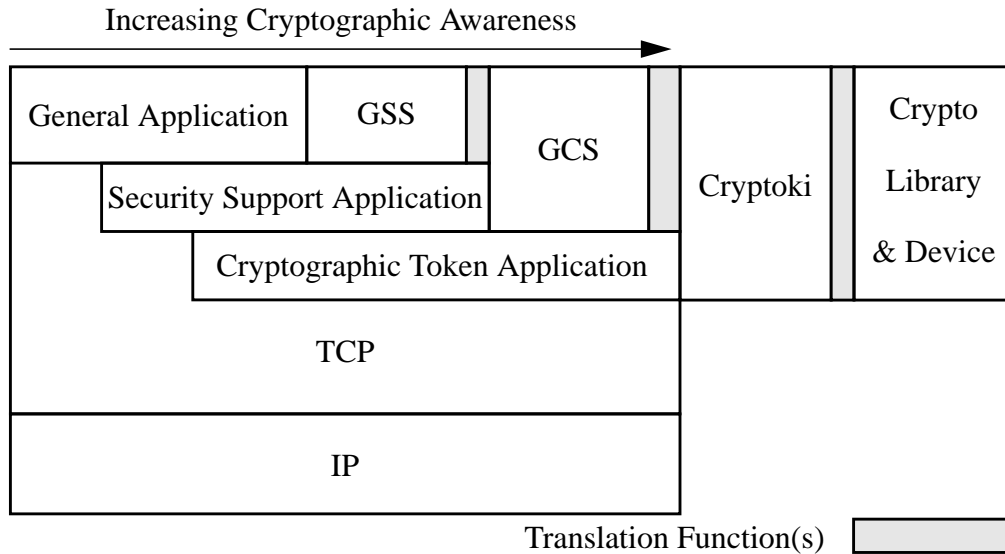


Figure 4: Complete CAPI Scenario 1

This scenario (**Figure 4**) provides the same capabilities as the general scenario with the addition of the GCS-API. The GCS-API provides the capability for security support applications and to make cryptographically aware calls. Security management is envisioned to be introduced at this layer. Again in order to make this a reality, a one-time effort to develop the thin layers of translation code will have to occur.

Figure 5 shows how the same scenario might look under a Microsoft proprietary operating system, such as Windows NT.

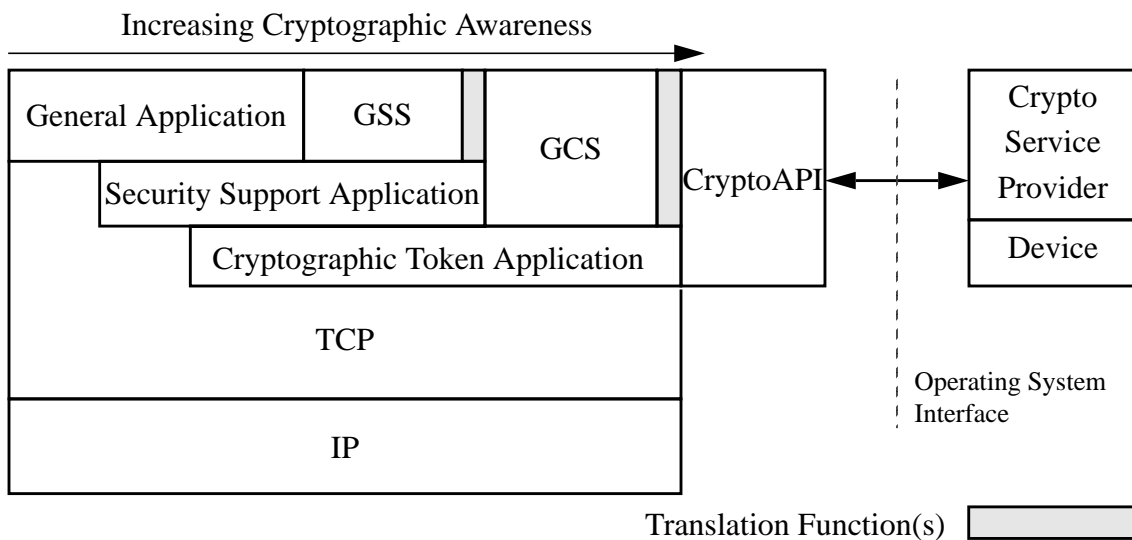


Figure 5: Complete CAPI Scenario 2

APPENDIX C: SCENARIOS

This appendix is provided to illustrate how CAPIs might be employed in system and application implementations. For these scenarios either hardware or software cryptographic modules could be used.

C.1 General

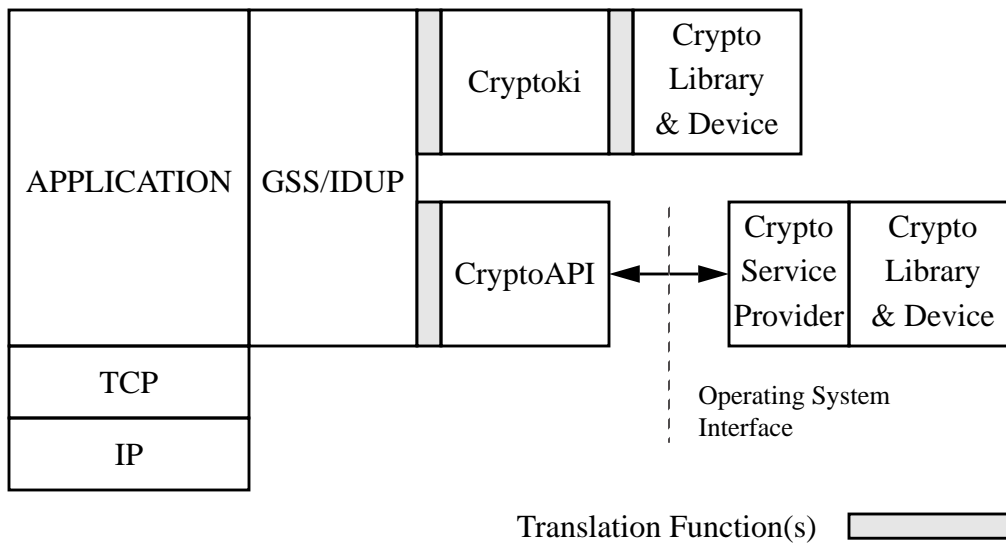


Figure 3: General CAPI Scenario

This scenario (**Figure 3**) illustrates how any cryptographically unaware application requiring security services should make GSS/IDUP calls to receive those services. Likewise any cryptographic token that builds to Cryptoki or CryptoAPI can be accessed by any application via a GSS/IDUP to Cryptoki or CryptoAPI layering. In order to make this a reality, a one-time effort to develop the thin layers of translation code will have to occur.

symmetric keys: see secret key.

token: The underlying hardware or software cryptographic device [5].

data structure: The format in which the cryptographic material and/or attributes are stored.

decipher: Convert enciphered text to the equivalent plain text by means of a cipher system [7].

decode: Convert encoded text to its equivalent plain text by means of a code [7].

decrypt: Generic term encompassing decode and decipher [7].

encipher: Convert plain text to equivalent cipher text by means of a cipher [7].

encode: Convert plain text to equivalent cipher text by means of a code [7].

encrypt: Generic term encompassing encipher and encode [7].

key: Information (usually a sequence of random or pseudorandom binary digits) used initially to set up and periodically change the operations performed in crypto-equipment for the purpose of encrypting or decrypting electronic signals, for determining electronic counter-countermeasures patterns, or for producing other key [7].

key blob: A provision to store keys outside of the CSP. Key blobs are created by exporting an existing key out of the provider[10].

key management: Process by which key is generated, stored, protected, transferred, loaded, used, and destroyed [7].

