A public key extension to the Common Cryptographic Architecture

by A. V. Le
S. M. Matyas
D. B. Johnson
J. D. Wilkins

A new method for extending the IBM Common Cryptographic Architecture (CCA) to include public key cryptography is presented. The public key extension provides nonrepudiation via digital signatures and an electronic means to distribute Data Encryption Algorithm (DEA) key-encrypting keys in a hybrid Data Encryption Algorithm-Public Key Algorithm (DEA-PKA) cryptographic system. The improvements are based on a novel method for extending the control vector concept used in the IBM Common Cryptographic Architecture. Four new key types that separate the public and private key pairs into four classes according to their broad uses within the cryptographic system are defined. The public key extension to the CCA is implemented in the IBM Transaction Security System (TSS). This paper discusses both the public key extension to the CCA and the TSS implementation of this architectural extension.

The IBM Common Cryptographic Architecture (CCA)[1] was first described in a series of papers appearing in 1991.[2-7] The CCA is based on the Data Encryption Algorithm (DEA)[8] and a method for controlling key usage, based on control vectors.[9,10] The CCA consists of a set of cryptographic services supporting a wide range of data security applications—data confidentiality, data integrity, personal identification number (PIN) processing, and DEA key management. This set of services is accessible through an application programming interface (API) and is available on the IBM Enterprise Systems Architecture/390* Integrated Cryptographic Feature (ICRF) and the IBM Transaction Security System (TSS). A public key extension, or tower, provides additional cryptographic services via the API. This extension is available on the Transaction Security System.

The new public key extension to the CCA provides an additional 20 cryptographic services through the API.[11-13] These new services access a public key algorithm (i.e., the RSA [Rivest, Shamir, Adleman] algorithm[14]) located within secure hardware.

For those unfamiliar with the subject, the DEA enciphers a 64-bit block of plaintext into a 64-bit block of ciphertext under the control of a 64-bit cryptographic key. The key contains 56 independent key bits and 8 bits that may be used for error detection. Although the DEA itself is public, the keys must be kept secret. Hence, the DEA is called a *secret key algorithm*; its distinguishing feature is that the same key is used for enciphering and deciphering. For this reason, secret key algorithms

are also referred to as *symmetric algorithms*, and secret keys are also referred to as *symmetric keys*.

In marked contrast, the distinguishing feature of a *public key algorithm* is that one key is used for enciphering and another, different from the first, is used for deciphering. For this reason, public key algorithms are also referred to as *asymmetric key algorithms*. This concept was first introduced by Diffie and Hellman.[15] The enciphering key is made public and is referred to as the *public key*; the deciphering key is kept secret and is referred to as the *private key*.[16] The public and private keys are related, but the relationship is such that, given the known value of the public key, it is computationally infeasible to determine the value of the private key. Public key algorithms have eliminated the need to transport secret keys between communicating parties in order to establish a secure channel. When a pair of users A and B wishes to establish a secure channel, each user sends his or her public key to the other over the open channel. That the authenticity of the public key is a major concern is a widely known and accepted fact. The recipient of a public key must ensure that the key comes from a genuine sender—the sender with whom the receiver intends to communicate. Otherwise, an adversary might misrepresent the public key to user B, claiming that it belongs to user A. User B might then encrypt data for user A under the adversary's public key, thus exposing these data to the adversary.

A secret key algorithm such as the DEA provides services for both confidentiality and authenticity of information. However, public key algorithms are often more granular with respect to the services they provide. Some public key algorithms support only confidentiality, whereas others support only authenticity. The RSA algorithm provides both capabilities.

There are other differences between known secret key and public key algorithms, such as key size, block size, encryption and decryption speed and complexity, and key generation speed and complexity, all of which have been widely discussed in the literature. See, for examples, References 17 and 18.

Need for public key cryptography

A well-known problem for a DEA-based cryptographic system is the distribution of initial cryptographic keys for enabling any two devices in a network to establish a secure communication channel. These initial keys may be used for distributing subsequent keys, or for protecting data communicated between devices. In a network of n cryptographic devices, on the order of n^2 initial keys are needed in order for all pairs of devices to communicate securely. The most conventional method for distributing the keys has been via manual delivery such as with trusted couriers. This is generally not cost-effective for a large network consisting of thousands of cryptographic devices. The problem is remedied to a degree in DEA-based networks by using key distribution protocols based on a key distribution center (KDC) topology.[17,19] With the KDC topology, only n initial keys are required to establish secure communication between the KDC and n devices. However, a new element is needed: a trusted key distribution center. The disadvantage here is that a peer-to-peer key distribution protocol cannot be accomplished without involving the KDC each time an initial key is to be distributed.

With public key cryptography, electronic distribution of initial keys is more feasible and economical using a simple, widely known protocol. When a device wishes to establish a secure channel, it first generates a public and private key pair. The public key is sent to the intended receiving device on the open communication channel, and the private key is retained by the generating device. On receipt of the public key, the receiving device encrypts an initial DEA key-encrypting key with the public key and sends the encrypted key value to the originating device. Since the private key is known only to the originating, or generating device, only this device can decrypt the encrypted initial key-encrypting key to establish a secure communication session with the other device. (Of course, this assumes that the underlying public key algorithm supports secrecy, so that keys can be encrypted and decrypted.) As discussed earlier, however, the authenticity of the public key is a major concern. Various methods have been proposed for certifying and registering public keys, and for improving the integrity of the key distribution process.[17,20] Many of these methods require the involvement of trusted certification centers or authentication servers whose roles are similar to those of key distribution centers in secret-key-based key distribution. Even with that requirement, public-key-based key distribution is still considered more advantageous than secret-

key-based key distribution.[17,18,21] The advantages are these. First, with public-key-based key distribution, the certification center or the authentication server can be off line and key distribution is still possible. In contrast, with secret-key-based key distribution, on-line access to a key distribution center is usually needed each time the communicating parties establish an initial keying relationship. Second, in public-key-based key distribution, the degree of trust placed on the central authority (e.g., a certification center) is generally less than the degree of trust placed on the central authority in secret-key-based key distribution. This is because with distributed public keys, one needs to be concerned only with their integrity, whereas with distributed secret keys (i.e., DEA keys), one is concerned with both the secrecy and the integrity of keys.

Public key cryptography is well-suited to the digital signature mechanism that supports nonrepudiation applications.

Public key cryptography is also well-suited to the digital signature mechanism that supports nonrepudiation applications, which are applications that can establish the authenticity of an originator of a message or data. As a simple example, let A be the originator of a message M, sent with proof of authenticity to B. A first generates a public and private key pair (PU, PR).[22] The private key PR is retained by A, and the public key PU is sent to B. Assume that the public key algorithm used supports authentication applications and has the reversible property that $e_{PU}\{d_{PR}[H(M)]\} = H(M)$, where d_{PR} denotes the decryption transformation with the private key PR, and e_{PU} denotes the encryption transformation with the public key PU. Next, A calculates a hash value $H(M)$, also known as a message digest, on the message M, using a hash function. A then decrypts $H(M)$ with the private key, and appends the result to the message M. A also sends the appended message to B (assuming no message secrecy is desired). The decrypted value of $H(M)$, denoted as $d_{PR}[H(M)]$, is often called the digital signature on M. Signing on a hash value of a long message is much faster and easier than signing on the message itself. Upon receiving the message and the attached digital signature, B validates message M and verifies the fact that M originated from A as follows. B first encrypts the signature $d_{PR}[H(M)]$ with the public key PU to recover $H(M)$. Next, B produces a hash value on message M, with the same hash function used by A. The produced hashed value is then compared for equality with the recovered $H(M)$. If the comparison is successful, the authenticity and integrity of message M can be assured. B cannot forge the digital signature, because B does not have the knowledge of the private key PR. Similarly, A cannot disclaim the digital signature, because only A knows the value of the private key PR. The hash value provides integrity to the message by serving as a redundant value to the message. Designers of public key cryptographic systems must exercise care in the selection of the hash function to ensure that, with high probability, two different messages do not result in an identical hash value. Detailed treatment of hash algorithms used for calculating digital signatures can be found in References 17 and 23.

Public key algorithms are not the only means for producing digital signatures. Digital signatures can be produced with secret key algorithms. They can also be produced with special transformations that, strictly speaking, are not secret key or public key algorithms.[17,23] However, public-key-based digital signature schemes[14,24] remain the most preferred schemes for many users.

Although public key cryptography (i.e., the RSA algorithm) is capable of providing both privacy and authentication services, its wide usage has been usually in the areas of key distribution and digital signature applications. It has not been as widely used in general data encryption and decryption applications. This is due to the fact that public key algorithms typically involve modular exponentiation and are computationally intensive[25] and usually slow for real-time applications (e.g., instantaneous electronic conversations). The DEA, on the other hand, is very fast when compared to existing public key algorithms (PKA) and well-suited to routine encryption of bulk data. Thus, it is very desirable to have a hybrid

DEA-PKA system that can combine and take advantage of the good features in both DEA and PKA.

Currently, there is a wealth of published articles and reports presenting ideas, techniques, and protocols for hybrid secret key and public key cryptographic systems. Although most of the published materials are largely conceptual, some of them do offer practical realizations. Most notable are the Defense Advanced Research Project Agency (DARPA) Internet Mail system,[26] the Bell-Northern Research Integrated Services Digital Network (ISDN) terminal,[27] the Cylink CIDEC-LS** system,[28] the Digital Distributed System Security Architecture,[29] the MEMO system,[30] and the Secure Data Network System (SDNS).[31,32] The DARPA Internet Mail system and the Digital Distributed System Security Architecture use RSA as the public key method for distributing secret symmetric keys. The CIDEC-LS system and the MEMO system use the Diffie-Hellman method[15] for distributing secret DEA key-encrypting keys. The Bell-Northern Research terminal uses the Diffie-Hellman method to distribute DEA session keys, which are then used to encrypt data exchanged during a communication session. And the SDNS utilizes a secret government algorithm to exchange and authenticate keys.

This paper presents a seamless PKA extension to the existing IBM Common Cryptographic Architecture. Whereas the goal of this hybrid architecture is similar to other mentioned hybrid systems, the achieved result is a unique and highly integrated cryptographic system that takes advantage of the rich functionality of the DEA-based Common Cryptographic Architecture[2] and the key-usage enforcement method based on the recently developed control vector concept.[9]

Rationale and objectives

As its name implies, the public key extension to CCA is not intended to be a stand-alone architecture, but is an optional add-on to the already existing CCA. If a product implements the public key extension, that product must also implement the base Common Cryptographic Architecture. Hence, the mentioned advantages of public key cryptography are used to the fullest extent to enhance the present CCA. The aim is twofold: (1) provide new data operation services such as nonrepudiation via digital signatures, and (2) provide expanded key management services such as distribution of DEA keys via encryption with a public key.

The digital signature services are independent services and are added to the existing data operation services. The expanded key management services also make use of digital signatures to provide a high-integrity key distribution channel for the distribution of both PKA keys and DEA keys. This kind of signature, which is created and verified as an integral part of the key management services, is called a *system digital signature*. A different kind of signature, called an *application digital signature*, is generated and verified on user-supplied data via signature services available to application programs.

