Transaction Security System extensions to the Common Cryptographic Architecture

by D. B. Johnson
G. M. Dolan

A well-designed application program interface for a line of cryptographic products 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 disparate systems. The Common Cryptographic Architecture is IBM's strategic cryptographic architecture. The Transaction Security System implements the Common Cryptographic Architecture in full. Furthermore, the Transaction Security System has implemented extensions to the architecture to address additional customer requirements. This paper gives the design rationale for some of the additional cryptographic functionality in the Transaction Security System beyond that mandated by the Common Cryptographic Architecture.

The companion paper on the Common Cryptographic Architecture[1] in this issue mentions the many applications today for commercial cryptography. Existing applications include file confidentiality, communications confidentiality, communications integrity, file integrity, and financial transaction authorization via a personal identification number (PIN).

Cryptographic services are implemented in various products, tailored for the environment where they operate. However, they should perform the same operation, with the same results, regardless of the product or the environment. If a customer uses a PC-based product for end-user cryptographic services, it should be compatible with a host-based product that provides the same services.

Cryptographic API model. A model for the Transaction Security System Cryptographic Application Programming Interface (Cryptographic API) is shown in Figure 1. The customer (or system) application (APPL) calls API services to provide the cryptographic transformations. The application typically transmits the cryptographically processed information to an application on another system, which in turn calls appropriate API services to achieve the desired security objective. The cryptographic subsystem consists of all cryptographic functions below the Cryptographic API. In the Transaction Security System, the cryptographic subsystem is composed of system software and a hardware cryptographic facility. The cryptographic facility contains the core of all cryptographic operations and provides zeroiza-

tion of the master key on tamper detection in order to thwart physical threats to the system. See the companion paper by D. G. Abraham, G. M. Dolan, G. P. Double, and J. V. Stevens[2] for information on the design rationale of the Transaction Security System implementation. The *Transaction Security System Programming Guide and Reference*[3] gives details on the callable services provided.

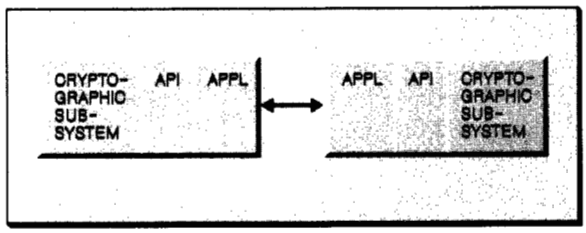
Terminology used in examples. In all examples in this paper the following names are used. The name *Ann* is used for the creator or sender of a key or other output of a Cryptographic API service. The name *Bill* is used for the intended recipient of the key or other output. A user at an intermediate node is *Charles*. The name used for an arbitrary adversary trying to penetrate the security mechanisms is *Eve*.

Objectives

It was realized early in the design process that macro level services that just externalize the cryptographic facility instruction set were not appropriate, as too much of the complexity would reside with the user. An early paradigm that has held up well over time is that the cryptographic facility enforces security while the software provides an intuitive interface and provides security enhancements inappropriate for the cryptographic facility. To enhance the intuitiveness of the design, one of the major goals was to be consistent, by which is meant that externalization of a certain functionality, as for example, a method of encryption or a PIN block format, is done in the same way in all relevant services. The goal of having an orthogonal design meant that conceptually separate functionality was implemented via separate services.

A major goal was to support the Common Cryptographic Architecture Cryptographic API definition.

Figure 1 Cryptographic subsystem model



Also, the services needed to be designed with the idea that they could be implemented on different products and be usable with many programming languages. For these reasons, the general Systems Application Architecture™ (SAA™) guidelines on designing callable services that were identified by John Ehrman at the Santa Teresa laboratory were attempted to be followed as this would give a level of assurance that these goals of generality and usability would be met.

A major goal was to support the Common Cryptographic Architecture Cryptographic API definition. See the companion paper in Reference 1 and the Common Cryptographic Architecture publication in Reference 4 for more information. The support is done in such a way that product specific extensions are often externalized as additional options for parameters in Common Cryptographic Architecture services.

The breadth of functionality of the Transaction Security System may be daunting for a first-time user. For many IBM customers, it is likely that there are few employees knowledgeable about cryptography. It may often be the case that a programmer is issued an "edict" to incorporate security into a program and has neither the time nor the inclination to delve into cryptography but only wants to learn enough to get the job done. This consideration led to the realization that a design which is intuitive and allows a layered explanation (for example, of the granularity of key usage) was a necessity.

In the Transaction Security System architecture, an encrypted cryptographic key is conceptually associated with its control vector, as the value of the encrypted key cannot be recovered to be used in a service without specifying its associated con-

trol vector. In addition, the control vector defines the allowed usage for a key. In the software, the association is made explicit via the key token concept. Most services require keys to be passed in a data structure called a key token. To be used in these services, the key token must contain at least the encrypted key and its associated control vector.