MISSI: Multilevel Information System Security Initiative. An NSA program for developing and delivering data and network security products for the U.S. government.

nonrepudiation: Method by which the sender of data is provided with proof of delivery and the recipient is assured of the sender's identity, so that neither can later deny having processed the data [7].

peer-entity authentication: The corroboration that a peer entity in an association is the one claimed [4].

private key: A key used in an asymmetric cryptographic algorithm. Possession of this key is restricted, usually to only one entity [4].

public key: The key used in an asymmetric cryptographic algorithm, that is publicly available [4].

secret key: The shared key between two entities that is used in a symmetric cryptographic algorithm [4].

security aware: The caller of an API that is aware of the security functionality and parameters which may be required and/or provided by an API [4].

security context: The relationship established between peers, using credentials established locally in conjunction with each peer or received by peers via delegations [1].

security policy: The set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information [9].

APPENDIX B: GLOSSARY

access control: Process of limiting access to the resources of an automated information system (AIS) only to authorized users, programs, processes, or other systems [7].

access control policy: The set of rules that define the conditions under which an access may take place [4].

API - Application Programming Interface: The interface between the application software and the application platform, across which all services are provided. The application programming interface is primarily in support of application portability, but system and application interoperability is also supported by a communication API [4].

asymmetric keys: Different keys are used to encrypt and decrypt data (i.e., public and private keys) [4].

authentication: Security measure designed to establish the validity of a transmission, message, or originator, or a means of verifying an individual's eligibility to receive specific categories of information [7].

availability: The property of being accessible and usable upon demand by an authorized entity [8].

cipher: Cryptographic system in which units of plain text are substituted according to a predetermined key [7].

ciphertext: Enciphered information [7].

clear text: Intelligible data that are not encrypted [4].

confidentiality: Assurance that information is not disclosed to unauthorized entities or processes [7].

credentials: Information, passed from one entity to another, that is used to establish the sending entity's access rights [7].

context: See security context.

cryptographic algorithm: Well-defined procedure or sequence of rules or steps used to produce cipher text from plain text and vice versa [7].

cryptographic context: See security context.

cryptographic services provider: Independent modules within the Microsoft Windows NT environment that perform cryptographic and/or cryptography support. CSPs are accessed via CryptoAPI[10].

cryptography: Principles, means, and methods for rendering plain information unintelligible and for restoring encrypted information to intelligible form [7].

data integrity: Condition that exists when data are unchanged from their source and have not been accidentally or maliciously modified, altered, or destroyed [7].

data origin authentication: Corroboration that the source of data is as claimed [7].

APPENDIX A: ACRONYMS

AH	Authentication Header
ANSI	American National Standards Institute
ASN.1	Abstract Syntax Notation version One
API	Application Program Interface
BER	Basic Encoding Rules
CAPI	Cryptographic Application Program Interface
CAT	Common Authentication Technology
CCITT	International Telegraph & Telephone Consultative Committee
CSP	Cryptographic Service Provider
COTS	Commercial Off The Shelf
DLL	Dynamic Linked Library
DES	Digital Encryption Standard
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
ESP	Encapsulating Security Payload
FIPS	Federal Information Processing Standard
GCS-API	Generic Crypto Service Application Program Interface
GSS-API	Generic Security Service Application Program Interface
GULS	Generic Upper Layers Security (protocol)
IDUP-GSS-API	Independent Data Unit Protection GSS-API
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
KEA	Key Encryption Algorithm
MISSI	Multilevel Information System Security Initiative
MSP	Message Security Protocol
NIST	National Institute of Standards and Technology
NLSP	Network Layer Security Protocol
NSA	National Security Agency
OID	Object Identifier
PKCS	Public Key Cryptography Standard
PIN	Personal Identification Number
POSIX	Portable Operating System Interface
QOP	Quality of Protection
RFC	Request For Comments
S-HTTP	Secure Hyper Text Transfer Protocol
TLSP	Transport Layer Security Protocol

11. REFERENCES

- [1] Emmett Paige, Jr., "Selection of Migration Systems," Assistant Secretary of Defense Memorandum, November, 1993.
- [2] Linn, J., "Generic Security Service Application Program Interface," RFC 1508, November, 1993.
- [3] Adams, C., "Independent Data Unit Protection Generic Security Service Application Program Interface (IDUP-GSS-API)," Internet draft 4, February, 1996.
- [4] X/Open, "X/Open Preliminary Specification: Generic Cryptographic Service API," draft 8, April 20, 1996.
- [5] Kaliski, B., "Cryptoki: A Cryptographic Token Interface, Version 1.0." RSA Laboratories, April 28, 1995.
- [6] Linn, J., "Generic Security Service Application Program Interface, Version 2, Internet draft 5, February, 1996.
- [7] National Security Telecommunications and Information Systems Security Committee, "National Information Systems Security Glossary," NSTISSI No. 4009, 5 June 1992
- [8] National Computer Security Center, "An Introduction to Certification and Accreditation," NCSC-TG-029, January 1994
- [9] National Computer Security Center, "A Guide to Understanding Security Modeling in Trusted Systems, NCSC-TG-010," October 1992
- [10] Microsoft Corporation, "Application Programmer's Guide: Microsoft CryptoAPI, preliminary version 0.9, January 17, 1996
- [11] NSA Cross-Organization Team, "Security Service API: Cryptographic API Recommendation," First Edition, National Security Agency, June 12, 1995

10. ACKNOWLEDGMENTS

This document is the result of a collaborative team effort. Acknowledgment is due to the many people who participated. Special acknowledgments are due to Shu-jen Chang, who authored NIST's proposed FIPS-Cryptographic Service Calls, Rich Ankney of Fischer International, who served as a consultant for the document, and David S. Miller of Cybersafe, who was our GCS-API team leader.