The expanded key management services emphasize the use of public key cryptography to distribute initial DEA key-encrypting keys and thus to eliminate the need for couriers and manual installation of the initial DEA key-encrypting keys. Once DEA key-encrypting keys are distributed via public key cryptography, subsequent DEA key-encrypting keys or data-encrypting keys can be distributed via existing CCA services. Thus, with the public key extension, hybrid key distribution is based on a three-level hierarchy, rather than a two-level hierarchy.[33] In the absence of a key management standard prescribing a key hierarchy, the authors adopted a three-level hierarchy for the following reasons:

- *Performance.* There are many applications where the data keys change rapidly from one transaction to another. In these applications, use of a two-level hierarchy might negatively affect system performance, because PKA encryption must be performed on each data key. With a three-level hierarchy, PKA encryption needs to be performed only on an initial DEA key-encrypting key. Thereafter, a DEA encryption is performed on each data key, and thus the performance impact is much less than that of the two-level hierarchy.
- *Flexibility.* The three-level hierarchy is more flexible than the two-level hierarchy, because the three-level hierarchy can be made to simulate a two-level hierarchy. For example, if PU, KEK, and KD represent a public key, a DEA key-encrypting key, and a DEA data key, respectively, then the single encrypted key value,

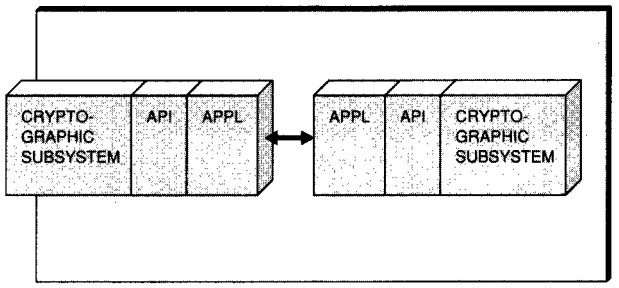
ePU(KD), in a two-level hierarchy, is replaced in a three-level hierarchy by two encrypted key values, ePU(KEK) and eKEK(KD). In contrast, the two-level hierarchy cannot be made to simulate the three-level hierarchy.

- *Security.* The three-level hierarchy simplifies the task of analyzing security in the key management design in that it ensures that the bridge from PKA to DEA is focused at one point only—the distribution of one key type (i.e., key-encrypting key). With a two-level hierarchy, there are potentially many different key types that may require distribution (e.g., privacy key, authentication key, PIN encryption key). That is, there are many bridges from PKA to DEA.

From customer requirements, governmental export regulations, and the rationale just presented, the following major objectives have been derived for the public key extension to the CCA:

- Provide PKA services with security and integrity level comparable to or better than existing CCA services.
- Support distribution of DEA key-encrypting keys with high integrity via PKA public keys.
- Support distribution of public keys with high integrity in both certification center and peer-to-peer environments.
- Support generation and verification of application digital signatures. Application digital signatures are generated on data specified by the application program (or user). For interoperability with other public key cryptographic systems, application digital signatures are also generated on system data (e.g., distributed public keys) for use in the key management services. However, greater integrity in the key distribution process can be achieved with system digital signatures discussed earlier in this paper.
- Support generation and verification of system digital signatures, which are generated on system data specified by the cryptographic subsystem, as an integral part of the key management services requiring high integrity.
- Comply with the well-accepted digital signature standard ISO/IEC IS 9796[34] in the generation and verification of both system digital signatures and application digital signatures.
- Prevent the system from being misused as a covert encryption channel by avoiding support for general data encryption and decryption capabilities with public key algorithms. PKA encryption is performed only on system-generated data. In cases where supplied data are encrypted, the encrypted data are not returned to the application program, but are compared in the hardware with application-supplied data.

Figure 1 Public key extension architectural model



As shall be seen, these major objectives have greatly influenced the design of the public key extension.

System overview

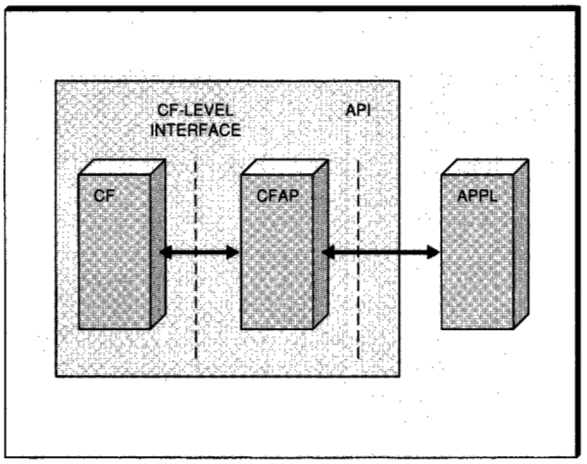
Figure 1 shows an architectural model of the public key extension in which two cryptographic applications interoperate. Each cryptographic application (APPL) interfaces to a cryptographic subsystem through an application programming interface (API). The model parallels the CCA architectural model presented in Reference 2. In the CCA architectural model, the cryptographic subsystem contains a set of cryptographic services that are invoked via the API. In the extended CCA architectural model, the cryptographic subsystem is expanded to include the following:

- New cryptographic services supporting the public key extension
- A new data structure, called the *profile vector*, used to configure operations of the cryptographic services

The cryptographic services and the profile vector can be implemented in hardware, software, or a combination of both, depending on the security, performance, and cost objectives at hand.

For the purpose of this paper, the functional aspect of public key extension is described in terms of the new API cryptographic services, because they matter most from a user's perspective. How-

Figure 2 Components of the cryptographic subsystem



ever, key management concepts developed for the public key extension are more easily discussed using a more granular architectural model in which the cryptographic subsystem is subdivided into software and hardware components. Thus, in the remainder of the paper, the public key extension and the cryptographic subsystem are discussed in terms of the specific implementation within the IBM Transaction Security System (TSS).

TSS implementation of the cryptographic subsystem

In the TSS implementation of the public key extension, the cryptographic subsystem is partitioned into two components: a hardware component called the cryptographic facility (CF) and a software component called the cryptographic facility access program (CFAP), as illustrated in Figure 2. The CF contains the cryptographic algorithms, storage for a small number of clear keys and cryptographic variables, and an instruction processor for executing a set of cryptographic instructions that may be invoked by the CFAP through a low-level interface. The CFAP also interfaces with the application programs through an API.

A typical cryptographic service request initiated by an application program at the API includes data and cryptographic keys. The CFAP processes the service request by executing one or more CF instructions. Appropriate parameter values are passed to each CF instruction at the CF-level interface. CF instruction outputs are returned to the CFAP at the same CF-level interface. A CF instruction output value may be used as an input to another CF instruction, or it may be returned to the application program.

Cryptographic facility. The CF is the heart of the cryptographic subsystem, as shown in Figure 3. The CF contains three major components: (1) encryption processor, (2) instruction processor, and (3) CF environment. The CF is implemented within a secure boundary, protected with tamper-resistant, tamper-detection, and key zeroization circuitry. This ensures that the CF can be accessed only through intended interfaces, and that clear keys and the results of intermediate steps of encipherment and decipherment are kept protected.

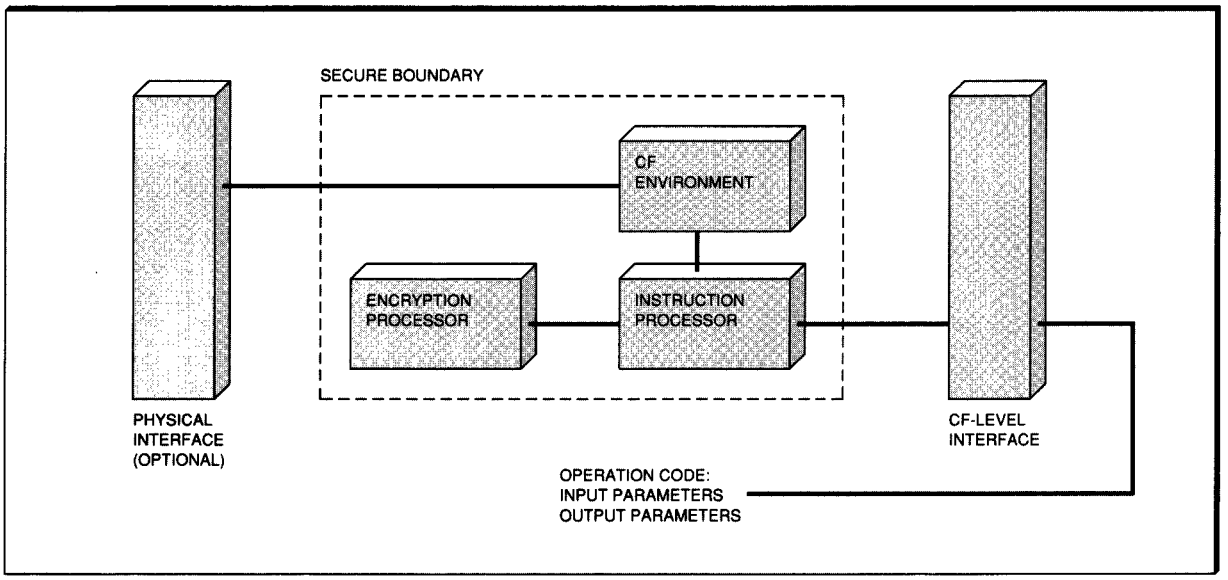
Encryption processor. The encryption processor implements the Data Encryption Algorithm (DEA) and one or more public key algorithms (PKA). The encryption processor is the basic engine for performing encryption and decryption operations required by the cryptographic instructions. The DEA portion of the encryption processor is part of the existing CCA. The PKA portion of the encryption processor is added to support the public key extension. The PKA portion includes a key generator and PKA encryption and decryption circuitry. The public key algorithm used in the public key extension for both digital signature and key distribution is the RSA algorithm. Provisions have been made to allow the CCA to be extended to support other public key algorithms as the market dictates.

The key generation method for RSA public and private keys follows the guidelines set forth by the digital signature standard ISO/IEC IS 9796. For digital signature purposes, the RSA key modulus length is between 512 and 1024 bits, inclusive. For DEA key distribution purposes, the RSA key modulus length is 512 bits.

Instruction processor. The instruction processor decodes and executes cryptographic instructions invoked by the CFAP at the CF-level interface. The instruction set in the instruction processor has been expanded to support the public key extension.

CF environment. The CF environment consists of a set of existing and newly added cryptographic

Figure 3 Cryptographic facility



variables—keys, flags, counters, and configuration data that collectively initialize and configure the CF. The newly added cryptographic variables include a 128-bit PKA master key (denoted KMP), which is a 128-bit DEA key-encrypting key, under which most PKA keys (i.e., public and private keys) are protected, and a special pair of keys called the *public device authentication key* and the *private device authentication key*. This special key pair is generated within the CF and is stored in the CF environment in clear form. The purpose of the device authentication key pair is discussed in a later section. The CF environment is expanded to include a *configuration vector*, which specifies operating conditions of the public key extension. The variables in the CF environment are initialized (1) via execution of certain CF instructions that read values supplied at the CF-level interface and load them into the CF environment, or (2) via an optional physical interface that allows values to be loaded into the CF environment (e.g., via an attached key entry unit). The configuration vector and the master keys in the CF environment are part of the public key extension key management concept that is to be discussed in a subsequent section.