There are certain key-management controls contained in the control vector that are not enforced by the cryptographic facility instructions for reasons either of simplicity of instruction design or the fact that such controls are just not enforceable at the cryptographic facility level. As an example of the former case, the control vector contains some fields that are now software-enforced but could be migrated to cryptographic facility enforcement later, such as support for limiting certain parameter specifications by prohibition of specification of certain options (for example, those that may not be needed or those that might be considered insecure by a customer). An example of a control vector field that cannot be enforced by the cryptographic facility is the KEY-STOR field which, when set, requires a caller to access the cryptographic key indirectly via a label in key storage and prohibits direct reference, that is, via an internal key token. Ann, the creator of the key, may set this field when it is desirable to add a further level of assurance that Bill, the intended user of the key, is required to go through the cryptographic subsystem to resolve the value of the key, as this allows key storage access control mechanisms to be invoked.

Of course, good architectural design must include the possibility for future growth. Obviously, this requirement may be met by adding new services, as specified by the architecture.[1] However, in many cases, the old service definition may almost meet the new requirement, except that some new option specification needs to be supported. This ability is addressed by the definition of a rule array in many services. The rule array length and rule array parameter, 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/or support a larger rule array size to accommodate increased functionality.

Data structures

Cryptographic key separation. An important concept used in both the Common Cryptographic Architecture Cryptographic API and the Transaction Security System Cryptographic API is cryptographic key separation. This concept allows the creator of a cryptographic key to declare the intended usage of the key via the specification of an associated control vector. The cryptographic subsystem then enforces this specification by denying requested services that are inappropriate for the declared control vector.

The mechanism for enforcing key separation for keys is via the control vector mechanism. See the paper by S. M. Matyas on control vectors[5] for details on the cryptographic transformations.

Generic key types. The method by which the software externalizes the functionality (which also may be seen as the complexity) of the control vector is a two-stage process. The first stage is for the user to learn about the generic key types, and the second stage is to utilize the power (and complexity) of full control vectors via use of the Control Vector Generate service.

A customer's desire or need for sophistication may typically grow over time. Initially, a simple operational system may be required. As experience is gained and knowledge increases, a customer may desire to increase the granularity of control of cryptographic key usage by the end user. 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 MAC Verify (MACVER) key. Even later, Ann may decide that Bill should be required to access the key via a key label and that it is invalid for Bill (or anyone else) to pass the key in an internal key token. This entails specifying the KEY-STOR option in the control vector for the MACVER key. With some coordination at Bill's node, access of the key record can be controlled so that only Bill is able to use the key.

Table 1 Generic key extensions

Key Type	Extended Use
DATA	May also be used in MAC Generate and MAC Verify and can be used in Ciphertext Translate if both the inbound and outbound keys are DATA keys.
MAC	May also be used in MAC Verify.
PINGEN	May be used in any service where a PIN calculation is done, including Encrypted PIN Verify.
IPINENC	May be used as the inbound PIN encrypting key to any PIN-management extension service that uses one.
OPINENC	May be used as the outbound PIN encrypting key to any PIN-management extension service that uses one.

Each generic key type is defined according to the service in which it can be used as input. Further specification of the control vector in the Control Vector Generate service results in restrictions on parameter specification possibilities for a function. This exposes the control vector to a user in a top-down method, rather than in a bottom-up method, keeping in spirit with the design. This frees the user from having to know all the details of the control vector.

The Transaction Security System supports the ten generic key types defined in the Common Cryptographic Architecture in full. The Transaction Security System defines certain additional uses for the generic key types as extensions to the Common Cryptographic Architecture definition.[1,4] These extensions are shown in Table 1.

The Transaction Security System defines additional generic key types as extensions to the Common Cryptographic Architecture definition. These additional generic key types are shown in Table 2.

ENCIPHER and DECIPHER keys may be used to simulate the public and private keys of a public key algorithm, within a network composed of Transaction Security System products and trusted security administrators. Use of the Control Vector Generate service allows a very granular level of key-usage specification.

Control vector externalization. From the companion paper by S. M. Matyas[5] it should be clear that the control vector is a pivotal concept in the architecture of the Transaction Security System. The control vector concept is more flexible and more intuitive than the variant concept used in previous cryptographic systems. A control vector on the Transaction Security System allows a very detailed level of key-usage specification. 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. Operational keys are defined just as in the Common Cryptographic Architecture. In addition, as a product-specific extension, all key types may be kept in key storage. Besides the Common Cryptographic Architecture definition of an internal key token containing an encrypted key and its associated control vector, the Transaction Security System supports optional specification of a crypto-period for the key (that is, the time period a key is valid) and optional parameter specifications such as the processing rule or initialization vector to use with this key. The rationale for the support for parameter specifications in the internal and external key token is as follows: Given the existing situation of many possibilities of cryptographic service parameter specifications, knowing just the (encrypted) key value is not necessarily enough to know how to use the key. For example, when doing decryption, one must also know the initialization vector to be used and the decryption process to be used. In earlier systems this information was assumed to be known by context or possibly there was only one way of doing something (for example, the