Another problem is that the market support and acceptance for a middle-level CAPI (i.e., GCS-API) may not develop. We are continuing to support X/Open SWG to minimize this risk and to encourage GCS-API as a middle-level CAPI.

9.3 Current Status

During the last year, our original CAPI recommendation has generated a greater awareness with respect to CAPIs. In addition, the CAPI specifications have gone through several revisions and Microsoft announced CryptoAPI. Therefore we decided to update our CAPI recommendation with this 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 a multitiered CAPI environment. The first prototype, Cryptoki, has been completed. Currently, researchers are building a GSS-API Simple Public Key Mechanism (SPKM) over CryptoAPI and Cryptoki, followed by inclusion of the IDUP extensions. At that point a decision will be made about GCS-API and how much further the specification has developed and whether user and vendor interest has increased. If favorable, then an effort to build a GCS-API mechanism will be initiated. 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.

Once the CAPI recommendation is deemed valid (via the implementations), project managers can begin putting CAPI requirements into their migration strategies for future releases of their products.

9.1 Recommendation

To restate, the recommendation is for application developers to use GSS/IDUP. GCS-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 C APIs (i.e., GCS-API, CryptoAPI, and Cryptoki) could also be used to build the underlying GSS/IDUP mechanisms. This provides us with a multitiered C API suite (**Figure 2**). GCS-API could build the security support functions for GSS/IDUP. Likewise, Cryptoki or CryptoAPI could provide the detailed cryptographic functions for GCS-API. While this presents the most complete set of functionality, this document does not mandate having to incorporate all the recommended C APIs. In reality, combinations of the four recommended C APIs will more likely be used. Development of “yet-another-C API” should be avoided. The appropriateness of a multitiered C API suite depends on the security policy and the intended environment. For example, on Microsoft-based operating systems, CryptoAPI could form the basis for the higher level C APIs, while on all other operating systems Cryptoki could be used.

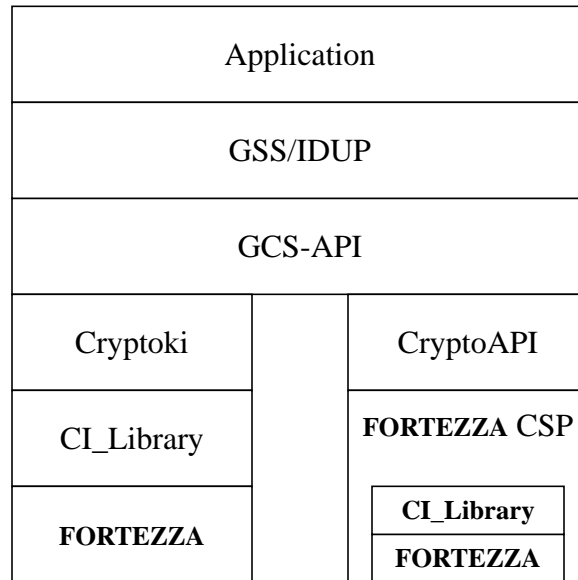


Figure 2: Recommended Multitiered C API Suite

9.2 Risks

As with all recommendations, this one contains risks. A potential problem is improper use of a lower level C API resulting in weakened cryptographic security. Since much of the software we intend to use will be COTS, we will have little control over the internal operation of the software. A number of approaches can be taken to limit the potential damage, ranging from general C API compliance testing to using a cryptographic infrastructure below the C API to reduce the amount of sensitive information that can be lost as a result of improper use of the C API.

All four CAPIs 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/IDUP provides the safest interface, but the most limited capability to manipulate the cryptography. GCS-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, it is recommended that application developers use GSS/IDUP. In those cases where an application requires direct access to the cryptography to perform security support or manipulate cryptographic tokens, GCS-API, CryptoAPI, or Cryptoki should be considered.

Table 2 lists envisioned applications and protocols and their associated recommended CAPI(s). Note: CAPIs are not strictly for applications, but are also available for communications protocols.

Application/Protocol	Recommended CAPI
Word Processor	GSS/IDUP
Mail Application	GSS/IDUP
File Storage	GSS/IDUP
Directory Service	GSS/IDUP
Network Management	GSS/IDUP
Authentication Application	GSS/IDUP
Key Management Application	GCS-API
Security Association Protocol	GCS-API, CryptoAPI, or Cryptoki
Certification Manager	GCS-API, CryptoAPI, or Cryptoki
ESP; AH; NLSP; TLSP	CryptoAPI or Cryptoki
MSP; S-HTTP; SSL; GULS	CryptoAPI or Cryptoki
Cryptographic Token Application	Cryptoki

Table 2:

9. CONCLUSION

This document has examined four C APIs that support a variety of functions at varying levels of cryptographic awareness. Table 1 summarizes the criteria from Section 4 and indicates how the four recommended C APIs met the criteria.

Criteria	GSS/IDUP	GCS	CryptoAPI	Cryptoki
Algorithm Independence	yes	yes	yes	yes
Application Independence	yes _[1]	yes _[2]	yes _[2]	yes _[2]
Cryptomodule Independence	yes	yes	yes	yes
Detailed Cryptographic Awareness Required	no	yes _[3]	yes _[3]	yes _[3]
Design and Auxiliary Services	-----	-----	-----	-----
Key Life Cycle Management	no	yes	no	no
Cryptomodule Verification	no	no	yes	no
User Authentication	no	no	yes	yes
Certificate Management	some	no	no	no
Query Capability	no	no	yes	yes
Set-up/Tear-down Capability	yes	yes	yes	yes
MISSI Support	yes	yes	yes	yes
Safe Programming [4]	4	2	2	2
Security Perimeter	yes	yes _[5]	yes _[6]	yes

Table 1:

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

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

[3] GCS-API, CryptoAPI, and Cryptoki are intended for the cryptographically aware programmer.

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

[5] Keys are protected physically (using hardware) or cryptographically (encrypted under a facility master key). Similar protection is provided for intermediate function results (if not kept within the security perimeter) and “exported” contexts.

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

8.4 Current Status

Version 1.0 of PKCS #11 has been available and stable since April 1995. Several cryptographic token vendors have adopted the API for their tokens. NSA has completed a prototype implementation for the FORTEZZA token. RSA is making a free PC/Windows implementation (based on a software token, hence its name is “SofToken”) for prospective developers, and they are also selling a version of their BSAFE commercial toolkit designed to utilize the Cryptoki interface to hardware personal tokens.

RSA Labs plans to revise the PKCS #11 standard sometime in 1996. According to RSA Labs, the new version 2.0 will use CCITT-style object identifiers (OIDs) instead of C numeric constants to represent algorithms, attributes and object types. The extension mechanisms will be better structured, and the set of predefined algorithms and attributes expanded.