Cryptographic facility access program. The cryptographic facility access program (CFAP) consists of a set of cryptographic services and data structures, including a profile vector and a software state vector, as shown in Figure 4.

Figure 4 Cryptographic facility access program

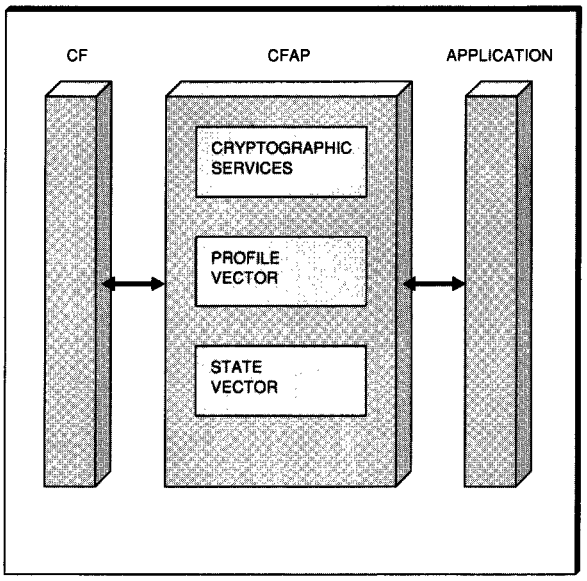


Table 1 Public key extension services

PKA subsystem management services
1. Profile vector build
2. Profile vector load
3. Device personalize
4. Configuration vector build
5. Environment reconfigure
6. Environment activate
7. Master key process
8. Master key set
PKA key token support services
9. PKA control vector generate
10. PKA key unit build
11. PKA key token build
12. PKA key token migrate
PKA key-management services
13. PKA key generate
14. Public key export
15. Public key import
16. Clear private key import
17. DEA key generate
18. DEA key import
PKA digital signature services
19. Application signature generate
20. Application signature verify

Cryptographic services. The existing CFAP has been expanded to include new cryptographic API services that support the public key extension. As with the existing CCA API services, the public key extension API services are in compliance with System Application Architecture* guidelines. Furthermore, the services have been designed with orthogonality in mind, so that product implementers can select and implement various levels or subsets of functionality to meet specific needs of different classes of customers.

Table 1 provides a categorized listing of the API-callable cryptographic services available in the public key extension, services that are implemented in the Transaction Security System. An overview of the services is provided in a later section.

Profile vector. The profile vector is a data structure used to configure and control the execution of cryptographic services. The profile vector consists of fields that specify the default values to be used for certain parameters in many cryptographic services when these parameters are not explicitly declared by the calling application. Some of these fields contain system-related specifications that either do not concern the average user or that are the set of specifications all users of the system must follow. By storing these specifications in the profile vector for use in many cryptographic services, the burden on the user to supply them is removed. Hence, the system becomes more user-friendly. Some other fields in the profile vector store specifications that concern users and are used frequently in many cryptographic services. By storing these specifications in the profile vector, the need for the user to supply them with each service call is eliminated. In many cryptographic services, these specifications will be used as default values for processing, unless specified otherwise by the user in the services. In this way, the system can accommodate both novice and experienced users.

The profile vector is envisioned to be set up by a network administrator and distributed individually to all systems in the network. The profile vector need not be the same for all systems, because some systems may have unique roles in the network. However, to ensure interoperability, certain fields (e.g., fields that contain system-related specifications) may be assigned the same values in all profile vectors by the network administrator. Once the profile vector is installed in a system, the system administrator may custom tailor the specifications of some fields to meet the needs of the users.

Software state vector. In addition to the standard data structure profile vector defined by the public key extension, the TSS implementation employs another data structure within the CFAP that is called the software state vector. The software state vector records various quantities dynamically produced within the system. It contains a flag byte that among other things, indicates whether the profile vector has been initialized on the system. It also includes status on the PKA master key and facilitates detection of situations where a PKA key is encrypted under an old PKA master key after the PKA master key has been changed. An important emerging requirement for all computing systems is the ability to support continuous operations. Some earlier cryptographic systems required that the system be shut down in order to change the PKA master key. In the public key

extension, when a cryptographic service detects such a condition on a PKA key, the service re-enciphers the PKA key with the current PKA master key and allows the service to proceed. This process is called the dynamic key update process.

The public key extension key management incorporates a new concept of a configuration vector.

We now present the public key extension key management concepts, on which much of the design of the CF is based.

Key management concepts

Master key. The concept of the master key in the public key extension is a continuation of the master key concept in the CCA. In the CCA, the DEA master key (denoted KM) is a DEA key-encrypting key used to encrypt all DEA keys stored outside the cryptographic facility. In the public key extension, the PKA master key is also a DEA key-encrypting key used to encrypt PKA keys residing outside the cryptographic facility. Conceptually, the DEA master key could also have been used to encrypt PKA public and private keys. However, having a separate PKA master key (denoted KMP), as implemented on the Transaction Security System, permits the public key extension to coexist with the base CCA system without affecting the existing CCA operations or existing DEA keys encrypted under the DEA master key. Of course, the system administrator may assign the same value for both master keys when installing the master keys on the system. The essential feature of the master key concept is to require only the master keys to be protected inside the secure hardware; other DEA keys and PKA keys may be encrypted under the master keys and stored outside the protected hardware.

Configuration vector. The public key extension key management incorporates a new concept of a configuration vector, which permits an installation to restrict the allowable cryptographic services that may be performed by the cryptographic hardware. Similar to a control vector, the configuration vector is a collection of encoded fields, except that the configuration vector is used to specify the operating conditions of a cryptographic facility and all keys of the system. This is in contrast to the control vector, which is used to specify usage of a key. Conceptually, a configuration vector is to the cryptographic facility and to all keys what the control vector is to a particular key. The aim of the architecture is to provide a configurable system permitting the installation to select the level of security and the cryptographic services allowable within the system.

An example of an encoded field in the configuration vector is the *certification field*, which specifies whether the cryptographic subsystem can function as a certification center (i.e., a central facility with some privileges in cryptographic capability, where systems in a network register public keys) or function only as a regular cryptographic subsystem.

Node identifier. The public key extension key management makes use of a node identifier to uniquely identify each cryptographic subsystem within a network. This is a nonsecret quantity administered and distributed by a network administrator. The node identifier of each cryptographic subsystem is loaded and stored in the CF environment. The node identifier is involved in the distribution of an initial DEA key-encrypting key in the following way. The service that performs the distribution of this initial key first generates a DEA key-encrypting key value and produces it in two forms. One form is a DEA key-encrypting key used to export DEA keys and is referred to as an EXPORTER key. The other form is a DEA key-encrypting key used to import DEA keys and is referred to as an IMPORTER key. There are some restrictions on the usage of this DEA key pair, but they are not important in the present discussion. The EXPORTER key is retained at the generating cryptographic subsystem for subsequent distribution of other DEA keys, and the IMPORTER is distributed to another cryptographic subsystem under a public key of the receiving cryptographic subsystem. At the generation time of this key pair, the node identifier of the generating cryptographic subsystem is copied from the CF to a field in a key record containing the IMPORTER key to be distributed. At the receiving cryptographic sub-

system, the node identifier imbedded in the distributed key record serves a dual purpose: (1) It is used in a stringent checking procedure (performed inside the CF) to ensure that the sending cryptographic subsystem is what it claims to be, and (2) it is used (also in a checking procedure inside the CF) as an anti-import value, that is, it is verified against the node identifier of the receiving cryptographic subsystem to ensure that this distributed IMPORTER key cannot be imported to the generating cryptographic subsystem. The reason the distributed IMPORTER key is not permitted to be imported to the generating cryptographic subsystem is as follows. If the IMPORTER key were imported to the generating cryptographic subsystem, it would coexist with its counterpart, the EXPORTER key, which was generated and retained at the generating cryptographic subsystem. The coexistence on a system of two DEA key-encrypting keys of the same clear values but with opposite usage attributes (i.e., one is for exporting keys and the other is for importing keys) would violate the unidirectional property of IMPORTER and EXPORTER keys. In many cryptographic applications, maintaining unidirectionality of DEA key encrypting keys is an important security requirement.

Cryptographic facility states. The public key extension key management also has a new concept of cryptographic facility states that controls the initialization of the CF. At any given time, the CF is defined to be in one of the following three states, in successive order: (1) *preinitialization*, (2) *initialization*, or (3) *run*. The preinitialization state is where personalization of the CF, such as loading a node identifier, is performed. The initialization state is where configuration of the CF, such as loading a configuration vector into the CF, is performed. The run state is the normal running state. For a cryptographic subsystem to be useful, its CF should reach the run state. In that state, a current PKA master key might be loaded, and all the public key extension cryptographic services are available for use.

Progressing the CF forward from one state to the next state is normal (e.g., from initialization state to run state). However, moving the CF from the run state to the initialization state or preinitialization state causes the master keys to be erased. This helps maintain a level of integrity of the CF and can prevent some forms of cheating committed by insiders. For example, it prevents a corrupt insider from reconfiguring a CF to an unauthorized role (e.g., by setting the certification field in the configuration vector to have the cryptographic subsystem act as a certification center). This is because reconfiguring the CF requires it to enter the initialization state where the master keys are erased and the CF is rendered useless. Suppose from the time a cryptographic subsystem is first brought up a corrupt insider configures the CF to a role not authorized by the network administrator. A routine audit of the configuration vector of the CF by the network administrator can help detect whether the CF is configured to its designated role. Again, if the insider alters the configuration of the CF to conceal its unauthorized role, the CF is rendered useless. The coupling of a CF configuration vector to its master keys can be used to enforce network security policy within a network that communicates via a public key protocol using a certification center.

PKA key types. In the public key extension, PKA keys are classified and separated according to their roles in supporting the key management services and digital signature services. The four types of public and private key pairs defined in the public key extension follow.

The *key-management key pair* supports key distribution services, where the key being distributed might be a public key or a secret DEA key-encrypting key. At a sending cryptographic subsystem, the public key-management key of a receiving cryptographic subsystem is used to encrypt the secret DEA key being distributed, and the private key-management key of the sending cryptographic subsystem can be used to sign the distributed DEA key (i.e., produce a digital signature). At the receiving cryptographic subsystem, the private key-management key of the receiving cryptographic subsystem is used to decrypt the distributed DEA key to recover the DEA key, and the public key-management key of the sending cryptographic subsystem is used to verify the digital signature produced on the distributed DEA key.

The *certification key pair* supports distribution of public keys via a certification center. The public and private certification key pair can be created only by a certification center. The private certification key always remains with the certification center, for the purpose of signing certificates containing distributed public keys. The correspond-

ing public certification key is normally distributed with integrity to other cryptographic subsystems of the network to verify the distributed certificates.

