A key-management scheme based on control vectors

by S. M. Matyas
A. V. Le
D. G. Abraham

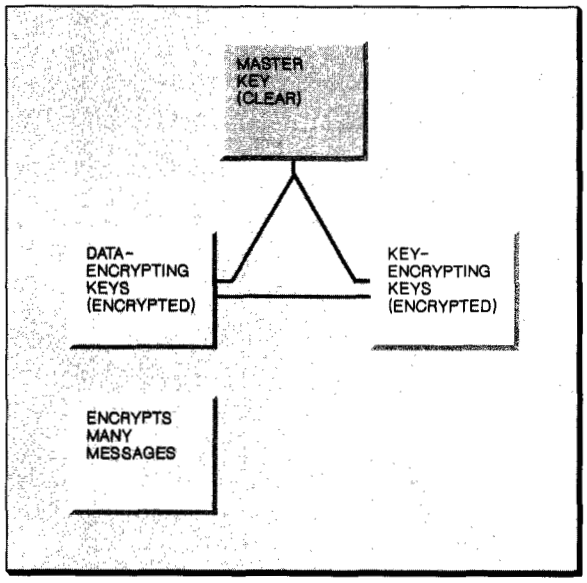
This paper presents a cryptographic key-management scheme based on control vectors. This is a new concept that permits cryptographic keys belonging to a cryptographic system to be easily, securely, and efficiently controlled. The new key-management scheme—built on the cryptographic architecture and key management implemented in a prior set of IBM cryptographic products—has been implemented in the newly announced IBM Transaction Security System.

In 1977, the National Bureau of Standards adopted an encryption algorithm, termed the *Data Encryption Standard* (DES),[1] as a federal standard. Following this milestone in cryptographic research and development, other advances in cryptography occurred in rapid succession, and many DES-based hardware and software products emerged to support new encryption-based protocols and cryptographic applications. In 1980, the American National Standards Institute (ANSI) adopted the same algorithm as a national standard, calling it the *Data Encryption Algorithm* (DEA).[2] 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. Each 64-bit key consists of 56 independent key bits and eight bits that may be used for error detection. Because the DEA is a published algorithm, data encrypted with the DEA are protected by keeping the key secret. In all, there are 72 057 594 037 927 936, that is, 2^{56} different cryptographic keys that may be used with the algorithm.

Key-management concepts. For two cryptographic devices to communicate using cryptography, each device must implement the same cryptographic algorithm and each device must be initialized with the same secret key. Data encrypted at a sending device are transmitted via a communications network to a receiving device, where they are decrypted. Access to data is controlled by the key, where possession or use of the key implies the right to decrypt and receive the clear data.

In a large network, data are transmitted among devices on behalf of many application programs and users. To protect these communications and to keep application programs and users from interfering with or reading the encrypted messages of any other, messages transmitted from one communicator to another are protected with a unique key shared by the communicators. Many keys are needed to facilitate this kind of end-to-end encipherment. Thus, cryptographic system services

Figure 1 A cryptographic key hierarchy



are needed to securely generate, distribute, and initialize these keys within the cryptographic system.

Most cryptographic systems make use of many different types of keys, so that information encrypted with a key of one type is not affected by using a key of another type. A key is assigned a *type* on the basis of the information the key encrypts or the use being made of the key. For example, a data-encrypting key encrypts data. A key-encrypting key encrypts keys. A PIN-encrypting key encrypts personal identification numbers (PINs) used in electronic funds transfer and point-of-sale applications. A MAC key is used to generate and authenticate message authentication codes (MACs).

The use of encryption is based on a strategy of protecting a large amount of information (a data file or communications session) with a smaller additional amount of information (a single key). Sophisticated key hierarchies have been devised using this principle. The hierarchy discussed in this paper is shown schematically in Figure 1. For example, the keys belonging to a cryptographic device are encrypted with a single master key and stored in a key data set. The master key is stored in clear form within the cryptographic hardware.

The concept of using a single master key to encrypt keys stored in a key data set is known as the *master key concept*. In order to electronically distribute keys from one device to another, e.g., to distribute a data-encrypting key as part of session initiation, each pair of devices shares a unique key-encrypting key under which all distributed keys are encrypted. Thus, a data-encrypting key encrypts many messages. A key-encrypting key encrypts many electronically distributed data-encrypting keys. A master key encrypts many key-encrypting and data-encrypting keys stored in a single key data set.