8.5 General

Cryptoki is a CAPI for cryptographically aware applications. Cryptoki presents an abstract token interface with most of the functionality required at this degree of cryptographic awareness. Additional functionality can be added using the normal Cryptoki extension mechanisms. Experience with the prototype has shown that, by using Cryptoki, only a minimal amount of application code needs to be changed to utilize different cryptographic tokens.

Cryptoki is also very strict. No internal data structures are exposed by the API, and the standard specifies check conditions for all data passed to the Cryptoki session from the application. Interaction with Cryptoki is limited exclusively to the functions defined in the standard. Cryptoki uses predefined, platform-independent data types for all subroutine arguments, allowing the compiler to perform strict type-checking.

Security Perimeter

Cryptoki does explicitly hide sensitive information (e.g. private keys). Practically everything that Cryptoki manipulates is an opaque object with a set of attributes. One such attribute is the CKA_PRIVATE attribute that, when TRUE, indicates the object cannot be exported outside of the token.

8.3 Weaknesses

The following RSA Cryptoki weaknesses were examined. Some of these weaknesses can be characterized as a function of being a lower level interface.

Degree of Cryptographic Awareness

Cryptoki is layered directly on top of actual cryptographic tokens or algorithm libraries. As such, it requires a substantial degree of cryptographic awareness. The generality of Cryptoki makes it well suited as a foundation for higher level libraries such as the GCS-API.

Modular Design and Auxiliary Services

Cryptoki provides few additional services beyond cryptography, key management, and user authentication. In addition, Cryptoki provides extensive facilities for querying the capabilities of the cryptomodule, but very little for querying its internal state. (The state of a token might be supported as an attribute unique to that kind of token.)

Authentication of the cryptomodule by the application is not addressed by the Cryptoki specification. It is vital to have high confidence in the identity of its cryptosubsystem; therefore, the CAPI must provide an interface to allow an application to authenticate its cryptosubsystem. Cryptoki provides extension mechanisms, so such a function could be added as an extension; ideally, however, this should be specified in the actual standard.

Safe Programming

Cryptoki was not designed for novice C programmers. This CAPI uses the C language in a sophisticated, object-oriented way, and programmers using this CAPI will need substantial C expertise and familiarity with object-oriented programming concepts. This complexity could be a potential source of errors when developing security applications.

Modular Design and Auxiliary Services

Cryptoki supports basic cryptographic operations, such as hash, signature and encryption that in turn provide the required security services of integrity, authentication, and confidentiality. Signatures also provide some support for nonrepudiation services, although clearly a more complete architecture is needed for full support. Many of the basic cryptographic operations (e.g., encryption and key derivation) can be used as primitives in support of key management operations. Another operation that may be required for most tokens is ASN.1 decoding of certificates. (ASN.1 BER decoding could be required for extracting the subject's and issuer's distinguished name, public keys, or public key parameters from public key certificates.)

Access control is another important service that a CAPI may provide. One Cryptoki feature that provides some level of access control is the notion of public and private objects. Public objects are accessible after opening the token; private objects are accessible only after logging into (or initializing) the token. Perhaps, this is adequate for the single-user token environment for which Cryptoki was envisioned. In a multi-user environment, additional controls would be required to protect a single token from multiple applications making simultaneous accesses.

One of the stated design goals for Cryptoki is resource sharing. Cryptoki does utilize the notions of contexts and handles that are suitable for multitasking, multithreaded systems. The design of the Cryptoki CAPI is operating system independent (beyond the need for a C compiler). Lack of any operating system dependencies means the implementor would have to provide access management in the case where many applications share a single token. Ideally, to achieve good security an application using Cryptoki should require trusted operating system facilities for getting the user's PIN, which is a capability supported by Cryptoki to provide user authentication.

MISSI Support

One prototype implementation of the Cryptoki CAPI has been built at NSA⁴. Several commercial vendors have built Cryptoki interfaces for their tokens, including National Semiconductor and SCI. Nothing in the standard prevents exposing the full cryptographic functionality of the FORTEZZA token as Cryptoki functions and objects.

Safe Programming

Cryptoki is designed in an object-oriented fashion. The object-oriented design provides a large degree of data hiding, that is one of the major mechanisms to support safe programming. According to RSA Labs, later versions of the PKCS #11 standard may be split into a language independent specification and C bindings, but the current edition is written only for C. In addition, the Cryptoki specification uses consistent naming rules for routines, constants, and data types. However, these rules could be more explicitly codified in the standard. In addition, the strict data type controls and object-oriented nature of the Cryptoki make it suitable for implementation in advanced languages (e.g., C++, Ada, or JAVA).

4. NSA is providing the C source code for our implementation on a restricted and completely unsupported basis, to U.S. companies and institutions under a Technology Transfer Agreement.

8. RSA CRYPTOKI

8.1 Introduction

Cryptoki (“Crypto-Key”) is a member of RSA's Public Key Cryptography Standards (PKCS) family, specifically PKCS #11, that provides guidance to the commercial cryptography community. Cryptoki is standardized and distributed without charge by RSA Labs, the research arm of RSA Data Security, Inc. Cryptoki provides a standard lower level CAPI, primarily for access to personal cryptographic tokens. RSA realized that their existing commercial libraries were not flexible or general enough to support the needs of applications working with such devices, and therefore developed Cryptoki. Additional goals in the Cryptoki design include portability, extensibility, generality, support for resource sharing, and algorithm independence.

8.2 Strengths

The RSA Cryptoki specification meets a number of the evaluation criteria.

Algorithm Independence

Cryptoki is algorithm independent. While the draft standard does specify particular algorithms, key types, and algorithm specific object attributes for RSA-proprietary and public algorithms, the use of symbolic identifiers to specify algorithms in the CAPI allows any organization the capability to register new algorithms and add them to Cryptoki. Optionally, the organization can choose to submit their new algorithm and other identifiers for addition to the standard; RSA Labs has expressed willingness to make such additions. Unique NSA algorithms and attributes may be easily identified in this manner. This will allow applications to access attributes that are unique to our algorithms. Applications need not specify which algorithms are to be used for various cryptographic functions. If necessary, the application can query the cryptomodule to discover which algorithms are supported.

Application Independence

Cryptoki is highly application independent, because it is such a low-level CAPI and because it defines a highly abstract and general view of the cryptography provider. A cryptography provider, as a token, stores a collection of objects and can manipulate them to perform cryptographic operations. Because the application may directly invoke cryptographic operations on token objects and because it must specify the cryptographic algorithms to employ, Cryptoki is therefore best suited for cryptographically aware applications.

Cryptomodule Independence

Cryptoki meets the design goal of cryptomodule independence by defining an abstract token model that is implemented by the combination of the physical token and the Cryptoki library. Cryptoki was written with personal cryptographic tokens (e.g., smart cards, PC cards) in mind. Extension mechanisms are defined to allow the addition of new capabilities (i.e., functions, object types, etc.) by individual implementors. This provides a way, for example, to access functions specific to an NSA token like FORTEZZA (although other tokens written to Cryptoki may not support these extensions).