Table 2 Additional generic key types

Key Types	Description
CIPHER	May be used only in Encipher and Decipher.
ENCIPHER	May be used in the Encipher service only.
DECIPHER	May be used in the Decipher service only.
CIPHERXL (Ciphertext Translation)	May be used in Ciphertext Translate only and is paired with a CIPHER key in Key Generate.
CIPHERXI (Ciphertext Translation Inbound)	May be used only as the inbound key in Ciphertext Translate and can be paired with an Encipher key in Key Generate.
CIPHERXO (Ciphertext Translation Outbound)	May be used only as the outbound key in Ciphertext Translate and may be paired with a DECIPHER key in Key Generate.
IKEYXLAT (Inbound Key Translation)	May be used as the inbound key-encrypting key in the Key Translate service.
OKEYXLAT (Outbound Key Translation)	May be used as the outbound key-encrypting key in the Key Translate service.
CVARENC (Cryptovariable Encrypting)	May be used in Cryptovariable Encipher, for example, to create a Control Vector Translate table entry.
CVARXCVL (Cryptovariable Translate Control Vector Left)	May be used as the key that encrypts the left half of a Control Vector Translate table entry.
CVARXCVR (Cryptovariable Translate Control Vector Right)	May be used as the key that encrypts the right half of a Control Vector Translate table entry.
CVARPINE (Cryptovariable PIN Encrypting)	May be used in encrypt PINs for remote PIN mailer support.
CVARDEC (Cryptovariable Decryption)	May be used to decrypt PINs encrypted by a CVARPINE key when doing remote PIN mailer support.

only key electronically distributed in some former cryptographic systems was what is now termed a DATA key), so that a constant in an earlier system is now a variable. As the architecture was developed, methods were needed to support the ability to relate the additional information to the key. To restate the premise, if one is unsure how to use the key, knowing just the key value is not enough. The alternative is to know by context the parameters for using the key, but as the system functionality increases, it is desirable to have a method that can be used to allow an unambiguous specification of correct parameters by the key generator, so that the burden on Bill, the intended user of the key, may be reduced.

For example, Ann may know that the DECIPHER key she will send to Bill should use the CBC method of decryption. The reason Ann knows this fact is that she owns and will keep and use the associated ENCIPHER key that is paired with Bill's DECIPHER key and she knows that she plans to use CBC encryption. Also, she might know the value of the initialization vector. Notice that both the method of encryption and decryption and the initialization vector need to be known by Bill to be able to use the DECIPHER key correctly. This means that Ann must, somehow, give Bill this additional information along with the key. Ann may decide it is simpler to bundle all this information into the key token and send the contents

of the key token to Bill, rather than ensure Bill gets this information piecemeal, possibly via disparate channels (for example, via face-to-face meetings, phone calls, electronic messages, etc.). This additional information in the external key token is propagated into the internal key token when the key is imported.

Support for the additional optional parameters and crypto-period will be termed the *enhanced key token* in this paper.

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. Note that, as a product-specific extension, a variation of an IMPORTER key is an IKEYXLAT (Inbound Key Translation) key and a variation of an EXPORTER key is an OKEYXLAT (Outbound Key Translation) key. The EXPORTER key on the sending system is paired with an IMPORTER key on the receiving system. External keys are kept in an external key token which is the foundation for the Transaction Security System external key distribution.

External key token examples. In the Transaction Security System, an arbitrary external key token either contains or does not contain an encrypted key and either has or does not have the key's associated control vector. There are four cases to cover:

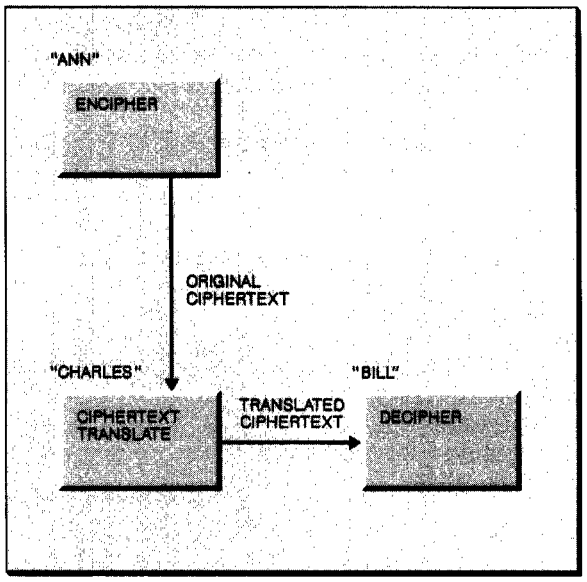
1. No control vector and no key. This is the initial form of a key token before it contains any values. This form is needed where a key token is both an input and an output field, but Bill has no input information to pass. This is supported via use of the null key token, that is, 64 bytes of zeros. Note that the TVV (Token Validation Value) for a key token of all zeros is also zero and that therefore a null key token passes the TVV test. The user is responsible to zero the location where a key token is the output of a Cryptographic API service (for example, Key Generate), as this helps prevent inadvertent overlays of user storage and ensures that no information in user storage is lost.
2. Control vector but no key. This is the form of an external key token that may be used to send a request for the generation of a certain key type in a key-distribution center environment. This is also the form of an external key token that Ann may create to send to Bill on another system so that he may generate a key.
3. Key but no control vector. This is the form of an external key token that supports tactical (that is, customer-designed) key-distribution protocols. For example, this is the form of a key token that may be used by Bill if Ann tells him the generic key type or other control vector information is known via some protocol other than the external key token. This is a method to consider if a customer defines key-distribution protocols and decides that the complete key token does not need to be sent. For example, it is possible to have a protocol where only the encrypted key is transmitted. On receipt of the encrypted key, Bill stores the key in an external key token of this form and specifies the key type in the Key Import service call. Alternately, Bill creates an appropriate key token via use of the Control Vector Generate and Key Token Build services. Note that Bill needs to know what key type (and possibly key attributes) to specify but may know this by context, etc. Clearly, this method is most useful when using generic key types and may not be practical if full control vector support is needed.
4. Key and control vector. This form of the external key token supports the strategic key-distribution protocol. By strategic is meant that it is guaranteed to work on all systems that conform to the Common Cryptographic Architecture Cryptographic API definition. Note that it also supports the Transaction Security System extensions to the Common Cryptographic Architecture and allows for future growth. This form has two methods of use:

    a. Bill calls a 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 correct, or if he wants to process the external key token regardless of its contents.
    b. Bill calls a service specifying a key type. The process is similar to that if TOKEN is specified, but also the the API verifies that the control vector is compatible with the key type specified.

Cryptographic API services

The Transaction Security System supports the 18 callable services defined in the Common Cryptographic Architecture Cryptographic API in full. Transaction Security System extensions to the

Figure 2 Ciphertext translation center



Common Cryptographic Architecture definition are described in this section.

Data operations services. The Transaction Security System supports in full the eight data operation services defined in the Common Cryptographic Architecture.

Encipher and Decipher. The Transaction Security System has the following product-specific extensions to the Encipher and Decipher services: (1) Support for specification of the encryption or decryption key via a key label, (2) segmenting support, and (3) support for CIPHER keys.

The IPS and CUSP encryption methods define a method of record chaining. If such record chaining is desired to be done when using the Common Cryptographic Architecture Cryptographic API, then the output chaining vector in the first eight bytes of the chaining vector parameter needs to be used as the initialization vector on each subsequent call. The Transaction Security System supports record chaining in a direct manner by supporting an additional segmenting rule in the rule array. A specification of INITIAL results in the use of an initialization vector for the encryption or decryption as specified in the initialization vector parameter. Notice that the default segmenting rule is INITIAL, so that if no segmenting rule is specified the results are exactly the same as the Common Cryptographic Architecture definition. However, if CONTINUE is specified, the system will use as an initialization vector the value that was previously stored in the chaining vector parameter from the previous call.

A CIPHER or ENCIPHER key may be used in the Encipher service and a CIPHER or DECIPHER key may be used in the Decipher service. Use of ENCIPHER and DECIPHER keys allows simulation of aspects of public key cryptography. For example, a mailbox application is possible where a user distributes an ENCIPHER key throughout a network and keeps a DECIPHER key private. Any arbitrary user in the network may encrypt a message and send it to the mailbox, but the intent is that only the owner of the DECIPHER key can decrypt the messages in the mailbox. Another example is a broadcast function, where a user keeps an ENCIPHER key private and distributes a DECIPHER key throughout the network. When the user desires to broadcast a message, the user may encrypt it with the ENCIPHER key. The user knows that any arbitrary user in the network can decrypt the message, but the intent is that any arbitrary user knows that only the owner of the ENCIPHER key could have encrypted this message.

Ciphertext Translate. The Transaction Security System has the following product-specific extensions to the Ciphertext Translate service: (1) Support for specification of the ciphertext translation keys via key label and (2) support for CIPHERXL keys.

The explanation of the use of CIPHERXL (Cipher Translate) keys is best given by an example. (See Figure 2.)

Ann and Bill do not share a CIPHER key. However, Charles has agreed to act as a ciphertext translation center for the network. Ann and Bill want Charles, at an intermediate node, to translate the ciphertext of Ann to a form that Bill can use. However, Ann and Bill do not want Charles, the third party, to have direct access to the plaintext, that is, the original message. Such a service is supplied by the Ciphertext Translate service. Charles is supplied with a CIPHERXL key by both Ann and Bill, and Charles can then call the Ciphertext Translate service, but the plaintext message never appears outside the cryptographic

subsystem. Both Ann and Bill created the different CIPHERXL keys to send to Charles via use of the Key Generate service, and they each kept the operational CIPHER key for their own use. The intent of using the Ciphertext Translate service is to disallow recovery of the plaintext at the intermediate node.