In order for a cryptographic system to be made operable, each device must first be initialized with a master key and at least one key-encrypting key. The master key permits keys stored in the key data set to be encrypted, and the key-encrypting key establishes a key-distribution channel with at least one other network device. When key distribution is performed in a peer-to-peer environment, each device is initialized with a key-encrypting key for each other device with which it wishes to communicate. However, when key distribution is performed with the assistance of a key-distribution center (KDC) or key-translation center (KTC), each device is initialized with only one key-encrypting key shared with the KDC or KTC. Thereafter, additional key-encrypting keys are distributed electronically and initialized automatically using the KDC or KTC. The key-distribution channel can also be made unidirectional. That is, one key-encrypting key encrypts keys transmitted from a first device to a second device and another key-encrypting key encrypts keys transmitted in the other direction.

Typically, the master key is generated and installed using manual entry techniques. Key-encrypting keys are generated as needed at designated generating devices and transported to designated receiving devices where they are installed. Although key-encrypting keys may be distributed by courier, the vast majority of all key distribution—generation, transmission, and reception of keys—is performed using automated, electronic methods.

An important feature of the cryptographic system is the method by which key separation is achieved. Key separation guarantees that keys of one type cannot be substituted and used as keys of another type. If, for example, a key-manage-

ment architecture defines two types of data keys, one for encipherment and another for decipherment, it must not be possible for a data-encipherment key to be substituted and used as a data-decipherment key. In the key-management scheme discussed in this paper, key separation and key-usage control are provided by a *control vector*.

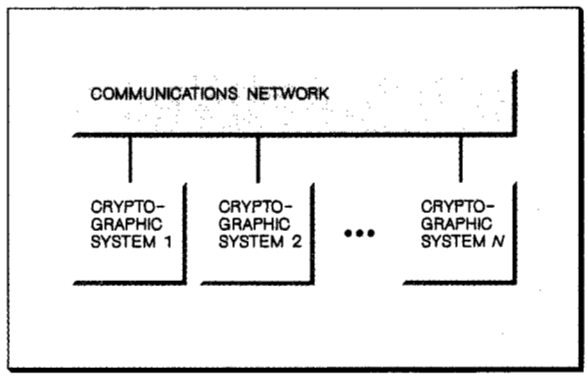
Key management is concerned with the generation, distribution, installation, storage, maintenance, and destruction of keys. This task includes methods for initializing keys, changing keys, re-enciphering keys kept in a system key data set (e.g., re-encipherment from a current to a new master key), and purging keys. The key generation, distribution, and installation processes may involve either persons as couriers or automated electronic procedures. The process includes techniques for the manual entry of keys by one or more persons or by electronic key distribution, using automated key servers. In summary, key management encompasses every aspect of the handling of keys, from the time a key is created until it ceases to exist.

Cryptographic architectural model. In the late 1970s, IBM introduced a line of cryptographic products based on the DES. The cryptographic architecture and key-management scheme are outlined in References 3 to 5 and are discussed in greater detail in References 6 and 7. In the discussion that follows, we refer to this key-management scheme, as IBM-1.

The cryptographic architecture implemented in this earlier line of cryptographic products defines a cryptographic network consisting of multiple cryptographic systems interconnected by a communications network, as illustrated in Figure 2. Each cryptographic system consists of a cryptographic facility (CF), a cryptographic key data set (CKDS), a cryptographic facility access program (CFAP), and using application programs (APPL), as illustrated in Figure 3.

