Common Cryptographic Architecture Cryptographic Application Programming Interface

by D. B. Johnson
G. M. Dolan
M. J. Kelly
A. V. Le
S. M. Matyas

Cryptography is considered by many users to be a complicated subject. An architecture for a cryptographic application programming interface simplifies customer use of cryptographic services by helping to ensure compliance with national and international standards and by providing intuitive high-level services that may be implemented on a broad range of operating systems and underlying hardware. This paper gives an overview of the design rationale of the recently announced Common Cryptographic Architecture Cryptographic Application Programming Interface and gives typical application scenarios showing methods of using the services described in the architecture to meet security requirements.

Possibly there once was a time when information was not considered a business asset, but such a time is no more. Protecting enterprise information is becoming more and more essential in maintaining an organization's competitive edge. Cryptography is one element of a set of basic data security measures that provide security for information assets. Other measures include physical and logical access control, identification and authentication mechanisms, security management, and journaling and auditing procedures. Data security measures such as access control and auditing may be adequate for applications where all information assets are on site and within the control of the enterprise. However, for applications where data must reside outside direct management control, or are shared among many users, prudent data processing installations may require the additional protection that cryptography provides.

Cryptography is the transformation of intelligible information into apparently unintelligible form in order to conceal information from unauthorized parties. Cryptography is the only known practical method to protect information transmitted electronically through communication networks. It can also be an economical way to protect stored data. Cryptographic methods can be used to protect not only the confidentiality of data, but the integrity of data as well. Data confidentiality is the protection of information from unauthorized disclosure. Data integrity is the protection of infor-

mation from unauthorized modification. The cryptographic transformation of data is defined by a cryptographic algorithm, or procedure, under the control of a value called the cryptographic key. The cryptographic key must be protected against unauthorized use, as the security of the cryptographic transformation depends on the secrecy and integrity of the key. Cryptography can therefore be viewed as a method to logically reduce the area that needs protection to ensure data confidentiality and/or data integrity to the area where the cryptographic keys are kept.

There are many applications today for commercial cryptography. Existing applications provide confidentiality of files and communications, integrity of messages, and user identification. Cryptographic services are implemented in disparate products.

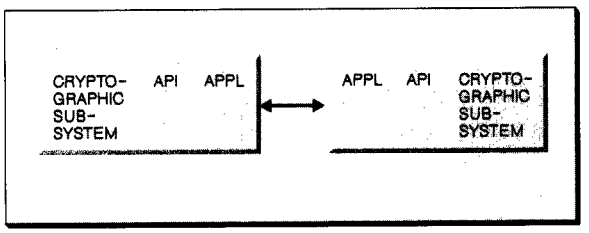
Software architecture rationale

Uses of cryptography. Files may be encrypted locally and decrypted either locally or on a remote system at an indeterminate time in the future. This requires that the cryptographic algorithm be compatible in the two systems and that the key used for decryption be available at the remote system. Some applications may require the encipherment of partial or full communications between systems. This requires the use of a common cryptographic key-management and key-exchange approach.

Communication integrity may be guaranteed by using a message authentication code (MAC) to detect alteration of messages transmitted over networks. Conceptually, a MAC is a cryptographic checksum that is derived using a key that is known only to the parties involved. The MAC is generated at the message origination node and appended to the message. On receipt of the message and its associated MAC, the MAC is verified at the message destination node.

Another use is the detection of unauthorized file modifications. A modification detection code (MDC) calculation may be used to reduce the problem of maintaining the integrity of an arbitrarily large data object to that of maintaining the integrity of a 128-bit quantity. For example, to detect program alteration, an MDC could be calculated on a known correct version of the program and then published in a generally available forum, such as a newspaper. A recipient of the program can then calculate the MDC for the received program and verify that the resulting MDC is the same. An MDC is different from a MAC in that the algorithm for calculating an MDC does not use a secret key and is public knowledge.

Figure 1 Cryptographic API model



Financial applications require the use of a PIN (personal identification number) for user authentication. The cryptographic support at various places in the network must provide for encryption, translation, and verification of PINs.

Cryptographic services are tailored for the environment where they will operate. However, they should perform the same operation, with the same results, regardless of the product or the environment. If a customer uses a workstation-based product for end-user cryptographic services, it should be compatible with host-based products that provide the same services. In addition, if a customer writes an application that requires cryptographic services, the application should be portable between the IBM strategic operating systems. All the above considerations result in the conclusion that a cryptographic architecture is both necessary and desirable.

Cryptographic API model. A model for the Common Cryptographic Architecture Cryptographic Application Programming Interface[1] (Cryptographic API) is shown in Figure 1. The cryptographic subsystem consists of all cryptographic functions below the Cryptographic API. The customer or system application (APPL) calls API services to provide the cryptographic transformations. The application typically transmits the encrypted information to an application on another system which in turn calls appropriate API services to achieve the desired security objective.

Cryptography and computer systems security

Security is an increasingly important aspect of the enterprise information system. The evolution of the personal computer, open networks, inter-enterprise connections, and computer viruses has made security mandatory for the computer systems of an enterprise.

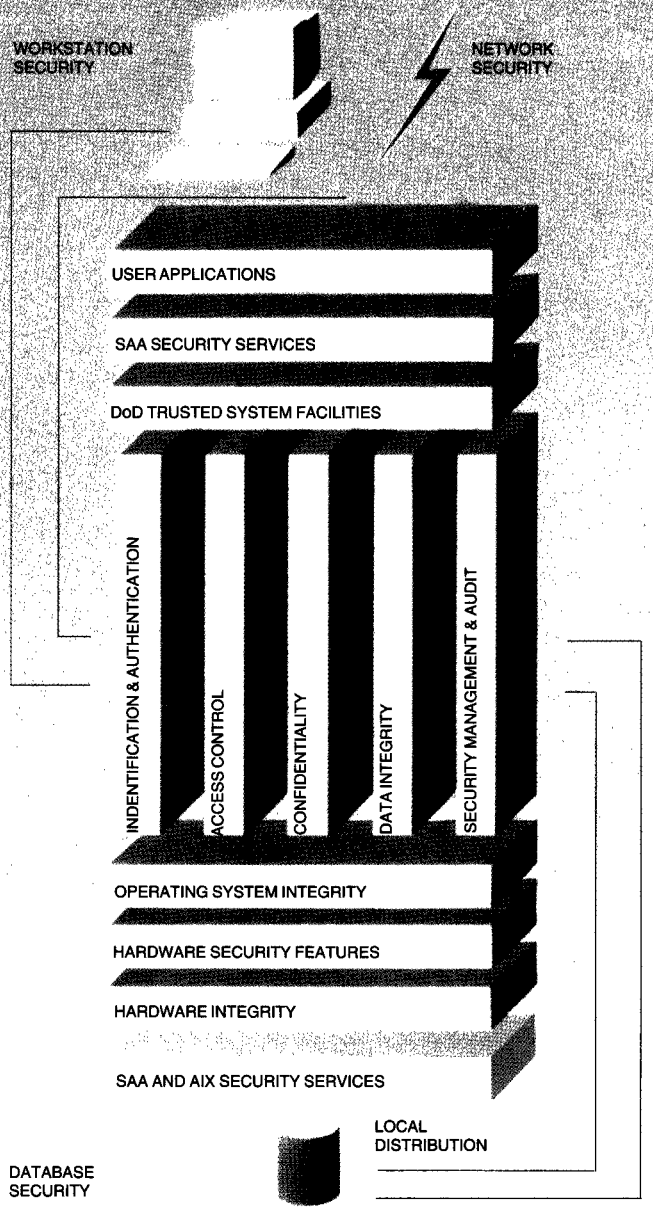
Cryptography is one of the major security mechanisms that comprise IBM's systems security strategy and architecture. IBM has been a leader in computer security since 1973 when a corporate policy letter regarding commitment to privacy and data security was issued. Product implementations followed, such as the Resource Access Control Facility (RACF) in 1976, the Multiple Virtual Storage (MVS) Operating System Integrity in 1977, and the evolution of the Data Encryption Algorithm and Data Encryption Standard (DES). Today, over 60 IBM products utilize DES, and this acceptance of DES led to work on the Common Cryptographic Architecture (CCA) to facilitate interoperability across the various evolving product implementations.

System security structure

In October 1989, IBM announced its structure for implementing system security. The model of this structure, shown in the accompanying figure, consists of the following layers of security facilities with related mechanisms and evolving structured interfaces.

System integrity is a property of an operating system and the cornerstone of the security of a system. It is defined as the ability of the operating system to prevent the circumvention or bypassing of its security mechanisms and depends on the existence of certain unique environmental hardware integrity mechanisms and security features. Each system requires a specific set of interfaces and documented guidelines that enforce system integrity. As a property of the operating system, system integrity must be incorporated in the design during the development process and continually be adhered to as the system evolves. Over the years IBM has published its system integrity guidelines and committed to accept Authorized Program Analysis Reports (APARs) where a specific program function is found to violate the integrity of the system.

Identification and authentication (I&A) is a set of procedures to verify the identification of users to a system and network before access is granted to protected resources. A user is identified by a system-unique identifier (userid). Authentication of a user can be accomplished by a number of different techniques. Current technology includes the long-standing password; challenge and response tokens, or one-time password generators for enhanced authentication in networks; smart cards, which offer significantly greater capability in authentication; and biometric recognition, such as signature dynamics, which offers the highest level of authentication. The technology of user authentication has been characterized as being a combination of what you know, what you possess, and what you are.