If Ann and Bill desire to use unidirectional ciphertext translation keys, then Ann could generate an ENCIPHER/CIPHERXI key pair in OPEX mode and send the CIPHERXI (Ciphertext Translation Inbound) key to Charles, and Bill could generate a DECIPHER/CIPHERXO key pair in OPEX mode and send the CIPHERXO (Ciphertext Translation Outbound) key to Charles.

As it is realized that this service may be desirable to use with DATA keys that work with systems that were designed before the Transaction Security System, the Ciphertext Translate service also supports DATAXLAT keys that are paired with a DATA key. However, if the keys are generated via the Key Generate service, 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.

MAC Generate. The Transaction Security System has the following product-specific extensions to the MAC Generate service: (1) Support for specification of the MAC generation key via key label and (2) support for use of a DATA key to generate a MAC.

Some systems generate a MAC today using a key that is electronically distributed without a control vector, that is, using what is termed a DATA key in the Common Cryptographic Architecture.

If a user wants to obtain the most limitation on the remote usage of the MAC Generate service, the user should distribute a MAC key, as that key has a more limited use when compared with a DATA key.

MAC Verify. The Transaction Security System has the following product-specific extensions to the MAC Verify service: (1) Support for specification of the MAC verification key via key label, (2) support for use of a MAC key to verify a MAC, and (3) support for use of a DATA key to verify a MAC.

The support for use of a MAC key to verify a MAC is added to improve usability, as the ability to generate a MAC certainly implies the ability to verify a MAC.

Some existing systems verify a MAC using a key that is electronically distributed without a control vector, that is, using what is termed a DATA key in the Common Cryptographic Architecture.

If a user wants to obtain the most limitation on the remote usage of the MAC Verify service, the user should distribute a MACVER key, as that key has the most limited use when compared with a MAC key or a DATA key.

Use of ENCIPHER and DECIPHER keys allows simulation of aspects of public key cryptography.

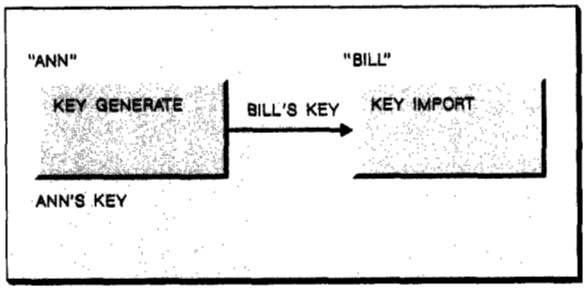
Key-management services. The Transaction Security System supports in full all seven key-management services defined in the Common Cryptographic Architecture.

Secure Key Import. The Transaction Security System, as a product-specific extension to the Secure Key Import service, supports the enhanced version of the key token and also supports the option of specifying TOKEN, which means to get the control vector value for the key from the supplied key token.

Key Export and DATA Key Export. The Transaction Security System supports, as a product-specific extension, the specification of an enhanced external key token to the Key Export and DATA Key Export services.

Prohibit Export. In the Common Cryptographic Architecture, all generic keys are defined to be

Figure 3 Peer to peer environment



exportable, that is, an operational form of the key may be input to the Key Export service. In the Transaction Security System, as a product-specific extension, a key may be generated so that it is not able to be transformed from operational to exportable form via specification of the NO-XPORT option for the key in the Control Vector Generate service. If the key is created in exportable form, then the creator of the key is assured that the key cannot be further exported by a normal user.

Also, in the Transaction Security System, as a product extension, a key that is currently exportable is able to be made nonexportable via use of the Prohibit Export service. This means that a call to the Key Export service with such an operational key will fail and no exportable external key token will be created.

Key Import and DATA Key Import. The Transaction Security System supports, as a product-specific extension, the specification of an enhanced external key token to the Key Import service. The DATA Key Import service is defined, which does the same transformation as Key Import, but only for a DATA key. The existence of the latter service allows an installation to define a higher level of authorization for the ability to import any key type.

Key Generate. The Key Generate service provides support for a caller to generate a key or a pair of keys in a peer-to-peer environment or in a key-distribution center environment. This service supports the full Common Cryptographic Architecture definition. In addition, it supports the enhanced external key token and this support includes the generation of keys as defined by the Control Vector Generate service, which allows a very granular specification of key usage.

The Key Generate service may be used in a peer-to-peer key-distribution environment to generate keys. The key type combinations defined by the Common Cryptographic Architecture are supported as well as additional key type combinations. 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 only for system backup purposes.

A typical peer-to-peer application occurs when a key is generated that can be used on this system (that is, one key is either operational or importable) and the same key value is used to generate a key that can be used on another system (that is, the key is exportable from this system and is importable on another system). All output-generated keys are encrypted. The Key Generate service is the standard method of creating keys in both the Transaction Security System and a system conforming to the Common Cryptographic Architecture.