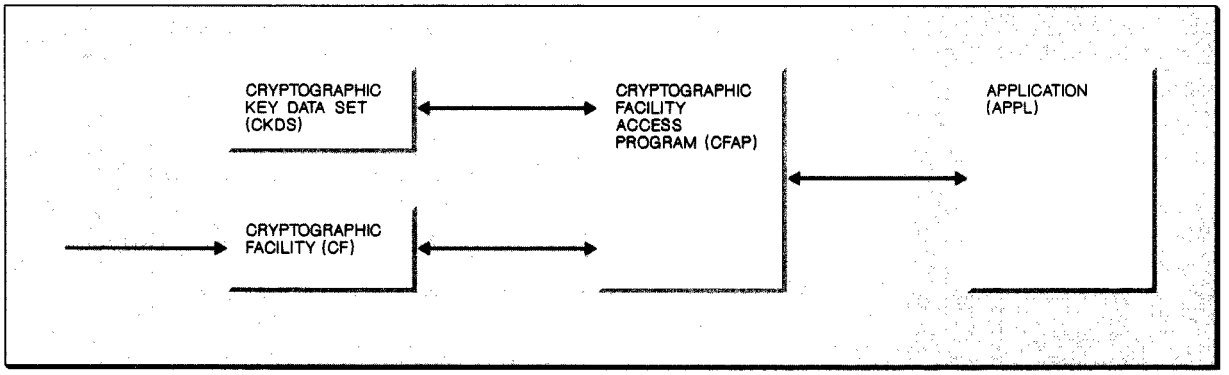
The CF is the hardware component of the cryptographic system and contains storage for a clear master key. All other keys belonging to the cryptographic system are encrypted under the master key and are stored in the CKDS. The CFAP is the software component of the cryptographic system. It interfaces with the APPL through an application programming interface (API) and with the CKDS and CF through architected system-level interfaces. In broad terms, the CFAP implements a set of cryptographic functions, denoted F1, F2, ..., F*m*, that may be invoked by application programs at the API, and the CF implements a set of cryptographic instructions, denoted I1, I2, ..., I*n*, that may be invoked by the CFAP through a CF-level programming interface. Except as noted, the elements of the cryptographic system described here are the same as those defined in Reference 6. In that reference, the CFAP is called a *key manager*, the CFAP cryptographic functions are called *programming macro instructions*, and the CF instructions are called *cryptographic operations*. Otherwise, the cryptographic architectural models are the same.

Figure 2 Cryptographic network



A typical request for cryptographic service is initiated by an APPL via a function call to the CFAP at the API. The service request includes key and data parameters. Also included are key identifiers that the CFAP uses to access encrypted keys from the CKDS. The CFAP processes the service request by issuing one or more cryptographic instructions to the CF at the CF-level interface. (The CF may also have an optional physical interface for direct entry of cryptographic variables into the CF, as illustrated in Figure 3 by a right-directed arrow toward the CF.) Each cryptographic instruction invoked at the CF-level interface has a set of input parameters processed by the CF to produce a set of output parameters returned by the CF to the CFAP. These outputs are processed by the CFAP in several ways. The CFAP may return output parameters to the APPL or it may use the output parameters as inputs to subsequently invoked in-

Figure 3 Cryptographic system



structions. Encrypted-key outputs may be stored in the CKDS.

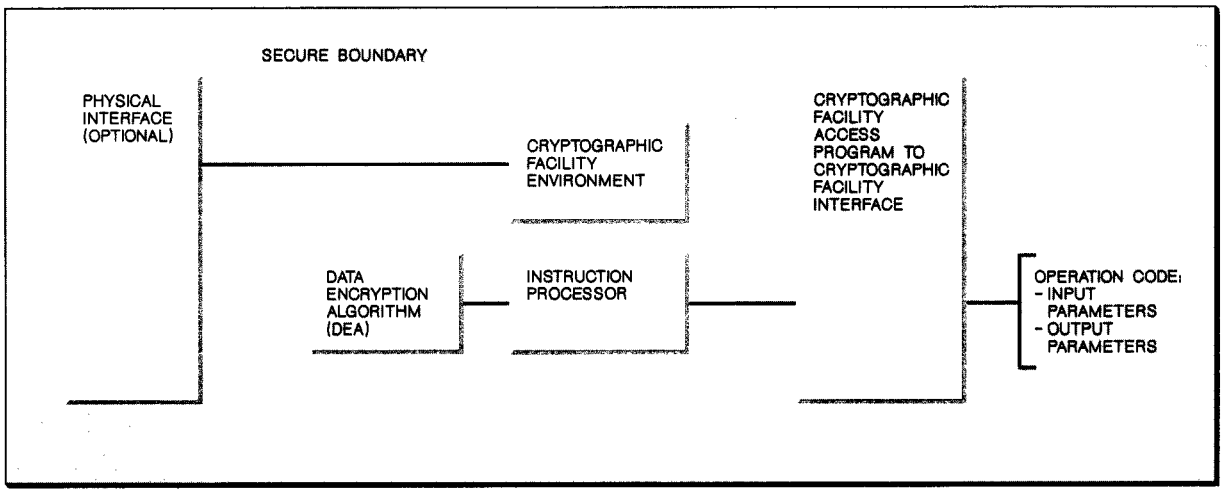
The elements composing the cryptographic system of Figure 3 permit the cryptographic system to be implemented in two parts: (1) a first part implemented within the cryptographic hardware, or CF, in order to meet cryptographic security and performance objectives, and (2) a second part containing everything that does not need to be implemented within the CF but which can be implemented safely and efficiently in the cryptographic software (i.e., the CFAP). By implementing the cryptographic system in two parts, a CF part and a CFAP part, the hardware component can be minimized, thus in many instances leading to a more cost-effective implementation.