System security structure

Access control protects resources from unauthorized disclosure or modification. Depending on the security policy of the enterprise, resources may be subject to either discretionary or mandatory access control. Discretionary access control limits users to accessing only those re-

sources for which they have been explicitly or implicitly identified, to the degree of priviledge registered for them in the access control list associated with the resource. Mandatory access control extends discretionary control and restricts users to accessing only resources for which they have a corresponding level of clearance, based on how resources are classified according to enterprise-defined levels and categories of security.

Confidentiality is another means to prevent information disclosure and is implemented via the use of cryptographic mechanisms. When sensitive data, such as business financial or personnel information, are stored on line, archived to off-line media, or transported across a network, confidentiality through the use of cryptography techniques is utilized. DES has become the industry-accepted standard for cryptography, and as it has been installed, specific techniques for the distribution and management of encryption keys between users and across networks have developed.

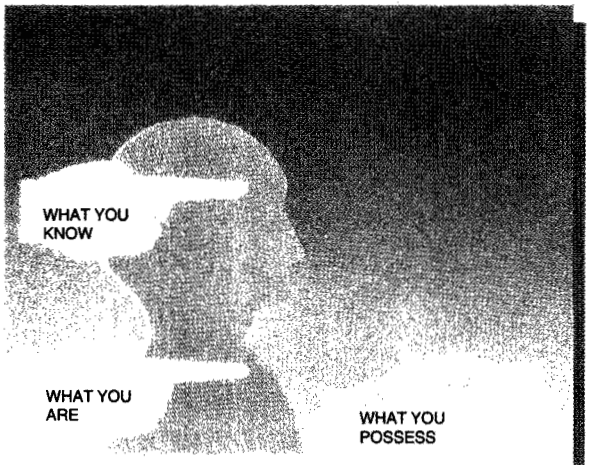
Data integrity is the means to detect the modification of data stored locally or transmitted over a network. Two types of data integrity services exist: the detection of intentional modification, which is provided by cryptographic functions such as message authentication code (MAC) or modification detection code (MDC), and the detection of accidental modification, which is provided by noncryptographic mechanisms. The CCA implements both the MAC and MDC functions. The use of a cryptographic function like MDC has proved effective in discovering and combatting computer viruses.

Security management encompasses registration and enrollment of users and resources in the enterprise information base, the facilities to administer security in various combinations of both centralized and decentralized rights-management based on the security policy of the enterprise, and the mechanisms to audit various levels of security-relevant events. Audit further breaks down into the detection, logging, and reporting of events as well as the invocation of security alerts for immediate action. As the only mechanism to validate verified access to protected resources, audit complements I&A and access control.

Department of Defense (DoD) trusted-system facilities are enhancements to the basic security facilities and are designed into the system to support the various levels of trust defined by the U.S. Government in their trusted-system criteria. In certain systems IBM has incorporated the functions to meet these criteria, whereas in other cases it has designed and implemented the functions, related documentation, and assurance material to pursue an evaluation at a particular level of trust. The major existing levels for which IBM systems have qualified are the "B1" multilevel security class that enforces labeling, mandatory access control, and full audit, and the "C2" class that enforces discretionary access control.

Systems Application Architecture™ (SAA™) and Advanced Interactive Executive™ (AIX™) security services are based on the underlying security facilities in each environment and will evolve to provide consistent services and interfaces to their base security mechanisms. As these services and interfaces evolve, an underlying software and, possibly, hardware architecture would be defined and consistently implemented across the SAA platforms, thus providing interoperability with the AIX family of systems. These architectures would then be reflected through to the user application as Common Programming Interfaces.

These security facilites and mechanisms, although embodied in the host operating system software, are also designed to be utilized in delivering specific network, database, and workstation security capability.



Characterizing user authentication technology

In September 1990, IBM expanded its commitment to security with the announcement of plans for an enterprise-wide security architecture to encompass the CCA, SAA, and emerging technologies required in a distributed computing environment. Further enhancements were also made to the various security facilities across the SAA and AIX systems.

CCA defines an application programming interface for a wide variety of confidentiality and data integrity functions that are available at both the application program and subsystem level. Currently, MVS/ESA™, VTAM™, IMS/ESA™, and CICS/ESA™ provide support for this interface to CCA based on the newly announced ES/9000™ Integrated Cryptographic Facility. In addition, the PC DOS workstation-based Transaction Security System implements this interface as part of a comprehensive finance industry security system. The new Virtual Machine/Enterprise Systems Architecture™ (VM/ESA™) product is currently implementing support for the Integrated Cryptographic Facility part of a guest virtual machine environment running MVS/ESA.

Curtis L. Symes

Terminology used in examples. In all examples in this paper the following names are used. The name *Ann* is used for the creator or sender of a crytographic key or other output of a Cryptographic API service. The name *Bill* is used for the intended recipient of the key or other output. Users at an intermediate node use a name that starts with the letter *C*, for example, *Charles*. The name used for an arbitrary adversary attempting to penetrate the security mechanisms is *Eve*.

The Common Cryptographic Architecture Cryptographic API definition uses a common key-management approach.

Common Cryptographic Architecture. The need for a Common Cryptographic Architecture Cryptographic API is derived by the applications that require cryptographic services. These applications are, in general, distributed across many IBM products. It is necessary to provide the capability to encipher data in one product, send the data to another system either directly or via a network, and decipher the data in the destination product. To ensure this capability, there must be a common set of cryptographic operations available on multiple systems. If the cryptographic operations are provided externally as common callable services on the disparate systems, then the additional advantage of enhancing program portability is achieved.

The Common Cryptographic Architecture Cryptographic API definition uses a common key-management approach and contains a set of consistent callable services. This allows for the implementation of cryptographic solutions that are independent of the networks and processing systems. Common key management ensures that all products that conform to the architecture allow users to share cryptographic keys in a consistent manner. The definition of key management provides methods for initializing keys on systems and networks, and also supports methods for the generation, distribution, exchange, and storage of keys.

The callable services provide a common high-level language interface for user or system applications. Thus, a small machine application could use the same service calls as a large machine application. The services provide a common set of functionality that is applicable to a wide variety of applications. The Common Cryptographic Architecture Cryptographic API services define a level of cryptographic capability that allows programs to be developed that work on disparate systems. In particular, the Common Cryptographic Architecture provides two forms of compatibility for applications: interoperability and portability.

Interoperability is the assurance that applications that use the architected services will work together. Interoperability is achieved by common encryption and decryption algorithms, common key-management definitions, and common external information formats. Interoperability has the following limitations:

1. As the internal key token definition is implementation-dependent, it should be assumed that an operational key token created on one cryptographic subsystem implementation will not work on another implementation. (See the description of internal and operational keys in the section on operational keys.)
2. As the key storage definition is implementation-dependent, it should be assumed that the key storage for one implementation will not work with another implementation unless a conversion utility exists and is executed.

Portability is the ability to move an application program from one system to another without changing the program. However, it should be noted that portability is at the source code level. That is, it should be assumed that it is necessary to recompile the application program in order for it to be able to run on a different cryptographic system or to run on a different cryptographic product on the same system.

Cryptographic API objectives. The objectives in designing the architecture for the Common Cryptographic Architecture Cryptographic API were to provide an intuitive multisystem application programming interface while allowing for future growth.

It was realized early in the design process that the callable services should not be defined at too low a level, as too much of the complexity would then reside with the user. The cryptographic services were designed to attempt to hide as much of the complexity of the underlying system as possible and present an intuitive interface. It was also decided early in the design that it was very desirable to follow the guidelines developed by John Ehrman at the Santa Teresa laboratory for designing Systems Application Architecture™ (SAA™) callable services. Doing so gave a level of confidence that the cryptographic services would be able to be implemented on many different systems. However, this decision required overcoming the challenge of abiding by the SAA callable service guidelines in regard to data types. Apparently, the only data types that are supported in all programming languages are character strings and 32-bit binary integers. Neither of these definitions applies to encrypted data, but further investigation showed that if the data do not need to be manipulated by the caller, an undifferentiated string type (that is, a string that may be composed of arbitrary hexadecimal values) could also be used in a parameter definition. This undifferentiated string data type allowed for the definition of data structures needed to contain information so that the interface could be defined in a more intuitive manner.

Good architectural design must include the possibility for future growth. This requirement may be met by adding new services. However, in many cases, the old service definition may almost meet the new requirement, except that some new parameter option specification requires support. This desirable design trait is addressed by the definition of a rule array in many services. The rule array length and rule array parameters, in effect, support a variable length method of passing information to the service. For any particular level of the software, there will be a defined maximum size of the rule array in any particular service. However, a later level of software may define more parameter options and support a larger rule array size to accommodate increased functionality.

Data structures

Control vector externalization. The control vector is a pivotal concept in the Common Cryptographic Architecture. See the companion papers in this issue[2,3] for more information. The control vector concept is more flexible and more intuitive than the variant concept used in previous cryptographic systems. From a software viewpoint, some important facts about the control vector need to be re-emphasized:

- It allows electronic key distribution without misuse of the key by a normal user.
- It is a compact nonsecret data structure.
- It is conceptually associated with a key.
- It has a dual use: it enforces cryptographic separation, and it specifies key usage.