7.3 Weaknesses

CryptoAPI contains weaknesses similar to those in the RSA Cryptoki standard. Some of these weaknesses can be characterized as a function of being a lower level interface.

Degree of Cryptographic Awareness

CryptoAPI is a lower level interface designed to interface directly to the underlying CSP, either in software or hardware. As such, it requires a substantial degree of cryptographic awareness. CryptoAPI is sufficiently general for building higher level libraries such as the GCS-API or GSS/IDUP.

Modular Design and Auxiliary Services/ Cryptomodule Independence

CryptoAPI is weak in the area of key and certificate management, providing only the ability to export and import an opaque key blob. Presumably, key and certificate management would be left to the application or CSP to implement, which would defeat much of the interpretation of cryptomodule independence. Additionally, successful exchange of applications and CSPs becomes less likely. The recommendation to layer lower level APIs with a higher level API such as GCS or GSS/IDUP would minimize this problem. Other auxiliary services are beyond the scope of the CryptoAPI, although there appears to be sufficient flexibility for some, such as user authentication, to be implemented.

Safe Programming

CryptoAPI was not designed for novice C programmers. Programmers using this CAPI will need substantial C expertise and cryptographic programming expertise. Efforts to abstract the CryptoAPI interface into C++ or Visual Basic objects have been demonstrated; however, they do not reduce the level of cryptographic programming expertise required for a good implementation. CSP developers will also need expert programmers familiar with the process and security models of Microsoft's operating systems.

7.4 Current Status

Microsoft has implemented the CryptoAPI for version 4.0 of their Windows NT operating system. With the initial implementation, they provide a default CSP that implements several commercial cryptographic algorithms, including RSA and MD5. Several companies have announced plans to implement CSPs for various software libraries and hardware tokens. Most notably, Spyrus Inc. has announced their development of a CSP for their LYNX EES card that supports the FORTEZZA algorithm suite.

them be interchangeable. However, it appears that Microsoft's programming paradigm assumes that an application written to the CryptoAPI would need to know the specifics of the underlying cryptography.

Provider types include related algorithms, such as digital signature, key exchange, and encryption algorithm. Ideally, applications should be written to a standard so that implementations of a provider type would be interchangeable. Microsoft has not completed the definition of such a standard. Presumably some popular provider types, beyond those such as RSA_FULL and DSS_SIGNATURE, will be defined as a result of market pressures. This leaves room for third-party implementors to extend functionality.

Modular Design and Auxiliary Services

CryptoAPI embodies the concepts of modular design, separating encryption and decryption services from signature services and hashing services. Auxiliary Services that are provided include cryptomodule authentication, query and set capability, and CryptoAPI session setup and tear-down.

MISSI Support

MISSI support is provided through CryptoAPI's ability to support multiple cryptomodules and cryptomodule independence. FORTEZZA support could be provided by creating a CSP that describes access to hardware implementations of KEA, DSA, and SKIPJACK.

Safe Programming

Safe programming is supported through the use of opaque handles to access CryptoAPI data structures. Intuitive names are used for functions, further supporting the concept of safe programming. Furthermore, a rich error reporting feature is available to notify the application of errors and exceptions that have occurred.

Security Perimeter

Implementors can take full advantage of the process isolation mechanisms provided by the underlying operating system to support a robust security perimeter. The actual CryptoAPI interface layer is embedded in the operating system and cannot be easily circumvented by the application developer. The CryptoAPI interface is designed to not propagate or provide direct access to sensitive information. Implementation of these features is largely a CSP developer's responsibility.

7. Microsoft CryptoAPI

7.1 General

CryptoAPI is a low-level interface that abstracts the cryptographic module, referred to as the Cryptographic Service Provider (CSP), to a standardized programming interface. CryptoAPI defines certain objects and data structures in an abstract method, leaving the actual interpretation of those objects and structures to the CSP. Microsoft defines the concept of cryptographic provider types. Each type may have multiple providers registered with the operating system, with one provider assigned as the default provider for that class. This hierarchy allows applications to be written to a specific provider, or to a generic type of providers. Provider types may be implemented in both hardware and software. Microsoft has defined the minimum capabilities of several predefined provider types. The available query capability allows applications to discover additional capabilities beyond the minimum defined.

Microsoft's implementation is embedded as an operating system service, providing the ability for CSPs to take advantage of the native operating system security features in their implementations. In addition, other operating system services can take advantage of the CSPs for additional protection of the operating system. This feature will not be addressed in this recommendation.

7.2 Strengths

Microsoft's CryptoAPI meets several evaluation criteria.

Algorithm Independence

CryptoAPI is algorithm independent. The function descriptions included in the CryptoAPI specification are written in the context of the default Microsoft provided CSP, RSABASE.DLL implementing the RSA_FULL set of algorithms. However, the data types are abstract and can be interpreted as required by the underlying cryptographic mechanism. Microsoft has explicit support for multiple cryptographic algorithms with the choices being application selectable.

Application Independence

CryptoAPI is application independent because it is designed to provide a generic interface to cryptographic services such as encrypt and decrypt. The vendor implemented services may be application specific (such as the encode and encrypt functions for the Message Security Protocol (MSP)), but that is beyond the scope of the CryptoAPI specification.

Cryptomodule Independence

CryptoAPI meets most of the cryptomodule independence requirements. The CryptoAPI allows for current and future cryptomodules to be implemented. Additionally, the CryptoAPI shields an application from knowing if a particular implementation is embodied in software or hardware. An application's ability to use multiple cryptomodule implementations is a function of the cryptomodules' adherence to a common interpretation of the CryptoAPI interface. It is possible to write two different cryptomodules implementing the same cryptography and have

Modular Design and Auxiliary Services

Authentication of the cryptomodule by the application is only vaguely specified by the GCS-API specification. It is vital to have high confidence in the identity of its cryptosubsystem; therefore, the CAPI must provide an interface to allow an application to authenticate its cryptosubsystem.

Safe Programming

GCS-API can be invoked in fine enough granularity so as to do more harm than good to the protection of data, if calls are improperly sequenced or parameter values are poorly chosen. The power to customize security applications comes at the price of some peril to the cryptographically unaware programmer.

Security Perimeter

GCS-API will export unprotected keys to applications that have access to them. Our goal is that no unprotected key ever be passed outside the security perimeter, and that the CAPI mechanism will act as the application's agent for manipulating objects within the security perimeter. This is not a condemnation of the X/Open design, but instead limits the applications that should use the interface to those providing security services. The implementor of GCS-API will need to take great care to ensure that the access control mechanism limits the invocation of clear-key producing calls to authorized parties.