The CF—the heart of the cryptographic system—contains an instruction processor, a Data Encryption Algorithm (DEA), and a CF environment, as illustrated in Figure 4. The *instruction processor* is a functional element that decodes and executes cryptographic instructions invoked by the CFAP at the CF-level interface. For each instruction, the CF-level interface defines (1) an operation code used to select a particular instruction for execution, (2) a set of input parameters passed from the CFAP to the CF, and (3) a set of output parameters returned by the CF to the CFAP. The instruction processor executes the selected instruction by performing an instruction-specific sequence of cryptographic processing steps whose control flow and subsequent output depend on the values of the input parameters and the contents of the CF environment. The CF environment consists of a set of cryptographic variables (e.g., keys, flags, counters, and CF configuration data) that collectively initialize and configure the CF. Chief among these is the 128-bit master key under which all keys in the CKDS are encrypted. The CF environment variables are initialized by the CF-level interface (i.e., by execution of certain CF instructions that read input parameters and load them into the CF environment). Otherwise, the variables are initialized by an optional physical interface that permits cryptographic variables to be loaded directly into the CF environment (e.g., via an attached key-entry device).

The cryptographic facility is implemented within a secure boundary that ensures that the CF is accessed only through architected interfaces. These interfaces are secure against intrusion, circumvention, and deception. This strategy ensures that clear keys and results of intermediate steps of encipherment and decipherment are kept secret. The physical embodiment of the CF is protected through the use of (1) *tamper-resistant designs* that resist physical probing and intrusion, (2) *tamper-detection circuitry* that detects attempted physical intrusion, and (3) *automatic zeroization* of keys if an attempted intrusion is detected.

The cryptographic instruction set must also be secure against an insider adversary with access to the CF-level interface. An adversary must not be able to recover keys in the clear outside the CF or subvert the intended security of the cryptographic system by attacks that make use of repeated executions of the CF instructions in any order, using intercepted or calculated informa-

Figure 4 Cryptographic facility



tion. That is, cryptographic security must not depend on denying an adversary access to the CF instruction interface merely because such denial could be effected by access control software, such as the IBM Resource Access Control Facility. The role of such access control measures in this model is more properly one of protecting access to and use of the system-managed encrypted keys stored in the CKDS. Access control measures are used to control which application programs have rights to use which encrypted keys in the CKDS. The cryptographic model also permits application programs to take possession of their own encrypted keys, in which case, possession of the key represents the right to use the key.

We now discuss the earlier cryptographic instruction set and key-management scheme implemented in IBM-1.

IBM-1 key-management review. The CF instruction set implemented in IBM-1[3–6] makes use of the following six cryptographic instructions:

- Encipher Data (ECPH)
- Decipher Data (DCPH)
- Set Master Key (SMK)
- Encipher Under Master Key (EMK)
- Re-Encipher From Master Key (RFMK)
- Re-Encipher To Master Key (RTMK)

Of these, the EMK, RFMK, and RTMK instructions are used for electronic key management.

The key-management scheme makes use of a 64-bit master key KM0 stored in clear form in the CF and two 64-bit variant master keys KM1 and KM2 derived from KM0. KM1 and KM2 are produced within the CF by an exclusive-OR operation on two 64-bit mask values v1 and v2 with KM0, respectively. That is, KM1 = KM0 $\oplus$ v1 and KM2 = KM0 $\oplus$ v2, where $\oplus$ denotes the exclusive-OR operation, and v1 and v2 are 64-bit universal constants defined by the key-management architecture.