Operational keys. An operational key is a key that is encrypted under the master key at a particular system and can be used in a service at that system. Operational keys are accessed either directly by value in an internal key token or indirectly by a key label.

An internal key token contains an encrypted cryptographic key and its associated control vector. It is typically used for a key with a short life, as for example, a key that is used for a session and is disposed of when the session is over. It allows implementation support for the capability for authorized users to change the master key while the system is on line. Previously, a master key change required that the cryptographic subsystem be taken off line.

In an internal key token, a field may be set to allow detection of situations where the master key has been changed but the encrypted key in the key token is still encrypted under the old master key. If desired, the system software on detection of such a condition, can re-encipher the encrypted key from encryption under the old master key to encryption under the current master key, and replace the old encrypted value with the new in the key token and continue with the originally requested operation.

A key label indirectly identifies an internal key token stored in key storage. An operational key is a candidate for being kept in key storage if it is a key with a long life (that is, it must survive multiple master key changes), if it is appropriate to enforce system access control to use this key, or if many users need access to this key.

From experience with previous systems, some customers have requested enhancements that al-

Table 1 Common Cryptographic Architecture services

Service Pseudonym	Service Name
Data-operation services	
1. Encode	CSNBECO
2. Decode	CSNBDCO
3. Encipher	CSNBENC
4. Decipher	CSNBDEC
5. Ciphertext Translate	CSNBCTT
6. MDC Generate	CSNBMDG
7. MAC Generate	CSNBMGN
8. MAC Verify	CSNBMVR
Key-management services	
9. Clear Key Import	CSNBCKI
10. DATA Key Export	CSNBDKX
11. Key Export	CSNBKEX
12. Key Generate	CSNBKGN
13. Key Import	CSNBKIM
14. RN Generate	CSNBRNG
15. Secure Key Import	CSNBSKI
PIN-management services	
16. Clear PIN Generate	CSNBPGN
17. Encrypted PIN Translate	CSNBPTR
18. Encrypted PIN Verify	CSNBPVR

low the Cryptographic API services to be passed keys either directly by value or indirectly via a key label. This has been satisfied in the Cryptographic API by the definition of the key identifier parameter found in most of the Cryptographic API services. A key identifier can be either an internal key token or a key label and is 64 bytes long. If the first byte is a character, then the data specified in a key identifier parameter are interpreted by the software as a key label specification. If the first byte is an X'01' (hexadecimel value), then the data specified in a key identifier parameter are interpreted by the software as an internal key token specification.

External keys. An external key is a key encrypted under a key-encrypting key (KEK). The KEK may be either an IMPORTER key or an EXPORTER key. An IMPORTER key is used to import keys onto the system. Importing a key consists of calling the Key Import service which re-encrypts a key in importable form (i.e., from encryption under an IMPORTER key) to operational form (i.e., to encryption under the master key of this system). This makes the key operational on this system. It is possible to create a key in import form directly by using the Key Generate service or the Secure Key Import service. An EXPORTER key is used to export keys to other systems. Exporting a key consists of calling the Key Export service which re-encrypts a key from operational form to exportable form (i.e., to encryption under an EXPORTER key). The encrypted key may then be electronically transported to the receiving system. The EXPORTER key on the sending system is paired with an IMPORTER key on the receiving system as both have the same value when internally decrypted. It is possible to create a key in export form directly by using the Key Generate service.

An external key is kept in an external key token. The external key token is the foundation for Common Cryptographic Architecture external key distribution. Use of the external key token helps ensure interoperability, and is built by the creator of the key. An external key token contains the encrypted key, its associated control vector, and a token validation value.

The token validation value (TVV) is used to verify that an external key token is valid and helps prevent an invalid or overlaid external key token from being accepted by a service. As many values in a key token have no inherent structure (i.e., they appear completely random), it is desirable to add redundancy to the definition of the external key token so that there is a high level of confidence that it has not been corrupted. As some cryptographic operations produce unintelligible ciphertext, it is important that errors not be discovered too late, after some unrecoverable process has occurred.

Cryptographic API services

A reference chart for the Common Cryptographic Architecture Cryptographic API services is shown in Table 1. The *service pseudonym* is the descriptive name for a service, while the *service name* is the formal name for the service and the name by which the service is called from a program. Following is an overview (from the caller's perspective) of the cryptographic functionality each service provides.

Data-operation services. The Cryptographic API provides data-operation services that allow callers to provide data confidentiality and data integrity. The Cryptographic API consists of callable services that provide the cryptographic transformations that allow a system service or customer

application caller of a service to meet security requirements.

Encode and Decode. The Encode and Decode services provide Electronic Code Book (ECB) encryption and decryption of eight bytes of text. The cryptographic key is supplied as an unencrypted eight-byte value, not as an internal key token. As the key is unencrypted, the caller of these services is responsible for ensuring the secrecy of the keys. These services provide compatibility support for previous systems and provide the Data Encryption Algorithm (DEA)[4] encryption and decryption primitives that may be used as subroutines in the design of installation-written special-purpose cryptographic services.

The Encipher and Decipher services provide support for encryption and decryption of sensitive data.

Encipher and Decipher. The Encipher and Decipher services provide support for encryption and decryption of sensitive data. Consider the following scenario: Ann wants to send messages to Bill. However, the messages are transmitted by means that cannot prevent Eve, an adversary, from reading the transmitted messages. The problem is to devise a method whereby the content of messages sent by Ann are kept from being determined by Eve. This is the classic example of the use of cryptography to provide data confidentiality, and the solution is to scramble Ann's intelligible message in such a way that Bill can unscramble it easily but Eve cannot without doing an excessively impractical amount of work.

This data security problem may be solved as follows: Ann and Bill agree to use the Cryptographic API Encipher and Decipher services in one of the five supported methods of encryption and also agree to share a secret value, in particular the value of a DATA key. Ann will Encipher the message and send the encrypted message to Bill, who will then Decipher the encrypted message to produce the original message. Eve may be able to intercept the encrypted message, but without knowing the value of the DATA key cannot determine the original message.

The Cryptographic API supports five different methods of encryption as follows:

1. ANSI X3.106 Cipher Block Chaining (CBC)
2. ANSI X9.23 Octet (Byte) Padding
3. IBM 4700 (Byte) Padding
4. IBM Information Protection System (IPS)
5. IBM Cryptographic Unit Support Program (CUSP)

Five different methods are supported because each method is in use today, each method is different from the others (mainly in its method of short block handling) and each has it own advantages and disadvantages. Note that the IBM 4700 also has a method of encryption which does not pad data and is the same as the ANSI X3.106 Cipher Block Chaining (CBC) method.

ANSI Standard X3.106[5] defines four modes of DEA encryption, and IBM products, in general, support the Cipher Block Chaining (CBC) method. This is a normal method to use but it has one obvious shortcoming, since it is not defined for data with a length that is not a multiple of eight bytes. Various other standards and IBM products have developed different solutions in attempting to handle this concern. ANSI Standard X9.23[6] defines an octet (byte) padding method which always pads the data so that the text length is a multiple of eight bytes. The standard specifies that the pad digits (except for the last digit, which contains a pad byte count) contain varying contents. The IBM 4700 Finance Communication System,[7] besides supporting the CBC method, supports padding the text with a caller-specified pad digit. Like the ANSI X9.23 octet padding method, the 4700 padding method always pads the data and the rightmost pad digit is a count in binary of the number of pad digits (including the count digit).

A disadvantage of using a padding method is that padding produces ciphertext that is longer than the plaintext. If a record needs to be encrypted and the plain text replaced with the encrypted text, it is obviously desirable if the length of the text does not increase. The IBM Cryptographic Unit Support Program (CUSP-3848),[8] which runs

on MVS systems, added extensions to the CBC definition to support a data length of any byte multiple where the resulting encrypted text is the same length as the clear text. The IBM Information Protection System (IPS, also known as the CIPHER command on VM systems)[9] defined an extension to the CBC definition similar to CUSP, in that the ciphertext is the same length as the plaintext. Furthermore, both CUSP and IPS define a concept called record chaining where, in effect, a data set is treated as one long record, i.e., a value is calculated that allows chaining from one record to the next in the data set. However, the definitions of IPS and CUSP record chaining differ.

Guidelines for choosing an encryption method. To reduce the complexity of using these different methods, the Cryptographic API defines the Encipher and Decipher services so that these five options are supported via a caller specification of the process rule parameter. Guidelines for the use of a particular method are as follows:

1. If exchanging encrypted data with a specific implementation, e.g., CUSP or ANSI X9.23, then use that method.
2. The CBC method should be used whenever possible. Use of this method requires that the plaintext length always be a multiple of 8.
3. If the Ciphertext Translate service is to be executed on the encrypted data at an intermediate node, then the caller must ensure that the ciphertext is a multiple of 8. To meet this requirement, using a process rule of CBC, X9.23, or 4700-PAD will prevent the possible inadvertent error of creating ciphertext that is not a multiple of 8 and that cannot be processed by the Ciphertext Translate service.
4. If the ciphertext length must be equal to the plaintext length and the plaintext length may not always be a multiple of 8, then either the IPS or CUSP method should be chosen.
5. If many similar records are being encrypted via repeated calls or if the text will be processed in segments, then the IPS or CUSP record chaining method is suggested.
6. The IPS record chaining method is preferred over the CUSP record chaining method.