6.4 General

A CAPI at this level of cryptographic awareness is designed to be used as an underlying layer to higher level CAPIs like GSS/IDUP, or for security applications that use cryptography in ways not accommodated by higher level CAPIs. GCS-API has come to a stable point in its development, and hence is ready for serious consideration as a CAPI that has some of the characteristics of both high-level and low-level CAPIs, depending on who uses it and what functions they are authorized to use, and how its cryptographic context library is configured. Despite this stability, middle-level CAPIs have not found the market support yet as compared to both high-level and low-level offerings. The natural reason for this is the narrow niche of security applications that such a CAPI supports. Once these applications exist and when marketability becomes clearer, the use of GCS-API should be confined to those developing security service applications.

These opaque handles support the ability to control the access by callers to cryptographic contexts and facilities. Utilizing cryptographic contexts from a library provided to the developer from a cryptographically aware source provides some assurance that calls will be invoked in the proper order, since certain algorithms contained therein imply automating and ordering of cryptographic operations. In addition, for calls that must process multiple buffers of data while maintaining cryptographic state between calls, a chaining flag is available.

Modular Design and Auxiliary Services

The GCS-API provides access to cryptographic algorithms to support the core set of cryptographic services: authentication, data integrity and confidentiality. GCS-API also contains a robust set of protected and unprotected key management calls, which is a unique characteristic of GCS-API that many other CAPIs do not possess. The placement of GCS-API as a middle-level CAPI also enables additional auxiliary security service APIs, such as a certificate management API or authentication API, to be readily included.

The GCS-API specification now contains guidance for implementors to prepare them for incorporation of cryptomodule authentication. This is currently only vaguely specified, as it is unclear whether a separate API will be needed or just the functionality embedded somewhere, as in the `gcs_initialize_session` call. It is felt that inclusion of such guidance is wise since this kind of authentication could affect the markets within which CAPI software, cryptomodules, and applications can be used.

MISSI Support

Currently available MISSI cryptographic hardware can be used as a cryptomodule under GCS-API, but only if the GCS-API conformance policy is lenient to the extent that not all calls must be implemented. Specifically, no unprotected keys can be output from our cryptography. Otherwise, mapping GCS-API calls to our cryptographic calls in both directions can be accomplished.

6.3 Weaknesses

GCS-API enables an application developer to choose from a highly automated and safe design requiring minimal cryptographic knowledge, to a highly configurable but unsafe design that requires detailed cryptographic knowledge. The latter choice is an area where aspects of GCS-API, if used improperly by a cryptographically unaware developer, could result in weaknesses in the data protection.

Degree of Cryptographic Awareness

Because GCS-API potentially requires a cryptographically aware developer, it is not recommended for general applications. The direct use of GCS-API should be the exception, to be used primarily by cryptographically aware applications and developers. (It should be noted that with proper configuration of a cryptographic context library, and access control on the APIs, GCS-API can potentially be safe for cryptographically unaware programmers.)

6. X/OPEN GCS-API

6.1 Introduction

The Security Working Group (SWG) of the X/Open Consortium has created a set of APIs and mechanisms to provide security services (primarily cryptographic and key management services) to applications. GCS-API is designed for both cryptographically aware and unaware applications. GCS-API provides cryptographic contexts facilities that can be used in the development environment in such a way as to require a lesser degree of cryptographic awareness on the part of application developers. In general, the use and configuration of GCS-API require a relatively high familiarity with cryptography. Applications developed with GCS-API are responsible for at least the basics of the operation of cryptographic algorithms and key management. Important inputs to this specification include the X/Open Distributed Security Framework, NIST's proposed FIPS-Cryptographic Service Calls, and contributions from IBM and ANSI X9F1. After eight drafts, GCS-API is currently at its first official status as a "Preliminary Specification." This assures a solid, stable specification that can now have multiple independent implementations tested in order to become part of the X/Open branded open system, "Common Applications Environment."

6.2 Strengths

The GCS-API meets a number of the evaluation criteria.

Algorithm Independence

GCS-API calls do not depend on the use of any particular algorithms. The developers of GCS-API intend to support all known classes of algorithms including secret key and public key based ciphers, key exchange algorithms, and key management systems. Using GCS-API, applications may specify algorithms to be used by building cryptographic contexts that stipulate the use of those algorithms. Alternatively, predefined cryptographic contexts may be used from a library to get the desired algorithm, effectively giving a certain QOP to the protected data.

Application Independence

GCS-API is aimed at both cryptographically aware and unaware applications. GCS-API can support the mechanisms used to implement security specific applications for both session-oriented and store-and-forward applications. GCS-API can also support the underlying mechanisms of higher level C APIs (i.e., GSS/IDUP).

Cryptomodule Independence

GCS-API may be implemented on top of any cryptographic technology, cryptomodule, or algorithm. However, as discussed in Section 2, it is recommended for GCS-API to use of lower layer cryptographic module or token interface, as discussed in Section 7 and 8.

Safe Programming

The GCS-API interface uses opaque "session contexts" to access the cryptographic contexts containing the appropriate key and algorithm structures. This enables information hiding.

5.4 General

The initial GSS-API presented in RFC 1508 is not complete and will not support the wide range of generic applications that are being considered when analyzing the CAPI candidates. Fortunately, the CAT working group has taken steps to expand the GSS-API interface to support applications that do not fit into the session-oriented paradigm. If the current RFC 1508 is superseded by the new draft GSS-API and merged with the IDUP extensions, then the IETF will produce a high-level API for security services that we should adopt for use in the majority of our applications that require cryptographic services but are not trusted with cryptographic knowledge. Eventually, it is envisioned that as security support services and mechanisms evolve, that most applications will need to use only a small subset of the GSS/IDUP calls, while the credential management and support calls are incorporated into their own APIs. Until this occurs, all of the GSS/IDUP calls will be required.

security perimeter, GSS/IDUP will reduce the amount of sensitive security information that is exposed to the application. Most data structures internal to the cryptographic module are identified by the application as opaque pointers. In this way, the application is prevented from having access to the data itself. Also, since GSS/IDUP is at a fairly high level, keys and other sensitive cryptographic data are never exported beyond the CAPI mechanism.

5.3 Weaknesses

The following GSS/IDUP weaknesses were found. However, one should keep in mind that the criteria were applied to GSS/IDUP being implemented alone and not with other lower level CAPIs. Once a CAPI is used in conjunction with other CAPIs, then some of these weaknesses actually become strengths.

Application Independence

The GSS-API works in a session-oriented paradigm, and does not support store and forward applications or applications with multiple receivers. The initial specification could be forced to work in these scenarios by modifying the underlying mechanisms and changing the target name to represent an alias or group, but this approach is not very clean. However, the IDUP-GSS-API does address the issue of store and forward applications, including multiple receivers. This is why we recommend the GSS/IDUP combination. Ideally, we would like to see these two specifications merged into one.