To illustrate electronic key distribution, let K denote a 64-bit data key generated at device i and electronically distributed to device j. Let KM0i and KM0j denote the 64-bit master keys installed at devices i and j, respectively. Let KKij denote a 64-bit key-encrypting key installed at devices i and j, where KKij is used by device i to encrypt keys electronically transmitted to device j.

The method for establishing a common data key K between two devices i and j is to generate first a pseudorandom number RN at device i, where RN is defined as follows:

$$RN = e_{KM0i}(K)$$

That is, RN is defined as the encryption e of some key value K under the master key of device i. It can then be used directly in the ECPH and DCPH

instructions at device i to encipher and decipher data, as follows:

ECPH: (RN, data) $\rightarrow e_K$(data)

DCPH: (RN, e_K(data)) $\rightarrow$ data

The ECPH and DCPH instructions assume that an encrypted value of K of the form e_{KM0i}(K) is specified as an input.

RN can also be used with the RFMK instruction to transform K under the encipherment of key-encrypting key KKij belonging to device j. To send K to device j, the RFMK instruction is used at device i to produce e_{KKij}(K) by exercising

RFMK: (e_{KM1i}(KKij), e_{KM0i}(K)) $\rightarrow e_{KKij}$(K)

where KKij is stored encrypted under the first variant of KM0i, denoted KM1i.

The use of key variants in achieving key separation and key-usage control is the same for all of key management.

The quantity e_{KKij}(K) is then transmitted to device j, where the RTMK instruction is used to recover e_{KM0j}(K) by exercising

RTMK: (e_{KM2j}(KKij), e_{KKij}(K)) $\rightarrow e_{KM0j}$(K)

where KKij is stored encrypted under the second variant of KM0j, denoted KM2j.

The quantity e_{KM0j}(K) can then be used directly in the ECPH or DCPH instructions at device j, or it can be used with an RFMK instruction to transform K under the encipherment of a key-encrypting key belonging to another device.

From the description of the RFMK and RTMK instructions, one can see that at device i KKij is encrypted using KM1i, i.e., the first variant of KM0i. At device j, KKij is encrypted using KM2j, i.e., the second variant of KM0j. By encrypting KKij in this manner at devices i and j, KKij is enabled for use with the RFMK instruction at device i and the RTMK instruction at device j. In effect, KKij is encrypted in a way that establishes a unidirectional or one-way key-distribution channel from device i to device j, thus permitting keys to be electronically distributed from device i to device j. To send keys from device j to device i requires a different KKji to be installed at both devices, i.e., KKji is installed at device j as KKij is installed at device i, and KKji is installed at device i as KKij is installed at device j.

The key-management scheme also provides for the use of a 128-bit master key KM0 and 128-bit key-encrypting keys KKij. In that case, the master key variants KM1 and KM2 are produced from KM0 by an exclusive-OR operation of nonsecret mask values v1 and v2 with the leftmost and rightmost 64-bit parts of KM0, i.e., KM1 = KM0 $\oplus$ (v1,v1) and KM2 = KM0 $\oplus$ (v2,v2), where the symbol $\oplus$ denotes the exclusive-OR operation and a comma (,) denotes concatenation. When 128-bit keys are incorporated into the key-management scheme, the encrypted key e_{KM1i}(KKij) depicted in the RFMK instruction is replaced by e^*_{KM1i}(KKLij), e^*_{KM1i}(KKRij), where KKLij and KKRij are the leftmost and rightmost 64-bit parts of KKij, and e* denotes encryption with a 128-bit key. Likewise, the encrypted key e_{KM2j}(KKij) depicted in the RTMK instruction is replaced by e^*_{KM2j}(KKLij), e^*_{KM2j}(KKRij). That is, the RFMK and RTMK instructions are redefined to accommodate 128-bit key-encrypting keys.