The Encipher and Decipher services require a caller to supply an initialization vector. In many cases, an initialization vector of binary zeroes may be used and often is enough to ensure security. However, if the plaintext is highly structured or is repetitious, examination of just the ciphertext may disclose the existence of the structure of the plaintext and such a disclosure may not meet security requirements. If this is the case, the caller should consider using a different nonzero initialization vector for each call, as this will effectively mask any structure or repetition that may exist in the plaintext. The output of the Random Number Generate service is suitable for use as a nonzero initialization vector.

Ciphertext Translate. Another aspect of data security is the possibility that two parties want a third party at an intermediate node to act as a server for them in translating data from encryption under one key to encryption under another. However, Ann and Bill do not want Charles, the third party, to have direct access to the plaintext, i.e., the original message. Such a service is supplied by the Ciphertext Translate service. A trusted system administrator for Charles's system is supplied with a DATAXLAT key by both Ann and Bill and he then manually installs the DATAXLAT keys. Charles can then call the Ciphertext Translate service and the plaintext message does not appear outside the cryptographic subsystem. The intent of using the Ciphertext Translate service is to disallow recovery of the plaintext at the intermediate node.

The Ciphertext Translate service works with systems that were designed before the Common Cryptographic Architecture; therefore when using the Key Generate service to generate a key for use in Ciphertext Translate, a DATAXLAT key is paired with a DATA key. However, note that it is a user at the intermediate node that must do the key generation and not the user at the terminal node. This implies that additional procedural controls may be appropriate when using DATAXLAT keys to ensure the expected security is achieved.

MDC Generate. The MDC Generate service provides support for integrity of data by calculation of a modification detection code (MDC).

Consider the following scenario: Ann creates a large file and wants to distribute the file to Bill. However, Ann and Bill do not share a secret. The problem is how to allow Bill to detect if the file has been altered or replaced in the time between being sent by Ann and being received by Bill. If Ann can inform Bill of 128 bits of data and ensure the integrity of those 128 bits, then the problem may

be solved as follows: Ann and Bill agree to use the MDC Generate service. Ann calculates an MDC for the file and informs Bill (with integrity) of the 128-bit MDC value. Ann then sends the file to Bill. On receiving the file, Bill calculates the MDC for the received file and compares it with Ann's MDC. If the MDC values are equal, the file is accepted as genuine. If not, the file is assumed to be bogus. Note that the output of the MDC Generate process is 128 bits. If an object is less than 128 bits, then the integrity of the object may be maintained directly, rather than through use of an MDC.

An additional use is the ability to detect if the file has been changed since it was originally received, e.g., if altered by a virus. Once an MDC has been established for a file, the MDC Generate service may be run at any later time on the file and the resulting generated MDC compared with the stored MDC to detect deliberate or inadvertent modification.

The MDC calculation[10] provides a publicly-known cryptographic one-way function. That is, the MDC calculation does not rely on any secret information and is easy to compute for specific data, yet it is hard to find data that will result in a given MDC. The data that are to have an MDC calculated may be arbitrarily long, because of support in the MDC Generate service for segmenting the text. The segmenting rule allows callers to break up long text into a first portion, any number of middle portions, and a last portion. In effect, the problem of ensuring integrity of a large file is reduced to the problem of ensuring integrity of a 128-bit value. By providing a data reduction mechanism, the MDC calculation reduces the size of the problem. For example, it is feasible to publish the MDC for a program in a source of public information, e.g., in a certain newspaper for a specific day. The idea is that the source of public information is generally available, cannot be easily spoofed, and any interested party may determine the MDC for the program and verify that the received program results in the same MDC.

Another use of the MDC Generate service is to hash a passphrase down to a value suitable for use as a cryptographic key. A passphrase is conceptually like a password except that it may be 80 characters in length. As the output of the MDC Generate service is 128 bits, the output will need to be truncated when being used for the value of a single length key. When considering such a usage of the MDC Generate service, it is important to remember to use a passphrase with a high variability, i.e., it should be possible to create any of the 2^{56} (over 72 quadrillion) single length keys. This will ensure that an adversary could not reduce the problem of trying to determine the cryptographic key used by trying to guess the passphrase that produced it. For example, a single dictionary word should not be used in the MDC Generate service to calculate a cryptographic key, as an adversary could try every word in a large dictionary and see if any of the values produced by the MDC Generate service is the one used. A large dictionary typically has a total of about 100000 words, which is much less than the number of possible cryptographic keys.

MAC Generate. The MAC Generate service provides support for data integrity via the calculation of a message authentication code (MAC). Conceptually, a MAC is a cryptographic checksum that is based on a shared secret between the message creator and the message recipient. The shared secret in this case is the value of the cryptographic key. The MAC calculation supported is described in the ANSI X9.9 standard,[11] specifically it is the MAC calculation on binary data. The data that are to have a MAC calculated may be arbitrarily long because of support in the service for segmenting the text, exactly as defined in the MDC Generate service.

Consider the following scenario: Two parties decide to send electronic messages between themselves. However, the messages are transmitted by means that cannot prevent an adversary from interjecting messages into the transmission channel. The problem is to devise a method whereby genuine messages sent by the other party are differentiated from bogus messages interjected by an adversary (with a very high probability). This data integrity problem may be solved as follows: The parties agree beforehand to follow the ANSI X9.9 standard and share a secret DEA key that is known only to themselves. Ann creates a message and calculates a MAC for the message and sends the message and the calculated MAC to Bill. Now consider the situation from Bill's viewpoint. For all he knows, the message is either genuine or bogus. To verify that the message is genuine, Bill calculates a MAC on the received message and compares it to the received MAC. If the comparison is equal, then the message is accepted as genuine (i.e., created by Ann who shares the se-

cret key) because only Ann knows the secret that allows for the correct MAC to be calculated.

An additional concern of the data integrity problem is as follows: What if Eve merely resends an old message that already has a correct MAC calculated for it? Such a message is termed a stale message and (depending on the application) it may be important for Bill to detect the replay of a stale message. The solution is for Ann to include time-varying information in the message that allows Bill to detect a replay of a stale message. Examples of time-varying information are a sequence number, a time stamp, or a random number nonce used in a request/response protocol. (A nonce is a technical term for a random quantity used in a security message protocol.) Bill interrogates the time-varying information to ensure that the message is acceptable according to preestablished criteria, such as the sequence numbers must always increase, the time stamp must be a time within a specified window of the current time, etc.

If both data confidentiality and data integrity are required, then both message encryption and message authentication may be done. It is recommended that the MAC be calculated on the plaintext message and that the key used to encrypt the message have a different value from the key used to calculate the MAC on the message.

MAC Verify. Another aspect to data integrity that may be desirable in some situations is the concept of a MACVER key, i.e., a key that may be used in the MAC Verify service but cannot be used in the MAC Generate service. The data that are to have a MAC verified may be arbitrarily long, because of support for segmenting the text by the MAC Verify service, in the same manner as is done in the MAC Generate service.

Consider the following scenario: Suppose the recipient may verify the MAC for a genuine message but the cryptographic work factor to generate a MAC for an arbitrary message is too large to be practical. Given a message with its associated MAC, the intent of this concept is that the only realistic way the MAC could have been calculated is by Ann, the owner of the MAC key and not Bill, the owner of the MACVER key, i.e., Ann cannot at some later time disavow or repudiate the message by claiming it was not created and authenticated by her. Such a service is supplied via the use of a MACVER key. A MACVER key may only be used in the MAC Verify service, not the MAC Generate service.

Key-management services. Doing data-operation services by importing clear key values requires the transporting of the clear key values and their associated key types from the originator to the recipient. When the originator and the recipient reside on different systems, this may be done by physically transporting the information by a courier.

However, there are concerns with such a manual procedure, because it is error prone as it requires human involvement. Good security practice dictates that each originator and recipient have a unique key value so that compromise of the key used in one channel does not compromise another channel. This means that many keys may need to be distributed. Good security practice also dictates that the keys be changed if compromise is suspected and, in any case, changed on a regular basis.

As the number of users of cryptographic services grows, the manual approach becomes infeasible to use with all keys. What is desired is a cryptographic key-distribution method that simplifies electronic distribution of keys, minimizes manual key entry, and maintains the intended usage of keys.

The Common Cryptographic Architecture key-management services allow a caller to support generation, installation, and distribution of cryptographic keys. The distribution of cryptographic keys is standardized through the use of an external key token.

Random Number Generate. The Random Number Generate service provides user access to a random or a strong cryptographically-based pseudorandom number generator. The output of a call to this service is suitable for cryptographic use. That is, a suitable value for the clear key parameter of the Clear Key Import or Secure Key Import service is the output of a Random Number Generate call. Also, when using a nonzero initialization vector in the Encipher service, a suitable value for the initialization vector parameter is the output of a Random Number Generate call. It may also be used as a nonce.

The output of this service has superior properties of randomness when compared with the output of

pseudorandom number generators provided in many programming languages, which typically are based on the method of linear congruences and are not cryptographically based. For these reasons, when a high quality random or pseudorandom number is desired for non-cryptographic reasons, this service may be used.