Modular Design and Auxiliary Services

Authentication of the cryptomodule by the application is not addressed by the GSS/IDUP specification. It is vital to have high confidence in the identity of its cryptosubsystem; therefore, the CAPI must provide an interface to allow an application to authenticate its cryptosubsystem.

The GSS/IDUP specification includes support calls based on structured programming languages. This aspect would not lend easily to an object-oriented implementation. While the underlying support is critical to an implementation, it is not important in specification documentation.

MISSI Support

Since the GSS/IDUP interface does not include auxiliary security services like user logon, it is more difficult to ensure MISSI support in a system that uses GSS/IDUP than one which uses a CAPI that provides that service. To ensure compatibility, other security services must be considered in conjunction with GSS/IDUP. Even within the scope of cryptography, the GSS/IDUP interface does not export some of our legacy capabilities to the application. For example, the application cannot authenticate the cryptographic module below the interface or access the key management capabilities of our current cryptography. It is envisioned that access to services like key management and the cryptomodule authentication would be performed through a Key Management API or a lower level CAPI like GCS-API, CryptoAPI, or Cryptoki.

creation of that context, the mechanism below the GSS/IDUP will determine whether to create the requested security context, deny the context, or grant the context with other mechanisms being used. The security policy decision process is beyond the scope of this document.

Cryptomodule Independence

GSS/IDUP is a higher level CAPI and hides the fact that the cryptographic mechanism is implemented in hardware or software. The only reference the application has to the cryptographic mechanism is an opaque handle that represents the context created by the CAPI mechanism. Each security context may utilize more than one cryptographic module to protect the information. Also, since security contexts are represented by opaque objects and not the identity of the application, each application may have numerous concurrent security contexts each potentially using different mechanisms.

Degree of Cryptographic Awareness

The GSS/IDUP has a minimal set of routines. The application using the GSS/IDUP can be cryptographically unaware. Although a few of the GSS/IDUP calls have a number of parameters, most can be set to specify default actions or services, thus reducing the amount of information the application must provide for the calls.

The security architecture used in the IETF CAT working group provides access to the basic cryptographic services: authentication, data integrity, confidentiality, and nonrepudiation; it did not, however, include a number of the security services seen in other CAPIs. GSS/IDUP does not provide a direct interface for services like user authenticated logon, key management, access control, and the storage and access to security database information. Applications that need to access these services will be cryptographically aware applications; therefore, they must utilize another API designed for the specific service (e.g., GCS-API, CryptoAPI, Cryptoki).

Modular Design and Auxiliary Services

The GSS/IDUP is split into four groups where each group has a small set of well-defined calls. The first group provides credential management; the second provides security context management/environmental-level calls; the third provides the per message/per IDU calls; and the fourth group is the miscellaneous support calls. In addition, IDUP-GSS-API provides a special-purpose call in support of nonrepudiation. In each case, the ordering of the parameters and the naming of functions and parameters follow the same rules.

Safe Programming

By using opaque objects to identify sensitive information like credentials, contexts, and other complex data structures, applications can reference the information without having access to the sensitive information itself. Call sequence is ensured by the fact that the GSS/IDUP requires credential pointers to do context operations and context pointers to do the per-message calls.

Security Perimeter

In systems that separate the client applications from the cryptographic service provider with a

5. GSS-API/IDUP-GSS-API

5.1 Introduction

The Common Authentication Technology (CAT) working group of the Internet Engineering Task Force (IETF) has been creating a set of APIs and mechanisms to provide security services to application programmers. The initial offering from this group was a suite of RFCs (Request For Comments), numbers 1508-1511, that was published in late 1993. These documents outlined the initial Generic Security Service API, the C bindings for the API, the Kerberos authentication protocol and a CAT overview. Since that time, GSS-API has received widespread attention as a number of development activities attempted to utilize the interface to access security services in the system. As is so often the case, deficiencies in the initial RFC have been identified, and work is continuing on a second version of the GSS-API along with a separate companion draft specification to support Independent Data Unit Protection (IDUP). Our recommendation is for the cryptographically unaware application to use both the GSS-API and IDUP-GSS-API (referred below as GSS/IDUP).

A useful concept provided by GSS/IDUP is that of security contexts. A security context is the relationship established between peers, using credentials established locally in conjunction with each peer or received by peers via delegations[2]. While our investigation focused on cryptography, it is important to realize that GSS/IDUP is more than just a CAPI. GSS/IDUP is a Security Service API³ (SSAPI) that provides the means to access other security services, notably authentication.

Since the work within the IETF has transpired without our involvement, there was little effort to ensure that our requirements can be met in the API. This section presents the features of GSS/IDUP that met our criteria and those features that did not. Since our original recommendation, version two of the GSS-API has continued to mature to become quite an impressive SSAPI. Noticeable changes between draft one and five can be found in APPENDIX D.1. Likewise, IDUP-GSS-API has also been maturing since our original recommendation. IDUP is currently at draft four and has grown considerably in detail as well as complexity. Changes between drafts one and four were numerous and can be found in APPENDIX D.2.

5.2 Strengths

The GSS/IDUP specifications meet the following evaluation criteria.

Algorithm Independence

GSS/IDUP is cryptographic algorithm independent as well as security mechanism independent. This allows applications to be completely unaware of the algorithms providing the protection. In cases where an application or user is knowledgeable of the available cryptographic capabilities, the interface supports the ability to specify a QOP parameter that indicates the desired level of protection when requesting a security context. During the

3. A Security Service API provides access to a variety of security services, not just cryptography.

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 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.

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

4. 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

Additional functionality that could be supported by a CAPI but is not visible to the application includes access control (to specific functions and/or algorithms), security event generation, and auditing. Detailed analysis of this functionality is beyond the scope of this document.

Industry and national standards bodies are working on CAPI standardization as well. Besides the work being done with respect to the four recommended CAPIs, ANSI X9 (Financial Services) is proposing a layered architecture similar to that described here, and the US National Institute of Standards and Technology (NIST) is also considering the benefits of a layered set of standards. Coordination with standards bodies is essential in encouraging these CAPIs to be widely accepted by the commercial community. Additionally, as these CAPIs become standards they should be moved into the IEEE POSIX standards process to simplify procurement.

3. BACKGROUND

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

The Generic Security Services API (GSS-API) [2,6] and the extensions for independent data unit protection (IDUP-GSS-API) [3] support cryptographically unaware applications. This CAPI, described in Section 5, provides a high-level interface to authentication, integrity, nonrepudiation and confidentiality services. The application merely indicates the required security services and optionally the quality of protection (QOP). GSS-API was designed to protect session-style communications (e.g., ftp) with other entities. IDUP-GSS-API does not assume real-time communications between sender and recipient, and protects each data unit (e.g., file or message) independently of all others. IDUP-GSS-API is therefore suitable for protecting data in store-and-forward applications.