The CF instruction set implemented in IBM-1 supports other key-distribution and key-management services for both communication security and file security. However, the use of key variants in achieving key separation and key-usage control, as illustrated in the example of electronic key distribution, is basically the same for all of key management. In contrast, the underlying cryptographic architecture and key-management scheme implemented in the IBM Transaction Security System, based on control vectors, provides many new and improved features and services. In the remainder of the paper, we discuss a cryptographic system design and key-management scheme implemented in the Transaction Security System. In the discussion that follows, we refer to this key-management system as IBM-2. The Transaction Security System is discussed in a companion paper in this issue.[8] Key handling with control vectors is discussed in another companion paper in this issue.[9]

Transaction Security System cryptographic system design objectives

The Transaction Security System cryptographic system design is based on the same cryptographic architectural model and security rules implemented in existing IBM cryptographic products just outlined. It also embraces several high-level strategic goals. It should be a general-purpose system that is applicable to a wide range of computing devices and should serve customer needs through the 1990s and beyond. The system should be open-ended, thereby allowing growth and extension to keep pace with new cryptographic methods, services, and standards required by users. The Transaction Security System should provide a stable base that permits users to plan and develop long-range cryptographic security strategies and to design their own cryptographic applications and high-level cryptographic security architectures. The system should provide product independence and product interoperability, as long as a product adheres to the architecture. It should support appropriate ANSI and ISO cryptographic standards, in whole or in part, so as to free users from total dependence on the cryptographic methods of a particular vendor. Compatibility with present IBM and non-IBM products is important as is the provision of an application programming interface (API) tailored for ease of use. Such a system should provide strong cryptographic protection consistent with commercial computer and networking environments while meeting industry requirements for performance.

The system should provide product independence and product interoperability.

Transaction Security System cryptographic facility instruction set

The cryptographic facility (CF) instruction set represents that part of the cryptographic system that must be implemented within the cryptographic hardware in order to achieve required security and performance objectives. However, other important objectives are achieved by the CF instruction set.

The instruction set is comprehensive, in that it provides a wide range of cryptographic services. The instruction set can be divided into nine functional categories, according to the cryptographic services provided. The nine functional categories are: (1) data management, (2) personal identification number (PIN) management, (3) electronic key management, (4) compatibility mode electronic key management, (5) ANSI X9.17 electronic key management, (6) CF initialization, (7) CF control, (8) CF configuration, and (9) utility. Electronic key management and compatibility mode electronic key management are the primary topics discussed in this paper.

The instruction set is minimal in that each instruction provides a unique and necessary function. Redundancy in the instruction set is reduced or eliminated. The instruction set is consistent, making it possible for one to infer how one instruction works from a knowledge of the workings of another. This helps in understanding how the instructions operate. The instructions that process data are streamlined. The design seeks to minimize the key-management overhead required to process keys in these instructions. This ensures that the most frequently used cryptographic instructions are the most efficient. The CF also provides a capability for CF instructions to be dynamically enabled and disabled. Enabling an instruction for execution can also be made contingent on proof of authorization via a password, personal identification number (PIN), signature verification, or some other such way. The IBM Transaction Security System cryptographic hardware and authorization mechanisms are discussed in Reference 8.

Transaction Security System electronic key management

The Transaction Security System cryptographic system design includes the following CF instructions that support electronic key management:

- Generate Key Set (GKS)
- Generate Key Set Extended (GKS-E)
- Re-Encipher To Master Key (RTMK)
- Translate Key (XLTKEY)

- Re-encipher From Master Key (RFMK)
- Replicate Key (REPK)
- Lower Export Authority (LEA)

Important differences exist between the first IBM key management (IBM-1) and the key-management scheme implemented in the Transaction Security System (IBM-2). We now discuss these differences.

Both IBM key-management schemes permit 64-bit key-encrypting keys to be stored within the cryptographic system as 128-bit keys.

Key encryption/decryption. Basically, IBM-1 is based on key variants and IBM-2 is based on control vectors. In IBM-1, data keys are encrypted under the master key KM0, and they are application-program-managed keys. Key-encrypting keys are encrypted under the first and second variants of the master key, and they are CFAP-managed keys stored in the CKDS. Data keys transmitted in the key-distribution channel are encrypted under a key-encrypting key (i.e., no key variants are used). In IBM-2, keys are encrypted and decrypted with algorithms that make use of control vectors. The control vector encryption (CVE) and control vector decryption (CVD) algorithms are described in Reference 9.