Clear Key Import and Secure Key Import. The Clear Key Import service provides the ability to create an internal key token for a DATA key for an arbitrary key value. The Secure Key Import service does a similar function except that any supported generic key type may be specified. This latter service is authorized, as it is not intended for the general user. Use of these services transforms the value of the key from an unencrypted form into an operational form where it may be used in a way appropriate for its specified key type. The output of these services is a data structure called the internal key token which contains the encrypted key value and an encoding of the specified key type. The key in an internal key token is an operational key, i.e., it is encrypted under the master key. Note that if the specification of the key type is changed either accidentally or intentionally the correct key value will not be recovered as the value of the encrypted key is cryptographically coupled to the control vector associated with the specified key type. See the companion paper by S. M. Matyas[2] for details on the control vector mechanism. It is recommended that the values of the clear keys to be imported by these services should be generated via a call to the Random Number Generate service or via a call to the MDC Generate service where the input text is of sufficient variability.

Key Export. The Common Cryptographic Architecture Cryptographic API supports electronic key distribution with minimal manual key installation. An initial EXPORTER key-encrypting key is installed on a system by a courier and an initial IMPORTER key-encrypting key is installed on another system. The EXPORTER key and the IMPORTER key have the same value. After the manual installation of these initial key-encrypting keys, all subsequent key distribution may be done electronically. For example, Ann on a system with an EXPORTER key installed as above may execute the Key Export service to perform the cryptographic transformations to convert the information for an operational key in an internal key token to a exportable key in an external key token. The output of the Key Export service is a data structure called the external key token which contains the encrypted key and its associated control vector. The key is encrypted under the key-encrypting key that exists on Ann's sending system as an EXPORTER key and on Bill's receiving system as an IMPORTER key. Note that if the specification of the control vector is changed either accidentally or intentionally the correct key value will not be recovered as the value of the encrypted key is cryptographically coupled to the control vector. The Key Export service also transforms a key label specification (which refers to an operational key in key storage) to an external key token.

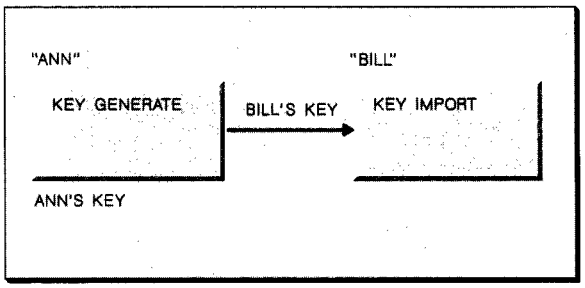
DATA Key Export. The DATA Key Export service does the same transformation as Key Export, but only for a DATA key. This allows an installation to define a higher level of authorization for the Key Export service, that is, the ability to export any key.

Key Import. The external key token can be electronically transmitted to another system that has the corresponding IMPORTER key. An application may execute the Key Import service to perform the cryptographic transformations to convert the information in the external key token (which is considered an importable key) to an operational key in an internal key token. The intended usage of the key (i.e., the key type) is maintained through the control vector mechanism. It cryptographically couples the usage attributes of a key with the key so that the key cannot be recovered and used without specification of the correct control vector (either explicitly in the external key token or implicitly via specification of the key type).

The definition of the external key token supports the strategic key-distribution protocol. The strategic protocol is guaranteed to work on all systems that conform to the Common Cryptographic Architecture Cryptographic API definition. The external key token has two methods of use as follows:

1. Bill calls the Key Import service specifying TOKEN. In this case, the control vector in the external key token is used as is, and any supported control vector will be processed. Bill might do this if he just created the key token and therefore knows the control vector is cor-

Figure 2 Peer-to-peer key-distribution environment



rect or if he wants to process the key token regardless of its contents.
2. Bill calls the Key Import service specifying a generic key type. This process is the same as if specifying TOKEN, except that the Cryptographic API also verifies that the control vector in the external key token is compatible with the generic key type specified.

In the following examples Ann has generated a MACVER key (that is, a key that may only be used with the MAC Verify service) and sent it to Bill.

1. Bill calls Key Import specifying a key type of TOKEN. This specification will import the key regardless of the control vector in the key token.
2. Bill calls Key Import specifying a key type of MACVER. This specification is correct and therefore will allow the import to continue and Bill is assured that he is importing a MACVER key.
3. Bill calls Key Import specifying a key type of PINVER (that is, a key that may be used in the Encrypted PIN Verify service). This call will fail as the Control Vector in the external key token does not agree with the specified expected key type. Bill knows that there has been a failure somewhere and can pursue resolving the problem.

Besides being paired with an EXPORTER key, an IMPORTER key may be used by itself to support file encryption. A DATA key needs to exist in import form. This may be done by using either Secure Key Import or Key Generate. The DATA key in import form (encrypted under a specific IMPORTER key) may be imported via the Key Import service resulting in an operational DATA key. The operational DATA key may then be used to encrypt a file. The operational DATA key is then discarded and the import form of the DATA key is kept with the encrypted file. If the IMPORTER key is kept in key storage, then any master key changes will not invalidate the IMPORTER key, assuming a product-specific key storage conversion utility is run. When the file is needed to be decrypted, then the Key Import service may be called with the appropriate IMPORTER key and import form of the DATA key to produce an operational DATA key which may then be used to decrypt the file.

Key Generate. The Key Generate service supports the generation of a key or pair of keys. If a pair of keys is generated, both of the keys have the same value but may have different key types (for example, MAC and MACVER) and different key forms (that is, operational, importable, or exportable) as allowed by the service. The Key Generate service is the standard method of creating keys in the Common Cryptographic Architecture. Use of the Key Generate service may allow an implementation to restrict usage of the Secure Key Import service to initial installation of EXPORTER and IMPORTER key-encrypting keys. It may also allow an implementation to prohibit usage of the Key Export service or possibly use it just for system backup purposes.

The Key Generate service provides support for a caller to generate a key or a pair of keys in a peer-to-peer key-distribution environment (Figure 2). In a typical application a key is generated that can be used on this system (i.e., one key is either operational or importable), and the same value is used for a key that can be used on another system (the key is exportable). Though the operational and exportable keys have the same value, they will typically have different key types. All output generated keys are encrypted. If a key is encrypted under the master key, then the key is operational on this system and is returned in an internal key token. If a key is encrypted under an IMPORTER key, then the key is returned in an external key token that may be imported to this system (possibly at some later time). If a key is encrypted under an EXPORTER key, then the key is returned in an external key token that may be transmitted to the system with the corresponding IMPORTER key where it may then be imported.

The Key Generate service may also be used by a key-distribution center (KDC) to generate keys to solve what is called the N^2 ("N squared") key-distribution problem. (Refer to Figure 4, later.) This problem results from noticing that the number of key-encrypting keys potentially needed is approximately the square of the number of systems, as any particular system may need to exchange keys with any other system. As the number of systems (or nodes) in a network is typically denoted by the variable N, this problem is known as the N^2 (N squared) key-distribution problem. The number of different keys needed to be generated and installed may therefore be large, even for a small network. The goal is to have the complexity of adding a new system to a network of systems grow proportional to the number of systems in the network and not proportional to the square of the number of systems.

This problem may be solved using the Common Cryptographic Architecture Cryptographic API through use of a designated key-distribution center (KDC). The key-distribution center has an EXPORTER key installed for each other system in the network, and each other system has the appropriate IMPORTER key installed accordingly. A typical KDC application is where one generated key is exportable to one system and the other generated key is exportable to another system. Notice that the key is not used on the generating system. When one system wants to establish a key with another system, a request is sent to the KDC which acts as a server to generate the two external key tokens needed. The external key tokens are then electronically transmitted to their respective systems where they can be imported.

Peer-to-peer key distribution. A typical peer-to-peer key-distribution scenario which illustrates the use of the Key Generate service is illustrated in Figure 2. A typical process flow follows:

1. Ann calls Key Generate with a mode of OPEX (that is, the first generated key is operational on this system and the second key is exportable), key type1 of MAC (that is, a key that can be used in the MAC Generate service) and key type2 of MACVER (that is, a key that can be used in the MAC Verify service but not the MAC Generate service), and also specifies via key label the appropriate EXPORTER key associated with Bill's node.
2. Ann keeps the generated operational MAC key token.
3. Ann sends the generated exportable MACVER key token to Bill.
4. Bill is expecting to receive a MACVER key from Ann, so on receipt of the external key token Bill calls Key Import specifying the received key token, a key type of MACVER and the IMPORTER key associated with Ann's node. This produces an operational MACVER key token for Bill.
5. Ann calls the MAC Generate service specifying the MAC key token to generate a message authentication code (MAC) for a specific message, appends the MAC to the message and sends this combination to Bill.
6. Bill receives the message and its MAC and calls the MAC Verify service specifying the operational MACVER key token to verify the MAC for the message. Assuming the MAC verifies, Bill is assured that the message really did come from Ann.
7. Eve cannot create a MAC for a bogus message because she cannot determine the value of the MAC key as it is encrypted under the master key of the system where Ann resides and cannot determine the value of the MACVER key used by Bill as it is encrypted under the secret key-encrypting key shared between the systems that Ann and Bill reside on.