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 3. Note that this example differs from the example in the companion paper by D. B. Johnson et al.[1] on the Common Cryptographic Architecture. In that example, a MAC/MACVER key pair was established, whereas in this example, an ENCIPHER/DECIPHER key pair is established. A typical process flow follows:

1. Ann calls Key Generate with a mode of OPEX, key type1 of ENCIPHER and key type2 of DECIPHER, and also specifies via key label the appropriate EXPORTER key associated with Bill's node.
2. Ann keeps the generated operational key token ENCIPHER.
3. Ann sends the generated exportable DECIPHER key token to Bill.
4. Bill is expecting to receive a DECIPHER key from Ann, so on receipt of the external key token Bill calls Key Import specifying the received DECIPHER key token, a key type of DECIPHER, and the IMPORTER key associated

with Ann's node. This produces an operational DECIPHER key token for Bill.
5. Ann calls the Encipher service specifying the ENCIPHER key token to encrypt a message and sends the encrypted message to Bill.
6. Bill receives the encrypted message and calls the Decipher service specifying the operational DECIPHER key token to decrypt the message. Bill can then read the message.
7. Eve cannot read the message because she cannot determine the value of the DECIPHER key used as it is encrypted under the key-encrypting key for which she also cannot determine the value.

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.[6]

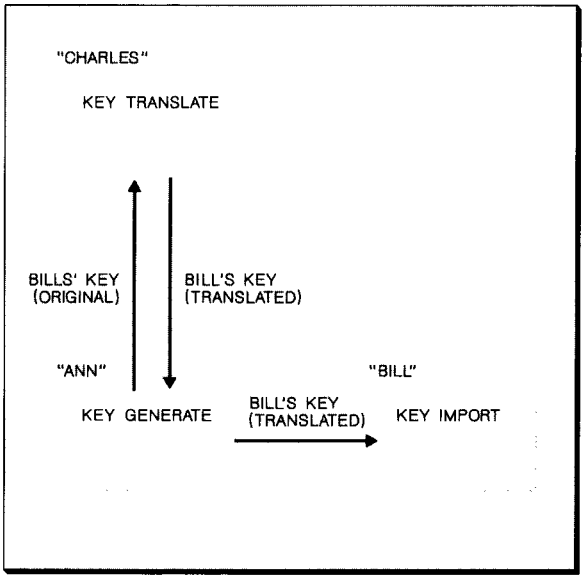
Key-distribution center. Use of the Key Generate service in a key-distribution center (KDC) is supported as described in the companion paper by D. B. Johnson et al.[1] The Key Generate service supports the key-distribution center (KDC) environment via generation of keys in EXEX (Exportable-Exportable) mode. The additional Transaction Security System generic key types beyond those defined in the Common Cryptographic Architecture and the additional granularity possible by using the Control Vector Generate service are fully supported.

Key Translate. Another method to accomplish key distribution is via use of a key-translation center (KTC). This ability is supported via the Key Translate service and allows Charles, at an intermediate node, to re-encipher an arbitrary key encrypted under one key-encrypting key to encryption under a different key-encrypting key, without the value of the arbitrary key appearing either in the clear or in a form that is usable at the key-translation center. This service is a product-specific extension.

Key-translation center. A typical key-translation center scenario which illustrates the use of an external key token is illustrated in in Figure 4. A typical process flow follows:

1. Ann calls Key Generate with a mode of OPEX, key type1 of PINGEN (PIN Generation) and key type2 of PINVER (PIN Verification), and also specifies via key label the appropriate EXPORTER key associated with Charles's node.
2. Ann keeps the generated operational PINGEN key token.
3. Ann sends the generated exportable PINVER key token to Charles.
4. Charles, once he gets the key token, calls Key Translate specifying an IKEYXLAT (Inbound Key Translation) key. This key has the same value as Ann's EXPORTER key. Charles also specifies an OKEYXLAT (Outbound Key Translation) key that has the same value as an IMPORTER key on Bill's system.
5. Charles sends the translated external key token to Bill, in this example, by way of Ann. Notice that Charles, as a normal user, does not have an operational form of the PINVER key, so he cannot use it on his system. He is acting strictly as a servant for Ann.
6. Bill is expecting to receive a PINVER key, so on receipt of the external key token Bill calls Key Import specifying the received PINVER key token, a key type of PINVER and the IMPORTER key associated with Charles's node. This produces an operational PINVER key token for Bill.
7. Ann calls the Clear PIN Generate service specifying the operational PINGEN key token and

Figure 4 Key translation center



creates PIN mailers for distribution to customers of her institution.
8. Bill can serve as a PIN Verification node for Ann, and yet Bill, as a normal user, cannot generate clear PINs with the PINVER key.
9. Eve cannot determine the value of the PINVER key as she does not know the value of any of the key-encrypting keys it was encrypted under.

For an example of the technical implementation details needed to support a key-translation center key-distribution environment with control vectors, see the companion paper by S. M. Matyas, A. V. Le, and D. G. Abraham.[6]

Control Vector Translate. The Control Vector Translate service can be thought of as a means to allow an installation to selectively break down the walls of key separation enforced via the control vector mechanism. It is exclusively a product-specific extension. Use of this service allows a user to change the control vector (or key type) associated with a key. The creation of a control vector translate table entry via the Cryptovariable Encipher service allows a specified mapping from one control vector to another. It requires special authorization, but the use of a control vector translate table entry to translate the control vector of a key from one control vector to another does not require special authorization—that is, it is available to a normal user. Control vector translation may be desirable when the ability to do two unrelated services with the same key value is required. Such a key is called a *bifunctional key*. An example where a bifunctional key may be needed is where it is mandatory to be compatible with an existing system that supports less key separation than the Transaction Security System.