In contrast to the certification key pair, the key-management key pair is normally used in peer-to-peer key distribution. Within the context of key management, the certification key pair is separated from the key-management key pair to clearly differentiate the authority of a certification center from that of an ordinary cryptographic subsystem.

Except for private user keys, private keys of all other types remain with the system that originates them.

The *user key pair* supports application signature generation and verification services on user-supplied data. The private user key is used to generate an application signature on application program data. The public user key is used to validate an application signature.

The *device authentication key pair* is a special key pair that permits a cryptographic subsystem to authenticate itself and its own data stored in the CF (e.g., the configuration vector) to other cryptographic subsystems with high integrity. Unlike other public and private key pairs, the public device authentication key and private device authentication key are generated only during the preinitialization state and stored within the secure boundary of the CF. The keys of other types are encrypted under the PKA master key and can be stored outside the CF. Also, each cryptographic subsystem may have many key pairs of other types, but only one device authentication key pair. The device authentication keys are kept in the CF and are erased whenever a new node identifier is loaded into the cryptographic subsystem. That is, it is not possible to cheat by assigning two different node identifiers to the same authentication key pair, except by pure chance. To authenticate a cryptographic subsystem to other cryptographic subsystems within the network, it is necessary to separately authenticate that the public device authentication key actually comes from the cryptographic subsystem (via a service that exports a public key or via an audit procedure). This can be done in a secure environment before the cryptographic subsystem is shipped and installed; or it can be done in the field using a trusted third party that audits the cryptographic subsystem. Thereafter, anything signed with the private device authentication key can be used to prove that the signed object originates with the said cryptographic subsystem. Prior to auditing the cryptographic subsystem, the signatures can be used but they are only provisionally trusted.

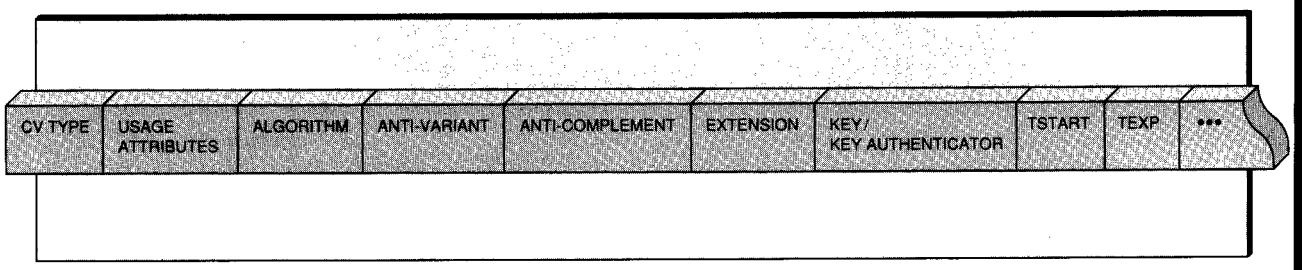
Except for private user keys, private keys of all other types remain with the systems that originate them. During key generation, a private user key can be provided in clear form as an output to an application or user; other private keys are produced under encryption of the PKA master key. A service exists to import a private user key. This supports a level of interoperability with public key systems that do not implement public key extension. No other private key type can be imported at a cryptographic subsystem, because this would undermine security.

The mechanism for cryptographically separating the PKA key types involves implementing a new set of control vectors, which is treated next.

PKA control vectors

The control vector method is a mechanism for controlling the usage of a cryptographic key and for enforcing the separation of cryptographic keys. The method was developed for the Common Cryptographic Architecture.[9] The control vector method calls for each cryptographic key to be coupled with its associated control vector in such a way that the nonsecret control vector must be specified to recover and use the key. The method provides for the creator of a cryptographic key to declare its intended usage attributes in the control vector associated with the key. Once a key is coupled to its associated control vector, its usage attributes are enforced during its lifetime to prevent misuse. The control vector method used in the public key extension is

Figure 5 Representative fields in a PKA control vector



built on the same architectural concepts and principles, but the cryptographic mechanisms differ.

Format of PKA control vectors. In this paper, we refer to the existing control vectors defined for DEA keys as DEA control vectors, and the new control vectors defined for public and private keys in public key extension as PKA control vectors. The DEA control vectors are 64 bits long, and the PKA control vectors are more than 128 bits long. Each PKA control vector consists of two parts: a system control block (SCB) and a user control block (UCB).

The SCB is 264 bytes long and contains system-managed control information associated with the key. Figure 5 shows several representative fields of a general SCB. They can be briefly explained as follows. The CV Type field indicates the key type of the PKA key. It indicates whether the key is a public or private key, and whether the public or private key is one of the following four types: (1) key-management key, (2) certification key, (3) user key, or (4) device authentication key. The Usage Attributes field indicates the cryptographic services in which the key can be used. The Algorithm field indicates the public key algorithm with which the key can be used. Currently, only the RSA algorithm is defined for this field. The Tstart (i.e., starting time) field and the Texp (i.e., expired time) field together indicate the time interval during which the key is valid for use. The Key/Key Authenticator field is a one-bit field that indicates whether the control vector is associated with a PKA key, if this bit has a value of 0. With a value called a key authenticator, the field is used to authenticate the PKA key, if this bit has a value of 1. Other fields shown in the figure are Anti-variant, Anti-complement, and Extension, which have important roles in distinguishing control vectors from other cryptographic variables. The Anti-complement bit (B'1') ensures that no two control vectors are complements of one another. The Anti-variant bit (B'0') in combination with the Anti-complement bit (B'1') ensures that no control vector can be equal in value to a variant mask as used in an earlier IBM key management scheme for separating cryptographic keys[9,3] and vice versa. The role of the Extension field is described in Reference 3.

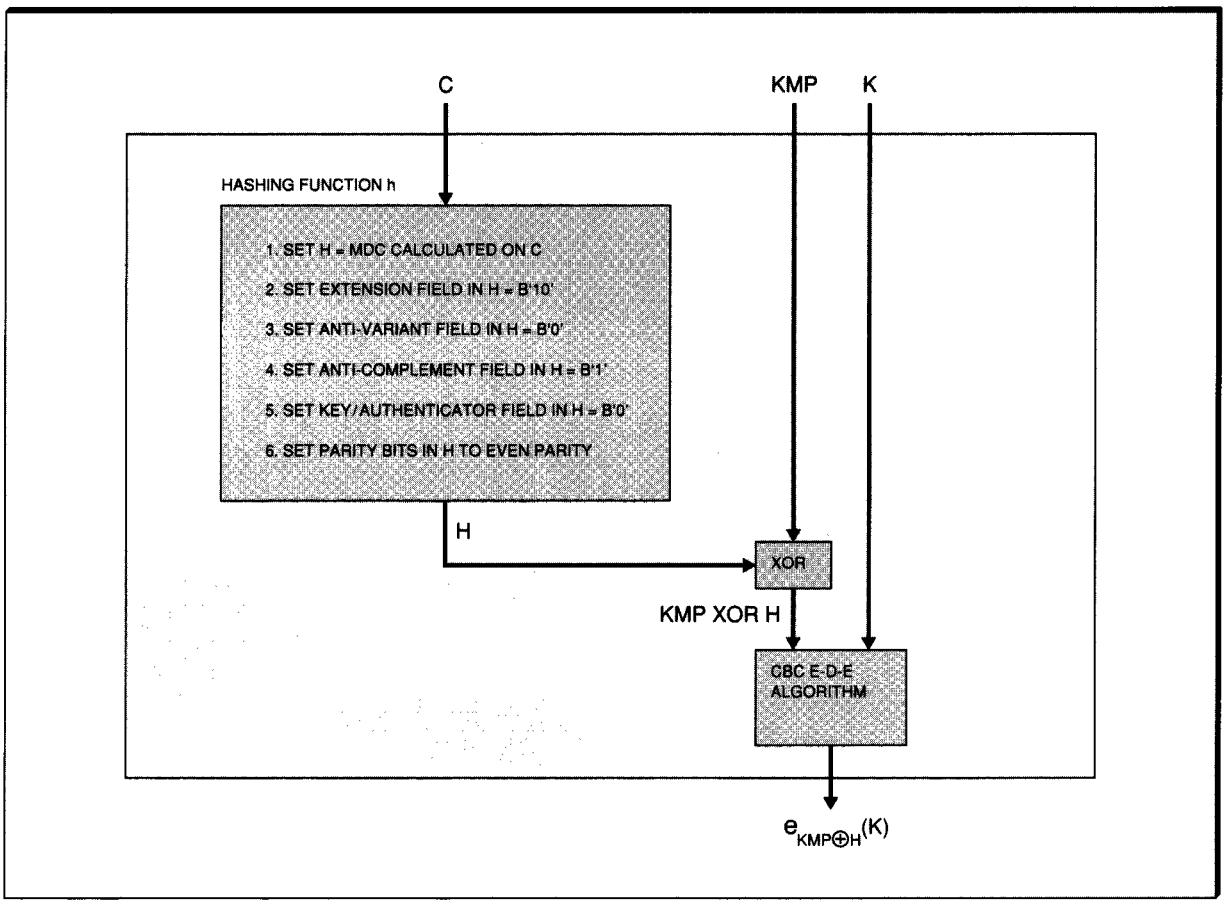
The UCB is 64 bytes long and contains user- or installation-managed control information associated with the key. The public key extension provides for a user or an installation to encode extra control information—deemed useful—in the UCB. However, the cryptographic facility does not check the values in the UCB; it is up to the application program to perform this checking.

Coupling of PKA control vectors to keys. From a system point of view, a PKA key can be classified as either an internal key or an external key. An internal key (also referred to as an operational key) is one that is in *internal form*, which is a form suitable for use on the system. An external key is one that is in *external form*, which is a form suitable for distributing to another system.

When a public key is exported to another system (i.e., the public key is transformed from the internal form to the external form), the coupling of the public key to its associated control vector is done via the digital signature, which is calculated both on the key and the control vector.