For an example of the technical implementation details needed to support a peer-to-peer key-distribution environment with control vectors, see the companion paper by S. M. Matyas, A. V. Le, and D. G. Abraham.[3]

Key-distribution center. Use of a peer-to-peer key-distribution protocol is appropriate for small networks or where a few nodes in a large network are able to act as a small network. However, attempting to solve the general key-distribution problem in a large network may result in a different solution, due to scaling factors of the necessary number of manually installed key-encrypting keys. For example, for the arbitrary network depicted in Figure 3, a key-distribution center is one method of attempting to solve the scaling problem. (See Figure 4.)

Use of the Key Generate service in a key-distribution center environment is also possible. In this environment, a third system generates the keys in external form for distribution to both requesting

Figure 3 Generalized computer network

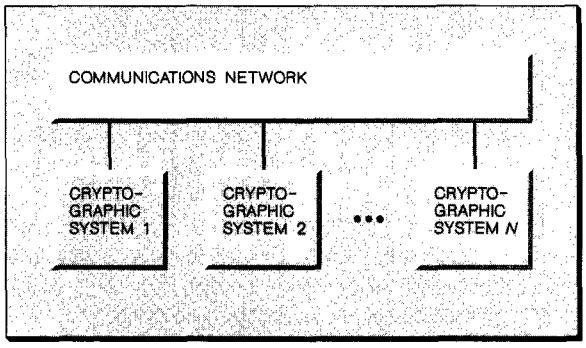
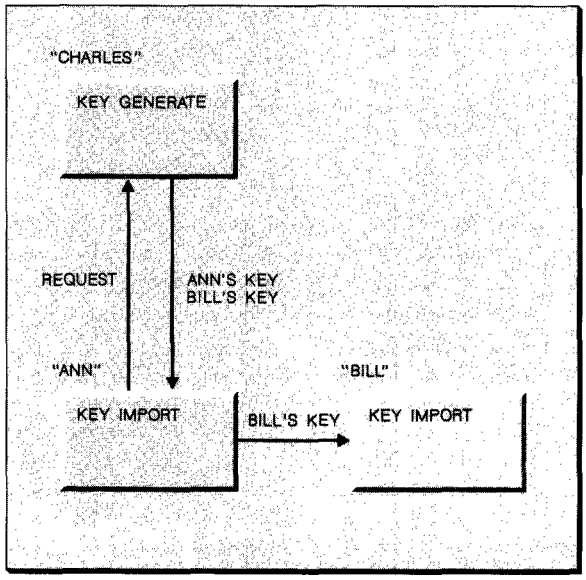


Figure 4 Key-distribution center environment



parties, that is, both generated keys are in exportable form. Using this method has an advantage in reducing the number of key-encrypting keys needed in the entire network, as each system only needs to establish a key exchange channel with the key-distribution center, rather than with all other systems.

For an example of the technical implementation details needed to support a key-distribution center environment with control vectors, see the companion paper by S. M. Matyas, A. V. Le, and D. G. Abraham.[3]

PIN-management services. A personal identification number (PIN) is used as an authentication mechanism to prove the identity of an individual. Conceptually, a PIN is similar to a password. Today, most PIN processing is done in connection with an automated teller machine (ATM) and authorizes personal financial transactions. A customer of a financial institution inserts the ATM card, then enters a PIN to provide the authorization. Information from the magnetic strip on the ATM card and the supplied trial PIN is then transmitted to a site that is authorized to verify the trial PIN.

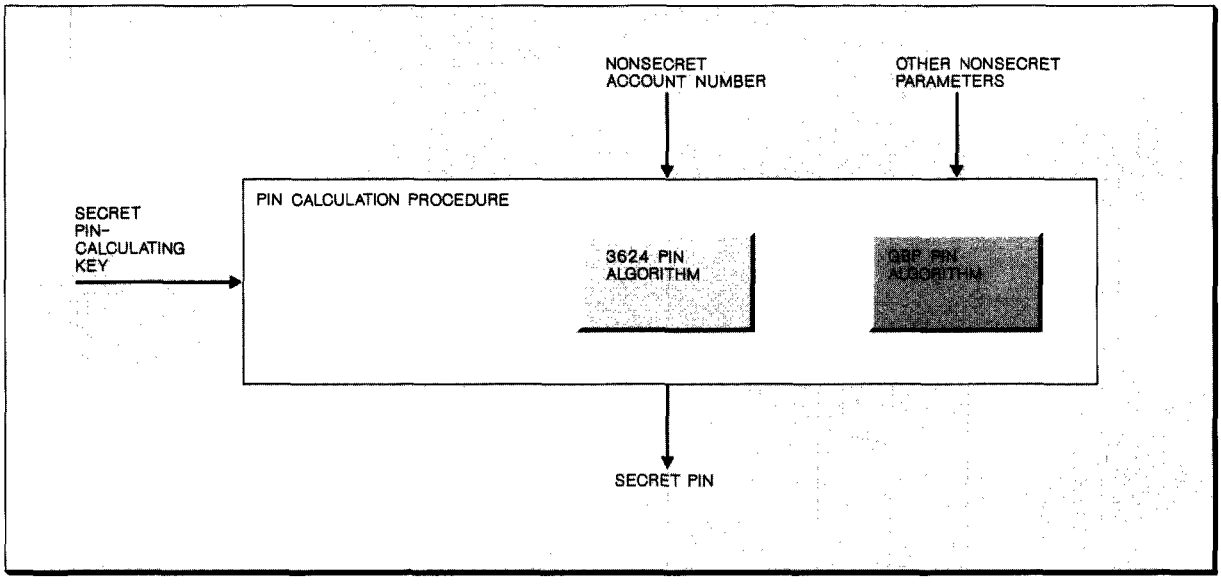
There are two basic methods used to verify a PIN: the PIN database method and the PIN calculation method.

The PIN database method of PIN verification is to keep the value of the PIN in a database in a specific encrypted PIN block format. Note that some PIN block formats have varying contents and so may not be practical for use in a PIN database. When a request for customer authorization arrives, the encrypted PIN block containing the trial PIN is compared for equality with the encrypted PIN block containing the correct PIN. Note that the PIN block containing the trial PIN may be encrypted under a different key than is used to encrypt the PIN database. In this case, the PIN block needs to be translated from encryption under one key to encryption under another key. Of course, the PIN block formats and the contents of the PIN blocks must be identical for the comparison to succeed. If this is not the case, then the PIN block needs to be reformatted to the PIN block format used in the PIN database.

The PIN calculation method of PIN verification is to extract the trial PIN from the encrypted PIN block, recalculate the (correct) PIN using customer account information and the secret PIN verification key, and compare the trial PIN with the calculated PIN.

Notice that both PIN verification methods do not allow a clear PIN value to appear in the clear outside the cryptographic subsystem. This is important for good PIN security and is a reason that a DATA key is not used to encrypt PIN blocks, as a DATA key may be used to decrypt encrypted data, i.e., its corresponding plaintext may appear in the clear outside the cryptographic subsystem.

Figure 5 Clear PIN calculation



The value of the PIN is determined by the method of PIN calculation. Five different methods of PIN calculation are supported.

When a PIN is transmitted between systems, it is contained in a 64-bit encrypted PIN block. Ten different PIN block formats are supported and an appropriate method of extracting the PIN from each PIN block format is provided.

Clear PIN Generate. The Common Cryptographic Architecture Clear PIN Generate service supports generation of the following five outputs:

1. Clear IBM 3624 PIN (output is an institution-assigned PIN)
2. Clear IBM 3624 PIN offset (input is a customer-selected PIN, output is the PIN offset)
3. Clear IBM German bank pool (GBP) PIN (output is an institution PIN)
4. Clear IBM German bank pool PIN offset (input is an institution PIN, output is the PIN offset)
5. Clear VISA™ PIN validation value (PVV), (input is a customer PIN)

The Clear PIN Generate service is an authorized service and is not intended for use by a normal (unauthorized) user. It may be used as part of the process of creating an installation PIN database or creating PIN mailers that are sent to the financial institution's customers that want to use an ATM. See Figures 5 and 6.

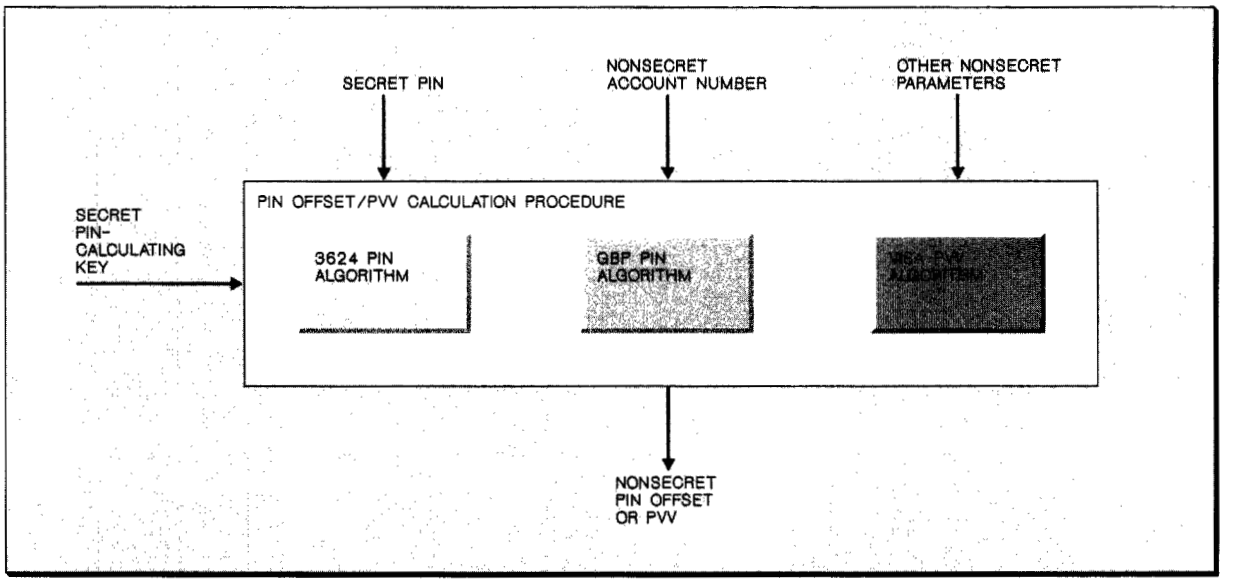
Encrypted PIN Translate. The Encrypted PIN Translate service allows an intermediate system to change the value of the key the PIN block is encrypted under, change the PIN block format, or change some of the non-PIN contents of the PIN block.