PIN-management services. The Transaction Security System supports the three Common Cryptographic Architecture PIN-management services in full and has implemented several product extensions.

Clear PIN Generate. The Transaction Security System Clear PIN Generate service, as a product-specific extension, supports the generation of a PIN via the interbank PIN calculation method.

Encrypted PIN Translate. The Transaction Security System Encrypted PIN Translate service supports, as a product-specific extension, the OEM-1 PIN block format, which is compatible with the PIN block format used in NCR®, Diebold®, and Docutel® equipment. In addition to the Common Cryptographic Architecture defined method of PIN extraction for each supported PIN block, the Transaction Security System defines some additional methods of PIN extraction for certain PIN blocks. Also, the IPINENC (Inbound PIN Encrypting) and OPINENC (Outbound PIN Encrypting) keys may be defined with additional granularity via use of the Control Vector Generate service to support the following:

1. Assurance of propagation of PIN block format control throughout the network (that is, either format control was used at all nodes or was used at no nodes)
2. Execution of a PIN sanity (reasonableness) check to allow early detection of invalid PIN blocks at a PIN translation node
3. Caller specification of the sequence number in the two PIN blocks that use sequence numbers in their definition

Encrypted PIN Verify. The Transaction Security System supports, as a product-specific extension, the ability to verify a PIN that was calculated via the interbank PIN method. Also, the OEM-1 PIN block format is supported and the same additional methods of PIN extraction are supported as are supported in the Encrypted PIN Translate service.

Clear PIN Generate Alternate. The Clear PIN Generate Alternate service is an authorized service that is a product-specific extension. It is similar to the Clear PIN Generate service, except that the service supports specification of the customer-selected PIN being input in an encrypted PIN block, rather than in the clear. This service supports conversion from a PIN database method of PIN verification to a PIN calculation method of PIN verification.

Clear PIN Encrypt. The Clear PIN Encrypt service is an authorized service that is a product-specific extension. Its purpose is to take as input a given PIN value, put it into specified PIN block format and encrypt the PIN block. This is a way that an encrypted PIN block may be initially created.

Clear PIN Verify. The Clear PIN Verify service is an authorized service that is a product-specific extension. Its purpose is to verify an unencrypted

PIN. This service is supplied for compatibility with some existing applications, but is not recommended for use as a general solution because it implies that the PINs to be verified are not encrypted (which may therefore imply a security exposure), and because misuse of this service can simulate the Clear PIN Generate service.

Encrypted PIN Generate. The Encrypted PIN Generate service can be thought of as a Clear PIN Generate service followed by a Clear PIN Encrypt service, except that the Encrypted PIN Generate service does not need special authorization. This is a method of creating a PIN database directly. This service is a product-specific extension.

Encrypted PIN Generate Alternate. The Encrypted PIN Generate Alternate service generates an encrypted 64-bit block that can be decrypted by the Cryptovariable Decipher service to support remote PIN mailer creation. This service is a product-specific extension.

Cryptovariable Decipher. The Cryptovariable Decipher service is used to decrypt the encrypted output of the Encrypted PIN Generate Alternate service. Use of this service is expected to typically be done at a remote offline site to support remote PIN mailer creation. This service is a product-specific extension.

Key-token management services. All key-token management services are product extensions to the Common Cryptographic Architecture definition.

Control Vector Generate. The Control Vector Generate service supports a very granular specification of key-usage attributes. This allows sophisticated security administrators to follow the general security principle of limited function, that is, defining keys that have only the absolutely required functionality to accomplish a certain task.

A user may view the Common Cryptographic Architecture Cryptographic API as a subset of the Transaction Security System Cryptographic API. When a user goes between the Common Cryptographic Architecture and the product extensions supported by the Transaction Security System, the mapping of key-usage attributes supported on each system is made obvious via the control vector associated with each key. This is desirable because it allows a consistent perception by the user of the attributes of a key.

Control Vector generation is defined so that parameters can be ignored (that is, the defaults make sense in the general case), and the user is not required to learn the details of control vector specification. Certain generic control vectors are defined with attributes as defined by the Common Cryptographic Architecture Cryptographic API functionality. For example, if a user wants to generate the control vector for a MAC key and lets everything else default, then the generic MAC control vector used in the Common Cryptographic Architecture Cryptographic API definition is the result. As another example, the default specification of the control vector export control field is XPORT-OK (that is, Key Export is allowed) and not NO-XPORT (that is, Key Export is not allowed), as the generic usage of keys in the Common Cryptographic Architecture Cryptographic API is that all keys are able to be exported. If a Common Cryptographic Architecture Cryptographic API generic key (with an implied control vector export control field specification of XPORT-OK) is sent to a Transaction Security System (where the export control field is supported), the user knows that on the Transaction Security System this field may be reset to NO-XPORT via use of the Prohibit Export service.