For an internal PKA key, the coupling of the key to its control vector is system-dependent and may depend on the secrecy and integrity requirements for public and private keys. In the public key ex-

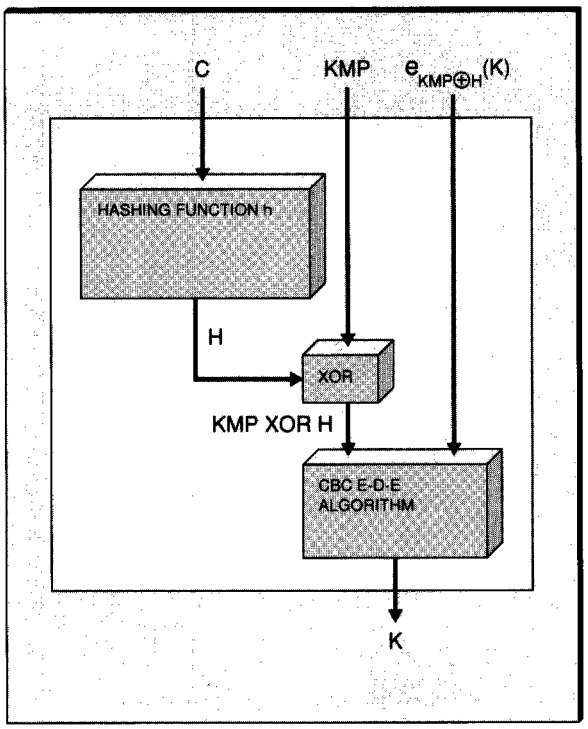
Figure 6 Encryption of a PKA key K



tension, the requirements on an internal public key are that (1) the integrity of the key value must be assured, and (2) the key must be coupled with its associated control vector to prevent misuse of the key. The requirements on an internal private key are that: (1) the integrity of the key value must be assured, (2) the secrecy of the key value must be assured, (3) the key must be coupled with its associated control vector to prevent misuse of the key, and (4) the key may be used only by the owner of the key or an authorized user. Access control methods are normally used to meet the fourth requirement on an internal private key. Here we discuss methods that may be used to meet the remainder of the aforementioned requirements on PKA internal public and private keys.

Figure 6 shows a method employed in the TSS to cryptographically couple a PKA key K to its associated control vector C, which consists of an SCB and a UCB. Usually, the value of K is represented in such a form that the performance of the public key algorithm can be optimized. The value of K is also expanded (e.g., via padding) to a block whose size is a multiple of 8 bytes, so that existing DEA encryption and decryption operations can be conveniently performed on K. This expansion process also includes prefixing the key value with a 64-bit random number to increase the appearance of randomness in the encrypted value of K. The long control vector C is first processed by a hashing function h to produce a 128-bit vector $H = h(C)$. The hashing function h is the same as that described in Reference 9, except that

Figure 7 Decryption of an encrypted PKA key



after an MDC hash value is calculated on C, the Anti-variant bit, the Anti-complement bit, the Key/Key Authenticator field, and the Extension field are set to appropriate values. Setting these fields to designated values helps protect against a class of potential attacks on control vectors that make use of the complementary property of the DEA, such as one described recently in Reference 35. Vector H is then exclusive-ORed with the PKA master key KMP, and the result, denoted KMP ⊕ H is used in an encryption algorithm, named CBC E-D-E, to encrypt K.

The CBC E-D-E algorithm is a generalization of the triple encryption scheme widely used in DEA-based key management schemes, such as ANSI X9.17,[36] for encrypting a 64-bit DEA key under a 128-bit DEA key-encrypting key. This is the basis of the term E-D-E, which stands for Encrypt-Decrypt-Encrypt. In the CBC E-D-E algorithm, the first 64 bits of KMP ⊕ H is used as a DEA key to encrypt K, using the cipher block chaining (CBC) mode of DEA encryption, with an initial chaining vector of zero. Next, a DEA decryption operation in the CBC mode is performed on the just-produced result, using the second 64 bits of KMP ⊕ H as the key and an initial chaining vector of zero. The decrypted result is then encrypted with the CBC mode of DEA encryption, using the first 64 bits of KMP ⊕ H as the key and an initial chaining vector of zero. The final result is an encrypted PKA key, denoted $e_{KMP \oplus H}(K)$. Since PKA keys are much longer than DEA keys, the cipher block chaining mode of DEA is used in the encryption of a PKA key, instead of the simpler electronic code book (ECB) mode of the DEA, to help eliminate patterns in the encrypted value of the PKA key.

The coupling method just discussed provides a unified way for meeting the requirements of assuring the secrecy of a key value and of coupling a key to its associated control vector [i.e., requirements (2) and (3)] for internal private keys. This method is also used in TSS to meet the requirement of coupling a key to its associated control vector [i.e., requirement (2)] on internal public keys. Although public keys in general need not be kept secret, one might well take advantage of the protection that already exists for private keys and use a single coupling method for both public and private keys. This coupling method does not require new cryptographic primitives to be added. Rather, it makes use of existing DEA encryption and decryption primitives already available in the CCA.

Figure 7 shows the method for decrypting an encrypted PKA key K, denoted $e_{KMP \oplus H}(K)$, using a PKA master key KMP and a PKA control vector C, assuming that the encrypted key is produced via the encryption method of Figure 6. The same hashing function h is applied to the long control vector C to produce H. H is then exclusive-ORed with the PKA master key KMP, and the result, denoted KMP ⊕ H, is used in the CBC D-E-D algorithm to decrypt $e_{KMP \oplus H}(K)$. The CBC D-E-D algorithm is just the inverse of the CBC E-D-E algorithm. That is, the sequence of operations, decrypt, encrypt, and decrypt, is used instead of the sequence encrypt, decrypt, and encrypt.

The requirement of assuring the integrity of key values [i.e., requirement (1)] on internal public and private keys can be satisfied via a concept called a key authenticator. Essentially, a key authenticator is a value derived from the value of a PKA public or private key and always accompanies the key for later verification of the key,

whenever the key is used. In TSS, the key authenticator is produced by applying the MDC-2 hashing algorithm described in Reference 9 to the value of the PKA key. The result is a 128-bit key authenticator, which is then encrypted under the PKA master key. The process of encrypting a key authenticator under the PKA master key KMP and the control vector C associated with the PKA key is the same as that of encrypting the PKA key (shown in Figure 6), except that a hashing function h' is used instead of the hashing function h. The hashing function h' is identical to the hashing function h, except that in step 5, the Key/Key Authenticator field is set to B'1' to indicate that $H = h'(C)$ is associated with a key authenticator.

The encrypted value of a PKA key, the control vector associated with the PKA key, and the encrypted key authenticator of the key constitute the internal form of the PKA key. In most requests to the CF for a cryptographic operation, a PKA key must be presented to the CF in internal form before processing can start. The process of decrypting an encrypted key authenticator is the same as that of decrypting the encrypted PKA key (shown in Figure 7), except that, again, the hashing function h' is used instead of the hashing function h.

When an encrypted PKA key, its associated control vector, and its companion encrypted key authenticator is presented (in internal form) to a cryptographic service (and ultimately to the CF of the cryptographic subsystem), the control vector is checked to ensure that the requested use of the key is permitted. If the checking is successful, the key recovery process is performed using the method illustrated in Figure 7. Otherwise, the service is aborted. Next, the key authenticator is recovered from the encrypted key authenticator. Another key authenticator is derived from the recovered PKA key, using the described method, and then compared for equality with the recovered key authenticator. If the comparison succeeds, the PKA key is considered to be genuine and is used in the public key algorithm to carry out the desired operation.

If an adversary cheats by specifying a control vector C' instead of the correct control vector C in the internal form of the PKA key, there are two possible outcomes. If the control vector checking fails, the cryptographic service is aborted. If the control vector checking succeeds, the recovered PKA key authenticator and the regenerated key authenticator do not compare and the service is aborted. Generally, the key authenticator provides detection of possible corrupted value of the recovered PKA key in the CF, regardless of whether this is caused by cheating or by inadvertent changes in the value of the key or the key authenticator due to hardware failures. In the absence of the key authenticator, a long time may pass before application programs can detect such a situation. Needless to say, all the checking and key recovery processes, as well as operations of the public key algorithm, are carried out inside the CF for security reasons.

In general, if the function that generates the key authenticator is a sufficiently strong cryptographic one-way function (e.g., MDC-2), the key authenticator need not be encrypted. However, encrypting the key authenticator provides extra security in implementations where a simpler or weaker one-way function is used for performance reasons.

It can be argued that the use of key authenticators may add additional DEA encrypt or decrypt cycles to the path. However, this does not materially affect performance, because most public key extension services themselves make use of a PKA encryption or decryption algorithm, which itself is very long compared to the DEA encryption and decryption operations performed on a key authenticator. In a hardware implementation, the processing of a key authenticator can be performed in parallel with other operations, and the overhead is considered negligible.

Of course, an implementer may use other methods to meet the requirements on public and private keys. For example, the requirement of coupling a PKA key to its control vector and the requirement of assuring integrity of a key value may be simultaneously met through the use of a special DEA key, called the *System Key Authentication Key* (SKAK). The SKAK is initialized and stored in a protected area of the cryptographic subsystem (e.g., the CF) and is used only by the subsystem during the internal processing of an internal PKA key. At the time a PKA key is created, the SKAK is used to calculate an authentication code on the concatenation of the clear value of the key and the control vector. The authentication code is then stored with the PKA key and the associated control vector. Together, they constitute the internal form of the PKA key. Needless to say,

if the PKA key is a private key, the value of the PKA key stored in the internal form of the key should appear in encrypted form.

When an internal PKA key is submitted to the cryptographic subsystem during a request for a cryptographic service, a new authentication code is calculated on the appropriate contents of the internal form of the key. The result is compared with the trial authentication code stored in the internal form. If the PKA key is a private key, the clear value of the key must be recovered and used in the process of calculating the new authentication code. If the comparison succeeds, the key is accepted as correct and normal processing continues for the requested service. If the comparison fails, the components of the internal key may have been corrupted and further processing is aborted.

It is suggested that the SKAK be a double-length DEA key and the authentication code calculation follow the cryptographic check function described in the ISO/IEC IS 9797,[37] and that the padding method used in the calculation be unambiguous to ensure that there are no synonyms.

Another method that may be used to singly meet the requirement of assuring the secrecy of the key value of a private key is as follows. The value of the private key is expanded via padding to form a block whose size is a multiple of 8 bytes, so that existing DEA encryption and decryption operations can be conveniently performed on the key value. The expansion process also includes prefixing the key value with a 64-bit random number to increase the appearance of randomness in the encrypted value of the key. The expanded value of the key is then encrypted under the PKA master key, using the CBC E-D-E algorithm of Figure 6. This method is just a simplification of the coupling method of Figure 6. The process of recovering the clear key value from the encrypted value of a private key is the inverse of the encryption process. That is, the CBC D-E-D algorithm is used instead of the CBC E-D-E algorithm.

Having demonstrated the features of key management and the PKA control vector mechanism—both of which constitute the underlying security mechanisms of the public key extension—we now present the functionality aspect of the public key extension. This includes the cryptographic tokens and the cryptographic services available to application programs via the public key extension API.

Cryptographic API tokens

In addition to the profile vector, the public key extension cryptographic services also use a set of new tokens, collectedly called PKA tokens. Most cryptographic services require keys and cryptographic variables to be passed in the form of a token. As in the CCA, the principle of the PKA tokens as designed in the architecture is to simplify processing of cryptographic variables produced and exchanged (1) among a cryptographic service and an application program, and (2) among application programs.

Each PKA token contains a token identifier, a PKA key or cryptographic variable, information associated with the key or cryptographic variable, and a token validation value (TVV). The token identifier distinguishes the type of token from other types. The TVV serves as a check sum, aiding in the detection of an invalid or erroneous token when one is submitted to a service for processing. When a key or cryptographic variable is transmitted among systems, it also appears as a token. Use of PKA tokens permits all relevant information regarding a key or cryptographic variable to be deposited in a single place, thus eliminating the need for carrying other information via additional methods. PKA tokens are classified into several types, including the following.