If just the value of the key is being changed, this is termed a key-translate invocation with a process rule of TRANSLAT. In any other case, the process is termed a key-translate invocation with a process rule of REFORMAT. REFORMAT functionality is a superset of TRANSLAT functionality, except that sometimes a REFORMAT may fail in successfully doing a PIN extraction or PIN formatting while a TRANSLAT process usually cannot fail if the parameters are specified correctly.

The Common Cryptographic Architecture Encrypted PIN Translate service supports the following ten PIN block formats for both the inbound PIN block and the outbound PIN block:

1. IBM 3624
2. IBM 3621 (same as IBM 5906)
3. IBM 4704 encrypting PIN pad
4. ISO 0 (same as ANSI X9.8,[12] VISA 1, and ECI 1)

Figure 6 Nonsecret PIN offset or PVV calculation



5. ISO 1 (same as ECI 4)
6. VISA 2
7. VISA 3
8. VISA 4
9. ECI 2
10. ECI 3

In the above list, ISO is an acronym for the International Organization for Standardization, ANSI is an acronym for the American National Standards Institute, and ECI is an acronym for eurocheque International S.C.

The Encrypted PIN Translate service supports the PIN database method of PIN verification by allowing a caller to convert a formatted PIN block to encryption under a specific key (presumably the key the PIN database is encrypted under) or to convert a PIN in one PIN block format to another PIN block format (presumably the PIN block format the PIN database is in). The Encrypted PIN Translate service also allows an application program to meet required PIN block formats for use in interchange with other systems, when the PIN originates in some other PIN block format.

Encrypted PIN Verify. The Encrypted PIN Verify service provides support for the PIN calculation method of PIN verification. The PIN corresponding to the supplied account information is generated inside the cryptographic subsystem and compared for equality with the PIN in the supplied encrypted PIN block. See Figure 7.

The Encrypted PIN Verify service supports verification of the following five inputs:

1. IBM 3624 institution-assigned PIN
2. IBM 3624 customer-selected PIN (via a PIN offset)
3. IBM German bank pool PIN (verify via an institution key)
4. IBM German bank pool PIN (verify via a pool key and a PIN offset)
5. VISA PIN (via a VISA PIN validation value [VISA PVV])

The Common Cryptographic Architecture Encrypted PIN Verify service supports the same ten inbound PIN block formats that are supported by the Encrypted PIN Translate service.

Cryptographic key separation. An important concept used in the Common Cryptographic Architecture Cryptographic API is cryptographic key separation. This concept provides for the creator of a cryptographic key (whether via the Key Generate service or the Secure Key Import service) to declare the intended usage of the key via a key type specification. The cryptographic subsystem

then enforces this specification by denying requested services that are inappropriate for the declared key type.

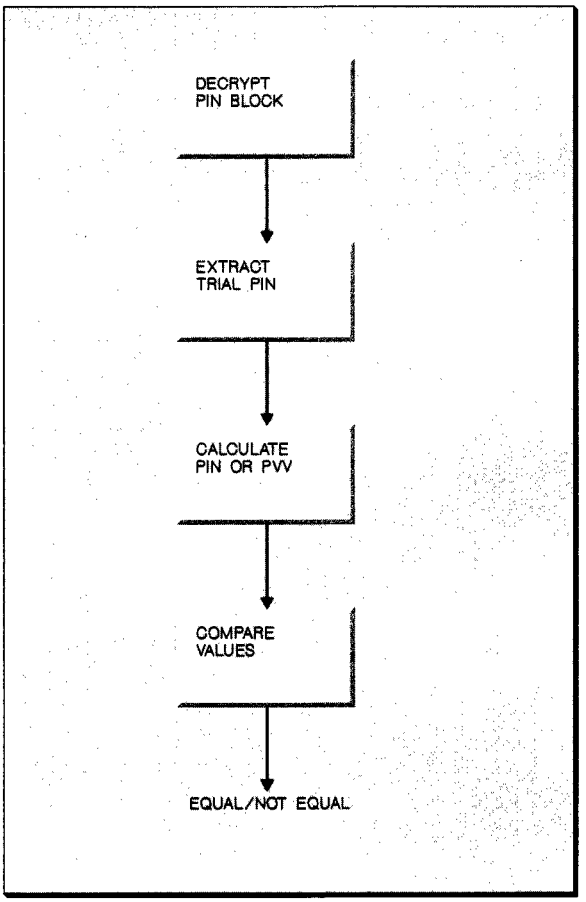
The mechanism for enforcing key separation for keys in internal key tokens or external key tokens is via the control vector mechanism. Each key K is encrypted in such a way that the value of the key-encrypting key KK (whether a master key, an EXPORTER key, or an IMPORTER key) and the control vector C (associated with K) must be specified to recover the key. The possible values of the control vectors are defined by the Common Cryptographic Architecture. See the companion paper by S. M. Matyas[2] for details on the cryptographic transformations. Besides using the control vector C to recover the value of key K, C is also examined to see if it has attributes that qualify it to be used in the called service in the requested way. If it does not, the service invocation fails. If C is valid, the requested service execution proceeds. If a caller alters the value of C to try to change the attributes of a key, the correct value of K is not recovered by the key decryption process and any resulting output of the service is invalid, that is, any output is equivalent to that resulting from using a random unknown key value in the service.

Generic key types. The method with which the Common Cryptographic Architecture Cryptographic API externalizes the power (and the complexity) of the control vector concept is by defining generic key types.

A customer's desire or need for sophistication will typically grow over time. Initially, a simple running system may be what is desired. As experience is gained and knowledge increases, a security administrator may want to increase the level of control over usage of cryptographic keys by the end users. For example, initially Ann may decide to distribute a key to Bill that can be used to generate a message authentication code (MAC) that is used to authenticate the text of an electronic message. This scenario likely entails electronic distribution of a MAC key. Later, Ann may decide that Bill does not really need to be able to generate a MAC, rather he only needs to verify a MAC. This suggests possible generation and distribution of a MACVER key.

Each generic key type is defined according to the service in which it can be used as input. This

Figure 7 Steps in algorithmic PIN verification



exposes the control vector to a user in a top-down manner, rather than in a bottom-up manner, which is the way the control vector was designed. This frees the user from needing to know all the details of the control vector definition.

For example, instead of defining a generic MAC key (only) and requiring the user to specify an additional option stating that this key is only usable in a MAC Verify service (as is done in the control vector field definition) it seemed preferable to define a generic MAC key as well as a generic MACVER key. A MAC key is used in the Common Cryptographic Architecture MAC Generate service. A MACVER key can only be used in the Common Cryptographic Architecture MAC Verify service.

Common Cryptographic Architecture key types. There are ten generic key types supported by the

Table 2 Common Cryptographic Architecture generic key types

Key Type	Attributes
DATA	A single length key that may be used in the Encipher and Decipher services. It may be used to encrypt and decrypt data.
DATAXLAT	A single length key that may be used as either the inbound or outbound key in the Ciphertext Translate service.
MAC	A single length key that may be used in the MAC Generate service. It may be used to calculate a message authentication code on data.
MACVER	A single length key that may be used in the MAC Verify service. It may be used to verify a message authentication code on data.
IMPORTER	A double length unidirectional inbound key-encrypting key. It may be used as the key-encrypting key in the Key Import, Key Generate (as appropriate), and Secure Key Import services.
EXPORTER	A double length unidirectional outbound key-encrypting key. It may be used as the key-encrypting key in the Key Export, Key Generate (as appropriate), and DATA Key Export services.
PINGEN	A double length key that may be used as the PIN generating key in Clear PIN Generate. This key is used internally in the PIN calculation.
PINVER	A double length key that may be used as the PIN verifying key in Encrypted PIN Verify. This key is used internally in the PIN calculation.
IPINENC	A double length unidirectional inbound PIN-encrypting key. It may be used as an inbound PIN-encrypting key in Encrypted PIN Translate and Encrypted PIN Verify. This key is used internally to decrypt an encrypted PIN block.
OPINENC	A double length unidirectional outbound PIN-encrypting key. It may be used as an outbound PIN-encrypting key in Encrypted PIN Translate.