Examples of key-usage granularity that is able to be specified via the Control Vector Generate service are as follows:

1. MAC and MACVER key specification requires that only a MAC with a length of 32 bits is to be processed.
2. CIPHER, ENCIPHER, and DECIPHER key specification that the ciphertext must be a multiple of eight. This might be specified when the creator of the key knows that the ciphertext will be translated at an intermediate node.
3. IMPORTER and EXPORTER keys may detail what services they may be used with and what types of keys they may operate with—that is, what key types they can encrypt.
4. PINGEN or PINVER keys may detail what services they may be used with and which method of PIN calculation algorithm they are to be used with.
5. IPINENC and OPINENC keys may detail what services they may be used with and what type of PIN block they are to be used with. They may also detail if PIN block format control must be maintained throughout all PIN-translation nodes in a network and may detail if the

contents of a PIN block should be sanity-checked before doing a PIN translation.

Key Token Build. The Key Token Build service is the method to create an enhanced key token. The Key Token Build service allows a user to prestore a control vector value in a key token before generation of a key by Ann. It also allows both pregeneration and postgeneration updates to specifications of service parameters associated with the key by Ann. This may be useful to ensure that a complete specification of the service options and parameter values is given to Bill, the intended user of the key. The enhanced key token can be thought of as defining a compact package of information that can help make Bill's use of a key more "idiot-proof."

Key Token Parse. The Key Token Parse service can be considered the inverse of the Control Vector Generate and Key Token Build services, as it takes the key token and outputs the parameters specifications that were used in the key token's creation. This service is necessary because it cannot be assumed that a caller will be able to easily interrogate bits in the key token in all programming languages.

Key Token Change. The Key Token Change service supports re-encrypting a key from encryption under the old master key to encryption under the current master key. It can also be used to note if the key is activated or deactivated.

Key record management services. The Transaction Security System defines the following key record management services: Key Record Create, Key Token Change, Key Record Read, Key Record Write, and Key Record List. All key record management services are extensions to the Common Cryptographic Architecture definition.

The Transaction Security System key record management services provide an interface that allows an application to manage key storage. The intent of the design is that the only method by which a key record may be created is via the Key Record Create service, and the only method by which a key record may be deleted from key storage completely is via the Key Token Change service. However, once a key record exists, it can be reused. This allows the system administrator to restrict access to the services that could fill up or completely erase key storage, while giving normal users the ability to, in effect, own a specific key record and process their own updates to the key record when needed.

Summary

The Transaction Security System supports much more functionality and granularity than previous DEA-based cryptographic systems. The design of the software architecture seeks to externalize this power by using a layered approach that masks much of the cryptographic complexity and requires users to learn only what they need to meet a specific security requirement. It conforms to SAA callable service guidelines. The software, by design, adheres to the Common Cryptographic Architecture Cryptographic API. It pursues the goals of generality, orthogonality, propriety, and consistency[7] as well as usability, interoperability, and program portability. Perhaps most important, the Transaction Security System was designed with the expectation that new functionality and security requirements will continue to emerge over time.

Acknowledgments

The author would like to thank the following people: Dennis ("Abe") Abraham of Transaction Security System Design, Charlotte, for the idea of a comprehensive cryptographic function set and his continuing input to the definition of the same, Stephen M. Matyas of the Cryptography Center of Competence, Manassas, for the pivotal architectural solutions, Mike Kelly, Gina Bourbeau, and Lucina Green of ICSF Design, Kingston, for providing models of callable services that follow SAA guidelines and their continuing comments, and Russ Prymak, An Le, and John Wilkins of the Cryptography Center of Competence, Manassas, for their constructive comments and help.

Cited references

1. D. B. Johnson et al., "Common Cryptographic Architecture Cryptographic Application Programming Interface," *IBM Systems Journal* **30**, No. 2, 130–150 (1991, this issue).

2. D. G. Abraham, G. M. Dolan, G. P. Double and J. V. Stevens, "Transaction Security System," *IBM Systems Journal* **30**, No. 2, 206–229 (1991, this issue).
3. *Transaction Security System Programming Guide and Reference*, SC31–2934, IBM Corporation (1991); available through IBM branch offices.
4. *Common Cryptographic Architecture Cryptographic Application Programming Interface*, SC40–1675, IBM Corporation (1991); available through IBM branch offices.
5. S. M. Matyas, "Key Handling with Control Vectors," *IBM Systems Journal* **30**, No. 2, 151–174 (1991, this issue).
6. 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).
7. A. J. Van de Goor, *Computer Architecture & Design*, Addison-Wesley Publishing Company, Inc., Reading, MA (1989), pp. 3–17.

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 in 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. His work on the Transaction Security System has involved 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.

Reprint Order No. G321-5432.