The *PKA internal key token* contains a public or private key in protected form and associated control information, including a control vector. Among the key tokens, the PKA internal key token is pivotal in many cryptographic services, because it contains a public or private key in the internal form ready to be processed by the services. This token is also known as the SA internal key token, where letter S (for "symmetric") indicates that a symmetric algorithm, such as DEA, is used to protect the key, and letter A (for "asymmetric") indicates that the key being protected is a public or private asymmetric key.

The *PKA external key token* contains a public key in clear (unencrypted) form and the associated control vector. A public key is in clear form only when it is exported to other systems, hence the term "external key." The PKA external key token is considered the foundation for distributing pub-

lic keys. This token is also known as the CA external key token, where letter C (for "clear") indicates that the key contained in this token is a clear key, and letter A indicates that the key is an asymmetric public key. Note that normally a private key is not kept in this form. However, to provide for interoperability with non-CCA systems, a private user key may be generated in this form and may be imported from this form. Such a private user key may only be used for digital signatures, and the control vector contains an indication that it was once in the clear. Maintaining the secrecy and integrity of the value of the private key when it is in the clear is the responsibility of the user, and cryptographic or physical methods may be used to achieve this.

Maintaining the secrecy and integrity of the value of the private key when it is in the clear is the responsibility of the user.

The *PKA-DEA external key token* contains a DEA key-encrypting key encrypted under a public key-management key. This key token is considered as the foundation for distributing DEA keys encrypted under public keys. This token is also known as the AS external key token, where letter A indicates that an asymmetric algorithm key is used to encrypt a key, and letter S indicates that the key being encrypted is a symmetric key, i.e., a DEA key.

The *system signature token* contains a system signature produced by the CF on keys and system data.

The *application signature token* contains an application signature produced by the CF on a hash value or a message digest of data supplied by application programs.

Features of the public key extension services

As discussed earlier in this paper, the public key extension services are structured at various levels of increasing functionality to enable product developers to select the level of functionality most appropriate to their customers. The major features of the API services available in the public key extension and implemented in the Transaction Security System are now presented.

PKA subsystem management services. The PKA subsystem management services allow an authorized application to initialize and manage the public key extension components of the cryptographic subsystem. The services that belong to this category are the profile vector build, profile vector load, device personalize, configuration vector build, environment reconfigure, environment activate, master key process, and the master key set services. The features of the PKA system management services are summarized below in the expected order of execution during system initialization.

The *profile vector build service* builds a profile vector that contains default values for variables used in other public key extension services. Examples of such variables are: node identifier; hashing method used in the construction of system signatures; hashing and padding methods used in the construction of application signatures; and default control vectors for each PKA key type. The types of digital signatures (i.e., system signature or application signature) allowed in public key import and DEA key import may also be specified.

The *profile vector load service* loads the specified profile vector into the cryptographic subsystem. The profile vector to be loaded may be the default profile vector or it may be one built by the profile vector build service. Once the profile vector is loaded, its specifications are made active.

The *device personalize service* makes the cryptographic subsystem unique in the network. The service first resets and invalidates all stored internal variables (except for the profile vector) in the cryptographic subsystem. Next, the service internally generates a public and private device authentication key pair. The service then loads this key pair, the associated PKA control vectors

(available from the profile vector), and the specified node identifier into a protected area of the cryptographic subsystem.

As mentioned previously, the value of the private device authentication key is always kept inside the CF of the cryptographic subsystem. There is no way a user, even a maximally authorized user, can determine its value. The public device authentication key may be exported to another cryptographic subsystem via the public key export service, to be described shortly. The private device authentication key may be used to digitally sign a PKA public key. Thus, the value of the PKA public key may be authenticated at another cryptographic subsystem by using the public device authentication key of the originating subsystem to verify the digital signature. The public device authentication key is used mainly by a user to digitally sign a new public key sent to a certification center to be certified.

The intent of the design of the device authentication keys is such that if a user reinitializes the system by calling the device personalize service again, such a reinitialization can be detected by the certification center, because a different device authentication key pair will be created for the subsystem. If a network administrator at a central subsystem knows the public device authentication key for a remote subsystem, and the network administrator sends a newly generated public key to a user at the remote subsystem asking the remote user to sign the public key with the private device authentication key of the remote subsystem, then on the subsequent verification of the digital signature, the network administrator is assured that the subsystem is not a device newly joining the network and has not in the past been personalized again.

The *configuration vector build service* builds a configuration vector containing specifications on the cryptographic capability of the system. One such capability is whether the system may operate as a certification center or whether this system is a normal cryptographic system.

The *environment reconfigure service* loads a specified configuration vector into the cryptographic subsystem. The configuration vector to be loaded may be the default configuration vector or one that was customized by the configuration vector build service. Once the configuration vector is loaded, its specifications are made active.

The *environment activate service* causes the cryptographic subsystem to enter the run state, where most of the public key extension services, including all PKA key management and PKA digital signature services, can be called.

The *master key process service* and the *master key set service* are used together to change the value of the PKA master key. Prior to changing the PKA master key, a user may wish to re-encipher all current operational keys from encryption under the current PKA master key to encryption under the new PKA master key by using the key token migrate service described in the following subsection.

PKA key token support. The PKA key token support services create and manage PKA key tokens. The PKA key token support services are the PKA control vector generate, PKA key unit build, PKA key token build, and the PKA key token migrate services.

The *PKA control vector generate service* builds a pair of PKA control vectors: one is associated with a PKA private key and the other is associated with a PKA public key. As mentioned previously, a PKA control vector contains encoded fields that specify the permitted usages of a PKA private key or public key. A PKA control vector is a component of a PKA skeleton key unit and is an input variable to the PKA key unit build service.

The *PKA key unit build service* builds a pair of public key extension data structures called skeleton key units. One key unit is for a public key and the other is for a private key. A PKA skeleton key unit contains a control vector and other information associated with a public key or a private key. A skeleton key unit is a component of various PKA key tokens, and is an input to the PKA key token build service.

The *PKA key token build service* builds a pair of PKA key tokens. One token is for a public key and the other is for a private key. The PKA key token built for a public key is always a PKA internal key token containing a skeleton key unit. The PKA key token built for a private key can be either a PKA internal key token or a PKA external key token, depending on the type of the private key. All the

relevant information about a PKA public or private key is stored together in a PKA key token.

The built PKA key token key pair, or just a single PKA key token of the pair, is subsequently supplied to various PKA key management services for processing. For example, the built key token pair may be supplied to the DEA key generate service for the generation of a PKA key pair. On completion of the PKA key generate service, the generated key pair is returned (usually in protected form) in the token key pair.

The *PKA key token migrate service* verifies the consistency of a PKA internal key token and, assuming positive verification, migrates the PKA key in the key token from protection under either the old or current PKA master key to protection under either the current or new PKA master key.

The PKA key token migrate service may be used to update the PKA internal key tokens after a PKA master key has been changed, or to prepare for an upcoming PKA master key change.

PKA key management services. PKA key management services provide services for generating and distributing PKA and DEA keys. The PKA key management services include the PKA key generate, public key export, public key import, clear private key import, DEA key generate, and the DEA key import services.

The *PKA key generate service* generates a pair of public and private keys. The generated key pair can be a pair of certification keys, key-management keys, or user keys, but not device authentication keys. The device authentication keys are special keys that can be generated only by a PKA system management service, called the device personalize service, described earlier in this paper. A PKA public key is always generated in internal form contained in a PKA internal key token. Because the RSA algorithm is used as the public key cryptographic algorithm in the public key extension, the value of the public exponent component of the RSA public key may be specified to be a randomly chosen value, a value of 3, or a value of 65 537. Because of the mathematical properties of the latter two values, use of either when performing RSA encryption is typically faster than when using a randomly-chosen value for the exponent of a PKA public key. In the public key extension, PKA public keys are used either for verification of a digital signature or to encrypt a DEA key for distribution to another public key extension system. Because of the better performance, one of these constant values is specified in some networks as the value of the exponent for all RSA public keys.

A PKA private key may be generated in internal form contained in a PKA internal key token or in external form contained in a PKA external key token. All private key types, except for the device authentication keys, can be generated in internal form. But only a private user key can be generated in external form. In this form it may be electronically distributed to another public key extension system. Meeting the requirements for the confidentiality or integrity of the value of the private user key when it is in this external form is the responsibility of the user.

The *public key export service* transforms a PKA public key from internal form to external form, so that it may be transmitted to another public key extension subsystem. All four public key extension public key types may be exported (i.e., public certification key, public key-management key, public user key, or public device authentication key). The external form of the public key may be digitally signed to allow for verification, by the recipient of the public key, of the key's authenticity and origin.

The *public key import service* imports a PKA public key transmitted from another public key extension subsystem in external form by transforming it to internal form so that the PKA public key may be used on this subsystem. All four PKA public key types may be imported. If the external form of the PKA public key is digitally signed, the digital signature may be verified. The cryptographic subsystem may be configured (via the configuration vector build and environment reconfigure services, described earlier in this paper) to require a digital signature on the external forms of public keys to be imported.

The *clear private key import service* imports a clear (unencrypted) private key in external form by transforming it to internal form, so that the private key may be used on the receiving subsystem. If the external form of the PKA private key is digitally signed, the digital signature should be verified before the clear private key import service is called. The only type of private key that

may be imported in external form is a private user key.

The *DEA key generate service* generates a double-length DEA key-encrypting key and produces it in two forms: one in operational internal form for use on this system, and the other in external form for distributing to a user at another system. The operational form of the DEA key is returned to the caller in a DEA internal key token and is encrypted by the current DEA master key of the creating system using a limited-use EXPORTER control vector. This limited-use EXPORTER control vector is differentiated from a CCA generic EXPORTER control vector in that the limited-use EXPORTER key can be used only in the CCA key generate service to generate and distribute DEA keys. It may not be used in the CCA key export service as either the EXPORTER key (i.e., a DEA key-encrypting key used to export other DEA keys) or the key to be exported.

The external form of the DEA key is returned to the caller in a PKA-DEA external key token and is encrypted by the specified public key-management key of the user on the receiving system. The PKA-DEA external key token contains the encrypted DEA key value and its associated limited-use IMPORTER control vector. This limited-use IMPORTER control vector is differentiated from a CCA generic IMPORTER control vector in that (after it is imported into operational form via the public key extension DEA key import service) the limited-use IMPORTER key can only be used in the CCA key import service to receive keys distributed from other systems. The operational form of the limited-use IMPORTER key may not be used in the CCA key generate service as an IMPORTER key (i.e., a DEA key-encrypting key used to import keys distributed from other systems). Also, the IMPORTER key may not be exported by the CCA key export service. The external form of the DEA key may be digitally signed by the private key-management key of the caller to allow later verification of its authenticity and origin.

The restriction on the usage capability of the EXPORTER and IMPORTER key pair produced by the DEA key generate service enhances security by supporting a single control point. The task of cryptanalysis on the hybrid key distribution is then confined to this key pair, as cryptanalysis on DEA key distribution in the CCA has been already performed.