Note 1: Any supported key type may be the source key of the Key Import or the Key Export services.
Note 2: Any supported key type may be specified for a key in the Secure Key Import service.
Note 3: Any key type may be generated via Key Generate, subject to the restrictions of the Key Generate service.

Common Cryptographic Architecture. The attributes for each key type are shown in Table 2.

The value of an EXPORTER key on one system is typically the value of an IMPORTER key on a second system. This allows the first system to send encrypted keys in external key tokens to the second system. If both transmission and reception of encrypted keys is desired, then both an IMPORTER and an EXPORTER key should be installed on both systems.

The OPINENC (Outbound PIN Encrypting) key is used internally to encrypt an unencrypted PIN block. The value of an OPINENC key on one system is typically the value of an IPINENC (Inbound PIN Encrypting) key on another system. This allows an encrypted PIN block to be transmitted from the first system to the second and translated.

The Common Cryptographic Architecture Cryptographic API supports a subset of the control vectors as described in the companion paper by S. M. Matyas.[2] This subset is as follows:

- Parity—As defined
- Cryptographic facility access program (CFAP) control—Must be defaulted to zero (no CFAP enforced control)
- Antivariant—As defined
- Extension—Must be defaulted to zero (64-bit control vector)
- Key part—Must be defaulted to zero (the control vector is associated with a key, not a key part)
- Export control—Must be defaulted to zero (export is allowed)
- Form—Data-operation keys must be defaulted to class 1 (a 64-bit key) and key-management and PIN management keys must be defaulted to class 3 (a 128-bit key where the left and right halves may or may not be equal).
- Control vector type/subtype—Each Common

Cryptographic Architecture generic key type has a corresponding control vector key type/subtype as follows.

DATA—Data Compatibility
DATAXLAT—Data Compatibility-Translate
MAC—Data MAC
MACVER—Data MAC
PINGEN—PIN Generating
PINVER—PIN Generating
IPINENC—PIN Encrypting-In
OPINENC—PIN Encrypting-Out
EXPORTER—Key-Encrypting Sender
IMPORTER—Key-Encrypting Receiver

In addition, the cryptovariable intermediate control vector is used internally by some implementations.

- Usage control—For each supported generic key type the usage control field is a specific value. As the supported key types are generic, each value is determined to allow the most functionality for a given control vector key type/subtype, except that the two verification key types (MACVER and PINVER) have only the appropriate verification functionality and the two key-encrypting key types (EXPORTER and IMPORTER) do not have key translation capability.
- Log—Must be defaulted to zero (no logging)
- Reserved—As defined

Summary

In summary, the design of the architecture of the Common Cryptographic Architecture Cryptographic API conforms to SAA callable service guidelines. It adheres to the goals of interoperability and program portability, supports more functions, and is more granular than previous cryptographic systems, yet is arguably more user-friendly than previous interfaces. Key-management services are provided which exploit the control vector to provide remote key-usage control. Perhaps most important, the Cryptographic API was designed with the expectation that additional functionality requirements will evolve, yet no one today is sure what those new directions will entail.

Acknowledgments

The author would like to thank the following IBM people: Dennis ("Abe") Abraham of Charlotte for the comprehensive cryptographic function set and his continuing input, Gina Bourbeau and Lucina Green of Kingston for providing models of callable services that follow SAA guidelines, Phil C. Yeh and Ronald Smith of Poughkeepsie for their help in refining the architecture definition, Bob Elander of Kingston for his help in defining the Common Cryptographic Architecture Cryptographic API, and Russ Prymak and John Wilkins of Manassas for their constructive comments and help.

Cited references

1. *Common Cryptographic Architecture Cryptographic Application Programming Interface*, SC40-1675, IBM Corporation (1990); available through IBM branch offices.
2. S. M. Matyas, "Key Handling with Control Vectors," *IBM Systems Journal* **30**, No. 2, 151–174 (1991, this issue).
3. S. M. Matyas, A. V. Le, and D. G. Abraham, "A Key-Management Scheme Based on Control Vectors," *IBM Systems Journal* **30**, No. 2, 175–191 (1991, this issue).
4. American National Standard X3.92-1981, *Data Encryption Algorithm*, American National Standards Institute, New York (1981).
5. American National Standard X3.106-1983, *Modes of Encryption of the Data Encryption Algorithm*, American National Standards Institute, New York (1983).
6. American National Standard X9.23-1988, *American National Standard for Financial Institution Encryption of Wholesale Financial Messages*, American Bankers Association, Washington (1988).
7. *4700 Finance Communication System Controller Programming Library Volume 5, Cryptographic Programming*, SH20-2621, IBM Corporation (1983); available through IBM branch offices.
8. *OS/VS1 and OS/VS2 MVS Cryptographic Unit Support: Installation Reference Manual*, SC28-1016, IBM Corporation (1980); available through IBM branch offices.
9. R. K. McNeill, *Information Protection System Cryptographic Programs for VM/CMS Users Guide*, SH20-2621, IBM Corporation (1982); available through IBM branch offices.
10. D. Coppersmith, S. Pilpel, C. H. Meyer, S. M. Matyas, M. M. Hyden, J. Oseas, B. Brachtl, and M. Schilling, *Data Authentication Using Modification Detection Codes Based on a Public One Way Encryption Function*, U.S. Patent No. 4,908,861 (March 13, 1990).
11. American National Standard X9.9-1986, *American National Standard for Financial Institution Message Authentication (Wholesale)*, American Bankers Association, Washington (1986).
12. American National Standard X9.8-1982, *American National Standard for Personal Identification Number (PIN) Management and Security*, American Bankers Association, Washington (1982).

Don B. Johnson *IBM Federal Sector Division, 9500 Godwin Drive, Manassas, Virginia 22110.* In 1974, Mr. Johnson received a B.A in mathematics from Oakland University, Rochester, Michigan. He subsequently joined the IBM Field Engineering Division where he worked as a program support representative on MVS systems, mainly at the General Motors Technical Center, Warren, Michigan. In 1978, he joined the 8100/DPPX Change Team in Kingston, New York. In 1982, he worked on DPPX/APL development in Lidingoe, Sweden. Since 1987 he has worked in the Cryptography Center of Competence in Manassas, Virginia. He holds four patents because of his contributions to the Common Cryptographic Architecture and the Transaction Security System product architecture. He is currently an advisory programmer and will soon complete the requirements for a master's degree in computer science from Union College, Schenectady, New York.

George M. Dolan *IBM Services Sector Division, 1001 W. T. Harris Boulevard, Charlotte, North Carolina 28257.* Mr. Dolan graduated from Lehigh University with a B.S. in electrical engineering. Since joining IBM at Endicott, New York, in 1961, he has had design responsibilities for various communications hardware and software products, which in recent years have been principally for the worldwide finance industry. Mr. Dolan is a senior engineer in the IBM Secure Workstation Development department. He has worked on the Transaction Security System, integrating cryptographic processors into IBM PS/2 and MVS systems and integrating the result into customer applications for the protection of data and user identification. His responsibilities include specifying the user programming interface and software structure in support of the Transaction Security System.

Michael J. Kelly *IBM Kingston, Neighborhood Road, Kingston, New York 12401.* Mr. Kelly is an advisory programmer working on the development of encryption products in the IBM Kingston laboratory. He received a master's degree in mathematics from Syracuse University. Prior to joining IBM in 1980, Mr. Kelly worked as a mathematician in the field of cryptology for the United States Department of Defense.

An V. Le *IBM Federal Sector Division, 9500 Godwin Drive, Manassas, Virginia 22110.* Mr. Le is a staff engineer in the Cryptography Center of Competence in the IBM Manassas laboratory. He received a master's degree in electrical engineering from the University of Utah, Salt Lake City, Utah, in 1982. He joined IBM in 1983 at Boca Raton, Florida, where he worked as a computer designer in a reduced instruction set computer project for several years. In 1986, he joined the Cryptography Center of Competence in Manassas, and has since been working in the area of cryptographic algorithms and architectures. Mr. Le holds four issued patents, four patent applications on file, and has published several technical disclosures in the area of computer design and cryptography. He has received two IBM Invention Achievement Awards.

Stephen M. Matyas *IBM Federal Sector Division, 9500 Godwin Drive, Manassas, Virginia 22110.* Formerly a member of the Cryptography Center of Competence at the IBM Kingston Development Laboratory, Dr. Matyas is currently a member of the Secure Products and Systems department at Manassas, Virginia. He has participated in the design and development of all major IBM cryptographic products, including the IBM Cryptographic Subsystem, and recently he has had the lead role in the design of the cryptographic architecture for IBM's recently announced Transaction Security System. Dr. Matyas holds 26 patents and has published numerous technical articles on all aspects of cryptographic system design. He is the coauthor of an award-winning book entitled *Cryptography—A New Dimension in Computer Data Security,* published by John Wiley & Sons, Inc. He is a contributing author to the *Encyclopedia of Science and Technology*, and *Telecommunications in the U.S.—Trends and Policies*. Dr. Matyas received a B.S. in mathematics from Western Michigan University and a Ph.D. in computer science from the University of Iowa. He is the recipient of an Outstanding Innovation Award for his part in the development of the Common Cryptographic Architecture. He is presently an IBM Senior Technical Staff Member.

Reprint Order No. G321-5427.