The Generic Crypto Services (GCS-API)[4], described in Section 6, supports both cryptographically aware and unaware applications. GCS-API allows a caller to establish a cryptographic context that uses one or more cryptographic modules to protect data. The GCS-API supports protection of data on a per buffer basis. This enables GCS-API to support all types of applications. GCS-API does not require (nor preclude) knowledge of specific algorithms. GCS-API is being developed within the Security Working Group of X/Open. Historical inputs to this document include the X/Open Distributed Security Framework, NIST's proposed FIPS-Cryptographic Service Calls, and contributions from IBM and ANSI X9F1.

The Microsoft CryptoAPI[10], described in Section 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. 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 crafting 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.

Cryptoki[5], described in Section 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 application. Note Cryptoki is the only CAPI that 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.

The background of the four CAPIs is discussed in Section 3. The criteria for comparing the evaluated CAPIs are discussed in Section 4. The four recommended CAPIs are examined in Sections 5 through 8. These CAPIs are recommended because they fulfill our criteria, seem likely to survive in the standards process, or because they have gained widespread vendor and user interest.

Section 9 concludes with our multitiered CAPI suite recommendation. In general, less knowledge of cryptography is required to use the higher level CAPIs, while applications which are “cryptographically aware” may call upon lower level CAPIs for finer granularity of control over cryptographic subsystems.

Sections 10 and 11 provide acknowledgments and references. The appendices include a list of acronyms, a glossary of terms and CAPI application scenarios.

2. SCOPE

This document considers a set of C APIs to be used with COTS applications for use in systems integration work. The C API insulates 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. The C API also greatly simplifies the process of incorporating various cryptographic algorithms to be used by an application.

A concept of cryptographic awareness has been used by several C API 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 implementor 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 C APIs. Using this metric, along with the level of abstraction with respect to our four recommended C APIs, produces the layering scheme illustrated in Figure 1.

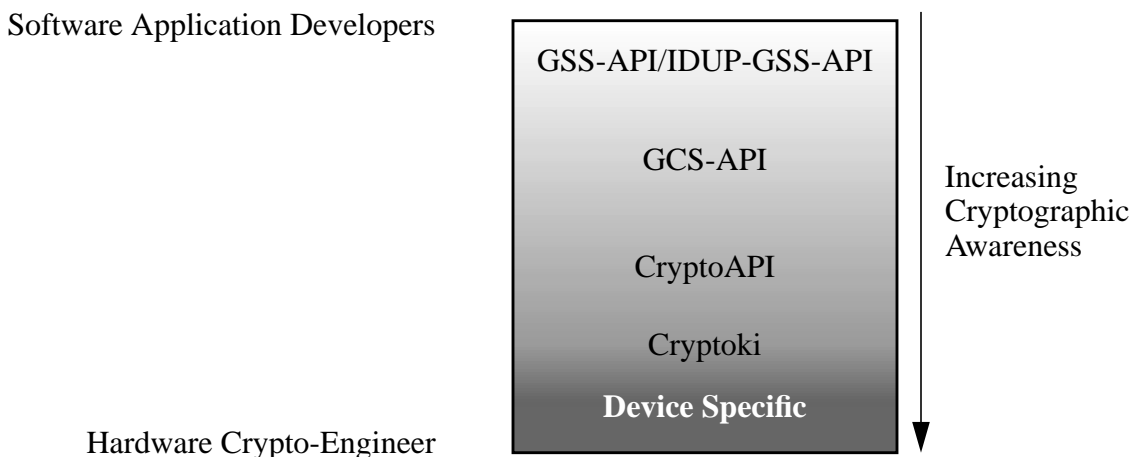


Figure 1: Levels of Cryptographic Awareness

1. INTRODUCTION

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 needed for commercial products[1]. A technique that can provide a much more flexible and powerful alternative is the use of a standardized CAPI.

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 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. This also reduces algorithm licensing fees when nonproprietary algorithms are desired. There are numerous efforts currently under way to create CAPI standards. They range in scope from very generic cryptographic support like that found in GSS-API to an interface more involved in the direct control of the cryptographic module (i.e., cryptographic token¹) like that in Cryptoki. A number of these CAPI efforts are receiving a great deal of support as applications are being written to use them. While we would ideally select a single CAPI standard for all applications, multiple CAPIs are required to support the broadest range of applications.

Increases in internal development costs, along with increased availability of commercial cryptography, have changed how implementors should integrate cryptography with 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-aware commercial, off-the-shelf (COTS) applications to a standard subset of CAPI mechanisms, we must select and use a subset of the CAPI standards, encouraging other CAPI efforts to align with one of the standards in this subset.

In deciding which subset of CAPIs to use, many CAPIs were considered; some could not be included because of their company’s proprietary nature and others because of their lack of maturity. The remainder of this document compares the selected subset of CAPIs highlighting their strengths and weaknesses.

1. Token is used within this recommendation to mean a module that implements cryptographic functions. This definition is consistent with that presented in RSA’s Cryptoki CAPI.

Security Service API: Cryptographic API Recommendation Second Edition

NSA Cross Organization CAPI Team

July 1, 1996

EXECUTIVE SUMMARY

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[1]. An approach that can provide flexibility and cost savings is the use of a standardized Cryptographic Application Program Interface (CAPI) suite.

The compelling case for a modular cryptographic interface has given rise to the development of numerous CAPI proposals. An NSA cross-organizational team was formed to assess the ability of these proposals to meet anticipated needs. After an initial review, the CAPI evaluation criteria were developed, and a detailed analysis of a subset of the available proposals was performed. The recommendation by the team and the findings of the analysis were originally reported in June of 1995. Published one year after the initial report, this new edition constitutes an updated recommendation, and it has been expanded with more detailed analysis, more examples, and with coverage of Microsoft's CAPI.

Rather than recommending a single CAPI, the team concluded that combinations of four widely accepted CAPI proposals be adopted. These proposals include the GSS-API (Internet Engineering Task Force), the GCS-API (X/Open), CryptoAPI (Microsoft), and PKCS #11 Cryptoki (RSA). These CAPIs were designed to support significantly different levels of cryptographic awareness, ranging from minimal cryptographic awareness to extensive knowledge of the underlying cryptography. Additional criteria used by the team to assess each of these CAPIs included algorithm independence, application independence, cryptomodule independence, modular design and auxiliary services, MISSI cryptographic support, safe programming, and security perimeter design approach.

Even though a CAPI suite was recommended to address the needs of a wide variety of applications, it is anticipated that most applications will require minimal knowledge of the underlying cryptography. Therefore, the high-level GSS-API should be selected for use whenever practical.