The *DEA key import service* imports a DEA key distributed from another public key extension subsystem in external form to internal form so that the DEA key may be used on this subsystem. The DEA key to be imported is the limited-use IMPORTER key, mentioned in the DEA key generate service. The recovery process has some inherent verification of the integrity of the external form of the limited-use IMPORTER key to help ensure that the public key used to encrypt the key is paired with the private key used to recover the key. Also, if a digital signature exists on the external form of the limited-use IMPORTER key, it may be verified to further ensure the authenticity and origin of the key. The cryptographic subsystem may be configured to require a digital signature on all external forms of DEA keys.

Once the limited-use EXPORTER and IMPORTER keys are in place, a normal CCA generic EXPORTER and IMPORTER key pair (or alternatively, a generic IMPORTER and EXPORTER key pair) may be generated at the first system using the CCA key generate service.[1] The first generated key is operational on this system, and the second generated key is in exportable form and is then transmitted to the second system, where it is then imported to become an operational key. From that point on, normal CCA key distribution methods may be used. Alternatively, a data key pair can be generated on the first system in place of a generic IMPORTER and EXPORTER key pair (or a generic EXPORTER and IMPORTER key pair). One of the data key pair can then be distributed under the limited-use EXPORTER key to the second system for immediate use of general data encryption and decryption.

As noted earlier, the PKA key management services are designed with an objective of supporting key distribution in both peer-to-peer and certification center environments. We now present an example that shows the use of several public key extension cryptographic services in a peer-to-peer environment.

Sample peer-to-peer key distribution environment. Figure 8 illustrates a typical peer-to-peer key distribution environment in which a user named Ann wishes to distribute one or more DEA keys to a user named Bill.

A typical process flow is as follows:

1. Ann calls the PKA key generate service (at her system) to generate a pair of PKA internal key tokens containing a pair of public and private key-management keys. Bill does the same thing at his system to obtain a pair of PKA internal key tokens containing a pair of public and private key-management keys.

2. Ann calls the public key export service to generate a PKA external key token on her public key-management key. Assuming this is the initial public key-management key, Ann may choose to request that a digital signature be generated on her public key, using her corresponding private key as the signing key. The digital signature serves to couple the public key to its associated control vector. Bill also performs this task with his own public and private key pair.

3. Ann sends her public key (contained in the PKA external key token) and the attached signature to Bill with integrity. The methods of sending a key with integrity are discussed shortly. Bill also performs this task to send his public key and the attached signature to Ann.

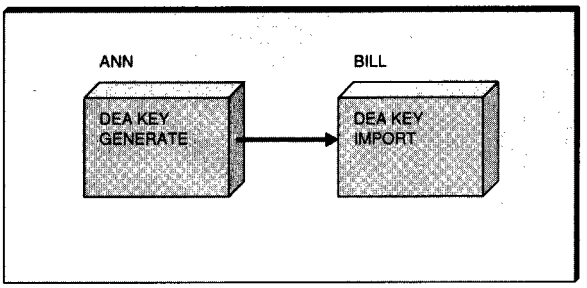
4. Ann calls the public key import service to import Bill's public key to her system for later use. If the signature attached to Bill's public key is signed with his private key, Ann may verify this signature with Bill's public key. Bill also performs this step to import Ann's public key to his system.

5. Ann calls the DEA key generate service to produce a pair of key tokens, one is a DEA internal key token containing an operational limited-use EXPORTER key, and the other is a PKA-DEA external key token containing an encrypted limited-use IMPORTER key. The key used to encrypt the limited-use IMPORTER key (for secrecy) is Bill's public key-management key. The key used to sign the limited-use IMPORTER key digitally (for integrity) is Ann's private key-management key. Ann sends the resulting PKA-DEA external key token and the digital signature to Bill. If symmetry is desired, the roles may be reversed, but this is not required to have a usable DEA key exchange.

6. Bill calls the DEA key import service to verify the digital signature and import the limited-use IMPORTER key. If symmetry is desired, Ann also performs this task.

7. From this point on, standard CCA key management may be performed. For example, the key

Figure 8 Peer-to-peer key distribution environment



distribution scenario may be continued as follows:

- Ann calls the CCA key generate service to produce a pair of message authentication code (MAC) keys, keeping the operational key for use on her system and sending the other key (in exported form) to Bill. The operational MAC key may be used to generate a MAC on the message containing the exported MAC key. If symmetry is desired, the roles of Ann and Bill may be reversed.
- Bill calls the CCA key import service to import the MAC key, and verifies the associated MAC by calculating a MAC for the key distribution message and comparing it for equality with the transmitted trial MAC.
- Ann calls the MAC generate service to calculate a MAC on a subsequent arbitrary message to Bill.
- On receipt of the message, Bill calls the CCA MAC generate service and compares the generated MAC with the transmitted MAC. Assuming they are equal, Bill accepts Ann's arbitrary message as genuine.
- Bill can also send an arbitrary message to Ann with integrity.

8. When Ann decides to change her public key-management key, her current private key-management key can be used to sign the PKA external key token for the new public key-management key. Bill may also perform this task. In this way, a chain of integrity of public key-management keys is maintained.

Acquiring the initial public key with integrity. As mentioned previously, the value of a distributed public key need not be kept secret, but it is re-

quired to have integrity. If another user with a different private and public key pair can cause his or her public key to be mistaken for a public key of a legitimate user, that user can pretend to be the legitimate user, at least insofar as digital signatures are concerned. This means that an information channel with sufficient integrity must be established between the communicating parties to distribute the initial public key. Once an initial public key has been established, it may be used to verify a digital signature on other public keys to assure the receiving party of the integrity and origin of the new public key.

Various techniques and procedures have been proposed for distributing public keys with varying levels of integrity.[17,20] It is noteworthy that, although none of the proposed techniques provides a perfect solution to the public key distribution problem, some of them have been found useful for various applications. Here, we present some examples of noncryptographic methods for distributing an initial public key with integrity that might be used:

- Publish the initial public key PKA external key token or a hash value of the token in a widely-circulated document. The intent is to ensure validity of the value by broad distribution. The hash value may be calculated by calling the CCA MDC generate service or another strong one-way function. When the PKA external key token is later electronically distributed, it can be verified by rehashing the received key token and ensuring that the actual hash value matches the published hash value.
- Distribute the public key or the hash value in multiple ways and use the public key only if all the values are received and all agree on the value.
- If very high security is required, deliver the key value or hash value to the other party personally or via a trusted courier.

PKA digital signature services. The PKA digital signature services support the generation and verification of application digital signatures that provide for data integrity and source nonrepudiation for user-specified data. The PKA digital signature services are the application signature generate and application signature verify services. The digital signatures generated and verified in these services are called the application signature. There is also another type of digital signature in the public key extension, called the system digital signature that is generated and verified on system data (e.g., keys and other cryptographic variables) as an integral part of many PKA key management services, such as the public key export and DEA key generate services.

The application signature generate service generates an application digital signature on the hash value of user-supplied data, using a private certification key, a private key-management key, or a private user key.

The application signature verify service verifies an application digital signature on the hash value of user-supplied data, using a public certification key, a public key-management key, or a public user key.

The method for producing digital signatures conforms to the part of the ISO/IEC IS 9796 standard that is specified for the RSA algorithm.

Prior to calling the application signature generate and application signature services, application programs must calculate the hash value on user-supplied data. This is to provide users with the flexibility of using any hash function preferred by users. The hash value can be calculated via the CCA MDC generate service, which supports the MDC-2 and MDC-4 hash algorithms.[9,38] Otherwise, it can also be calculated via user-supplied functions that support the hash algorithms preferred by users. The length of the hash value on data to be signed must conform to the following ISO/IEC IS 9796 requirement for signatures produced by the RSA algorithm:

$$(\textit{length of hash value in bytes}) * 16 \leq (\textit{modulus length in bits}) + 2$$

With the length of the RSA modulus ranging from 512 to 1024 bits in the public key extension (for digital signatures), this requirement does not appear to be a problem for most existing and emerging hash algorithms, including the secure hash algorithm (SHA).[39]

The hash function used in the calculation of the hash value must be indicated to the application signature generate service, so that the service can communicate this indication to the verifier of the application signature via the field called the *hash algorithm* field in the application signature token.

On receipt of the application signature token and the data on which the application signature is produced, the verifier of the signature must first extract the hash algorithm indication in the application signature token and then use the indicated hash function to calculate a hash value on the data before invoking the application signature verify service to verify the application signature. Of course, if the length of the data to be signed conforms to the above length requirement, a user may choose to sign directly on the data.

When a first user calls the application signature generate service to generate an application digital signature on specific data and a second user calls the application signature verify service to verify the application digital signature on the first user's data, the second user may be assured of the integrity of the first user's data. The second user may also be assured that those data are actually signed by the first user, using the first user's private key. Because the public key is public information, an impartial third party may verify that the particular public key associated with the first user does actually verify the digital signature associated with the data. Thus these services provide support for both data integrity and source nonrepudiation. However, note that if an imposter has access to or knowledge of the first user's private key, the imposter will be able to generate digital signatures associated with the first user. This would allow the imposter to pose as the first user. For this reason, good security practice requires that the ability to determine the value of a private key or the ability to use a private key requires the identification and authentication of the user as the valid user of the private key.

Because the system signature is produced and verified as an integral part of many PKA key management services, these services maintain an unbroken chain of integrity from the point of system signature generation to the point of system signature verification. Thus it is important that the application signatures services cannot be misused to subvert the intended integrity of the system signatures. The public key extension key management architecture distinguishes between system signatures and application signatures according to the following signature production rule. A system signature is generated from a special signature record, called a system signature record. The system signature record is a 253-bit record containing important control information and a 128-bit hash value calculated on the data to be signed. On the other hand, an application signature is generated from a hash value supplied by the caller that contains a whole number of bytes. Because the system signature record is not a whole number of bytes, cryptographic separation is maintained between system signatures and application signatures. The cryptographic service that produces an application signature requires the input hash value to contain a whole number of bytes and thus cannot be misused to produce system signatures. Thus, the integrity of the system signature is preserved by the key management architecture.

Concluding remarks

Major features of the public key extension to the Common Cryptographic Architecture have been described. Although the concept of a hybrid PKA-DEA cryptographic system is not new, the public key extension provides a unique approach toward solving problems in both conventional and public-key cryptography. The control vector concept has been successfully integrated into the architecture to separate keys and prevent their misuse. Although the control vector is used in a different way from the Common Cryptographic Architecture, the security objectives in the Common Cryptographic Architecture have been maintained and improved in the public key extension. The public key extension has been designed with flexibility in mind so that it may be extended to support new public key algorithms and new cryptographic services, as the market dictates.

Acknowledgments

The authors wish to acknowledge their former colleagues, William Rohland and William Martin, who participated in the design of the public key extension. The authors also wish to thank C. H. Meyer, C. Holloway, and P. Maclaine-Pont for their valuable inputs during the design process. We also thank D. Abraham, G. M. Dolan, J. S. Stevens, B. E. Fitzpatrick, T. W. Arnold, and S. E. Wince for their constructive comments during the course of incorporating the public key extension into the Transaction Security System.

Cited references and notes

1. *Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference*, SC40-1675, IBM Corporation (1992); available through IBM branch offices.
2. D. B. Johnson, G. M. Dolan, M. J. Kelly, A. V. Le, and S. M. Matyas, "Common Cryptographic Architecture Cryptographic Application Programming Interface," *IBM Systems Journal* **30**, No. 2, 130–150 (1991).
3. S. M. Matyas, "Key Handling with Control Vectors," *IBM Systems Journal* **30**, No. 2, 151–174 (1991).
4. 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).
5. P. C. Yeh and R. M. Smith, Sr., "ESA/390 Integrated Cryptographic Facility: An Overview," *IBM Systems Journal* **30**, No. 2, 192–205 (1991).
6. 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).
7. D. B. Johnson and G. M. Dolan, "Transaction Security System Extensions to the Common Cryptographic Architecture," *IBM Systems Journal* **30**, No. 2, 230–243 (1991).
8. American National Standard X3.92-1981, *Data Encryption Algorithm*, American National Standards Institute, New York (December 31, 1981).
9. S. M. Matyas, "Key Processing with Control Vectors," *Journal of Cryptology* **3**, No. 2, 113–136 (1990).
10. B. Brachtl, S. M. Matyas, and C. H. Meyer, *Control Use of Cryptographic Keys via Generating Station Established Control Values*, U.S. Patent No. 4,850,017 (July 18, 1989).
11. *Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference—Public Key Algorithm*, SC40-1676, IBM Corporation (1993); available through IBM branch offices.
12. *Transaction Security System Programming Reference: Volume II, Public Key Cryptography*, SC31-2888, IBM Corporation (1992); available through IBM branch offices.
13. *The IBM Transaction Security System Concepts and Programming Guide: Volume II, Public Key Cryptography*, GC31-2889, IBM Corporation (1992); available through IBM branch offices.
14. R. L. Rivest, A. Shamir, and L. Adleman, "A Method of Obtaining Digital Signatures and Public Key Cryptosystems," *Communications of the ACM* **21**, No. 2, 120–126 (February 1978).
15. W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory* **IT-22**, No. 6, 644–645 (November 1976).
16. In this paper, the term *private key* refers to a private key of public key algorithms. The term *secret key* refers to a secret key of symmetric algorithms, such as DEA. That is, the terms private key and secret key do not interchange.
17. D. W. Davies and W. L. Price, *Security for Computer Networks, Second Edition*, John Wiley & Sons, Inc., New York (1989).
18. J. Nechvatal, "Public Key Cryptography," *Contemporary Cryptology: The Science of Information Integrity*, G. J. Simmons, Editor, IEEE Press, Piscataway, NJ (1992), pp. 177–288.
19. D. Branstad, "Encryption Protection in Computer Data Communications," *Proceedings of the Fourth Data Communication Symposium*, Quebec City, Canada (October 7–9, 1975).
20. S. M. Matyas, "Public Key Registration," *Advances in Cryptology—CRYPTO '86 Proceedings*, A. M. Odlyzko, Editor, Springer-Verlag, New York (1987), pp. 451–458.
21. C. S. Kline and G. J. Popek, "Public Key vs. Conventional Key Encryption," *American Federation Information Processing Societies Conference Proceedings, Vol. 48: National Computer Conference*, R. E. Merwin, Editor, New York, June 4–7, 1979, AFIPS Press, Montvale, NJ (1979), pp. 831–837.
22. By convention, in this paper the public key PU is used for enciphering, and the private key PR is used for deciphering.
23. C. J. Mitchell, F. Piper, and P. Wild, "Digital Signatures," *Contemporary Cryptology: The Science of Information Integrity*, G. J. Simmons, Editor, IEEE Press, Piscataway, NJ (1992), pp. 177–288.
24. T. El Gamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory* **IT-31**, No. 4, 469–472 (July 1985).
25. P. G. Comba, "Exponentiation Cryptosystems on the IBM PC," *IBM Systems Journal* **29**, No. 4, 526–536 (1990).
26. J. Lint and S. T. Kent, "Privacy for DARPA-Internet Mail," *The Proceedings of the Twelfth National Computer Security Conference* (1989), pp. 215–229.
27. W. Diffie, B. O'Higgins, L. Strawczynski, and D. Steer, "An ISDN Secure Telephone Unit," *Proceedings National Computer Forum* **41**, No. 1, 473–477 (1987).
28. J. Graff, "Key Management Systems Combining X9.17 and Public Key Techniques," *The Proceedings of Seventh Annual ISSA Conference* (1990), pp. B-9-1 to B-9-8.
29. M Gasser, A. Goldstein, C. Kaufman, and B. Lampson, "The Digital Distributed System Security Architecture," *The Proceedings of the Twelfth National Computer Security Conference* (1989), pp. 305–319.
30. B. Schanning, S. A. Powers, and J. Kowalchuk, "MEMO: Privacy and Authentication for the Automated Office," *Fifth Conference on Local Computer Networks*, (1980), pp. 21–30.
31. R. Nelson, "SDNS Services and Architecture," *The Proceedings of the Tenth National Computer Security Conference* (1987), pp. 153–157.
32. P. A. Lambert, "Architectural Model of the SDNS Key Management Protocol," *The Proceedings of the Eleventh National Computer Security Conference* (1988), pp. 126–128.
33. In this paper, the number of levels of key distribution hierarchy are defined relative to the data key being distributed. The term *three-level key distribution hierarchy* refers to one in which a public key is used to distribute a DEA key-encrypting key, which in turn is used to distribute a DEA data key. The term *two-level hierarchy* refers to one in which a public key is used to distribute a DEA data key directly.
34. ISO/IEC IS 9796-1991, *Information Technology—Security Techniques—Digital Signature Scheme Giving Message Recovery*, International Organization for Standardization, CH-1211, Geneva, Switzerland.
35. D. Longley and S. M. Matyas, "Technical note—Complementarity Attacks and Control Vectors," *IBM Systems Journal* **32**, No. 2, 321–325 (1993).
36. American National Standard X9.17-1985, *American Na-*

tional Standard for Financial Institution Key Management (Wholesale), American Bankers Association, Washington, D.C. (1985).
37. ISO/IEC IS 9797-1989, *Data Cryptographic Techniques—Data Integrity Mechanism Using a Cryptographic Check Function Employing a Block Cipher Algorithm,* International Organization for Standardization, CH-1211, Geneva, Switzerland.
38. ISO CD 10118 (as of July 21, 1992) *Data Cryptographic Techniques—Hashing Operation Using a Symmetric Block Cipher Algorithm,* International Organization for Standardization, CH-1211, Geneva Switzerland.
39. "A Proposed Federal Information Processing Standard for Secure Hash Standard," *Federal Register Announcement* (January 31, 1992), pp. 3747–3749.

Accepted for publication March 15, 1993.

An V. Le *IBM Federal Systems Company, 9500 Godwin Drive, Manassas, Virginia 22110.* Mr. Le is an advisory engineer in the Cryptographic 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 Cryptographic Center of Competence in Manassas, and has since been working in the areas of cryptographic algorithms and architectures. Mr. Le is a codeveloper of the Common Cryptographic Architecture (CCA) implemented in several IBM products. He is also a key designer of the public key extension to the CCA implemented in the IBM Transaction Security System. Mr. Le holds 14 issued patents, several patents on file, and has published over 20 technical papers and technical disclosures in the area of computer design and cryptography. He has received six IBM Invention Achievement Awards, several informal awards, two IBM FSC Manassas Laboratory Technical Publication Plateau Awards, and one IBM FSC President's Patent Award for two patents on key management for public key cryptography.

Stephen M. Matyas *IBM Federal Systems Company, 9500 Godwin Drive, Manassas, Virginia 22110.* Formerly a member of the Cryptography Competency Center at the IBM Kingston Development Laboratory, Dr. Matyas is currently manager of the Secure Products and Systems department at Manassas, Virginia. He participated in the design and development of all major IBM cryptographic products, he played a lead role in the design of the IBM Common Cryptographic Architecture, and is the inventor of the control vector concept. Dr. Matyas holds 28 patents and has published numerous technical articles covering many 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. degree in mathematics from Western Michigan University and a Ph.D. degree in computer science from the University of Iowa. He is the recipient of an IBM Outstanding Innovation Award, an IBM FSC President's Patent Award, and is an IBM eighteenth-level inventor. Dr. Matyas is currently an IBM Senior Technical Staff Member.

Donald B. Johnson *IBM Federal Systems Company, 9500 Godwin Drive, Manassas, Virginia 22110 (electronic mail: dbj@manvm1.vnet.ibm.com).* Mr. Johnson received a B.A. in mathematics from Oakland University, Rochester, Michigan, and an M.S. in computer science from Union College, Schenectady, New York. He joined the IBM Field Engineering Division in 1974 as a program support representative. In 1978, he joined the 8100/DPPX Change Team in Kingston, New York. In 1982, he worked on DPPX/APL development in Lidingo, Sweden. Since 1987 Mr. Johnson has worked in the Cryptographic Center of Competence in Manassas, Virginia. He is the chairman of the Confidentiality and Integrity Work Group of the IBM Security Architecture Board. He is the recipient of an Outstanding Innovation Award for his contributions to the Common Cryptographic Architecture application programming interface and an IBM FSC President's Patent Award. Mr. Johnson has achieved the sixth plateau in the IBM Invention Achievement Award program and holds 15 patents with other patents pending because of his contributions to the IBM Common Cryptographic Architecture and the IBM Transaction Security System product architecture. Mr. Johnson has achieved IBM FSC Manassas Author Recognition and has authored two papers on cryptography, which have been published in the *IBM Systems Journal*. One of these papers won an award as one of the best papers published that year from the IBM Manassas site. Currently, he is an advisory programmer.

John D. Wilkins *IBM Federal Systems Company, 9500 Godwin Drive, Manassas, Virginia 22110 (electronic mail: jwilkins@manvm1.vnet.ibm.com).* Mr. Wilkins received a B.S. in mathematics and computer science from Bucknell University in 1980. He is currently a staff programmer in the Secure Products and Systems Department in Manassas. His responsibilities include the design and evaluation of IBM cryptographic products, protocols, and architectures. He is a codeveloper of the IBM Common Cryptographic Architecture that was implemented in the IBM ESA/390 Integrated Cryptographic Feature and IBM Transaction Security System. Mr. Wilkins is a sixth-level inventor and holds 15 cryptography-related patents. He received an IBM FSC President's Patent Award in 1991 for two inventions relating to key management in public key cryptography. He has coauthored papers on public key cryptography, an exportable cryptographic algorithm, and network security issues for the IBM Interdivisional Technical Liaison on Computer Security and for the UNIX Internal Technical Exchange. Prior to being hired by IBM in 1986, he was a lead software engineer for the signal collection and analysis system in a prominent U.S. Department of Defense intelligence program.

Reprint Order No. G321-5521.