Key-distribution channel. In IBM-1, only data keys are transmitted in the key-distribution channel. Hence, there is no need to define variants of a key-encrypting key in order to maintain cryptographic separation among key types in the key-distribution channel. However, in IBM-2, many key types are distributed from one device to another, and hence the CVE and CVD algorithms are used to cipher keys transmitted in the key-distribution channel. The exception is the data compatibility key, which is transmitted in the key-distribution channel using a vector of all zeros (i.e., no control vector).

Encryption of 128-bit keys. In IBM-1, the leftmost and rightmost 64-bit parts of a 128-bit key-encrypting key are each encrypted with the same master key variant. In IBM-2, a *form field* in the control vector indicates whether the encrypted 64-bit key is a leftmost or rightmost 64-bit part of a 128-bit key. Otherwise, the two control vectors are the same. Thus, if C1 and C2 are the control vectors used to encrypt the leftmost and rightmost 64-bit parts of a 128-bit key, then C1 and C2 differ only in the encoded values stored in the form fields in C1 and C2. This feature of IBM-2 prevents the leftmost 64-bit part of a 128-bit key from being substituted and used for a rightmost 64-bit part of a 128-bit key, and vice versa.

Compatibility support for 64-bit key-encrypting keys. Both IBM-1 and IBM-2 permit 64-bit key-encrypting keys to be stored within the cryptographic system as 128-bit keys. However, this compatibility mode feature is implemented differently in each key-management methodology. Because in IBM-1 the leftmost and rightmost 64-bit parts of a 128-bit key are interchangeable, 64-bit key-encrypting keys are supported by creating and encrypting the leftmost 64-bit part of a 128-bit key to produce a value of the form $e^*_{KM1}(KKL)$ and then defining $e^*_{KM1}(KKR)$ to be equal to $e^*_{KM1}(KKL)$. However, in IBM-2, where the leftmost and rightmost 64-bit parts of a 128-bit key are encrypted with different control vectors C1 and C2, a Replicate Key (REPK) instruction is provided that transforms an encrypted key of the form $e^*_{KM\oplus C1}(KKL)$ to the form $e^*_{KM\oplus C2}(KKR)$, where KKR equals KKL. C1 designates KKL as the leftmost 64-bit part of KK, and C2 designates KKR as the rightmost 64-bit part of KK.

Key distribution via the GKS instruction. In IBM-1, key distribution is effected through the use of the RFMK and RTMK instructions. A key to be distributed is first produced in the encrypted form $e_{KM0i}(K)$. The quantity $e_{KM0i}(K)$ is then re-enciphered to the encrypted form $e_{KKij}(K)$ using the RFMK instruction. At the receiving device, the received quantity $e_{KKij}(K)$ is re-enciphered to the encrypted form $e_{KM0j}(K)$ using the RTMK instruction. The disadvantage in using the RFMK instruction for key distribution is that all distributed keys are first produced in the form $e_{KM0i}(K)$. Thus, a key intended for use at a receiving device is also available in usable form at the sending device, thereby opening up the possibility that the sending device may misuse the receiving device's key.

However, in IBM-2, this threat is eliminated through the use of a GKS instruction, which encrypts the generated key directly under a key-encrypting sender key KESK of the receiving device. Thus, the key is not exposed in an operable form at the sending device, and the GKS and RTMK instructions become the preferred method for generating and distributing keys. The GKS instruction generates two encrypted copies of a key, where the control vectors for each of the encrypted copies may be equal or different. An instruction mode parameter permits the so-produced encrypted keys to be encrypted with a master key KM, a key-encrypting sender key KESK, or a key-encrypting receiver key KERK.

A GKS-E (extended mode) instruction permits special combinations of keys to be generated that are usually prohibited.

Key-translation capability. In IBM-1, key translation could be performed only by first importing the key with the RTMK instruction and then exporting it with the RFMK instruction. However, the disadvantage of this method is that the device performing the key-translation operation has, by definition, an imported copy of the key in a form usable at that device. This might permit intercepted encrypted communications to be decrypted by the device, although no explicit right to do so may have been granted by the sending device. In IBM-2, a Translate Key (XLTKEY) instruction permits keys to be translated within the CF from encryption under a first key-encrypting key to encryption under a second key-encrypting key. During the key-translate process, the control vectors associated with the first and second key-encrypting keys are checked for equality, thus ensuring that key usage is propagated.

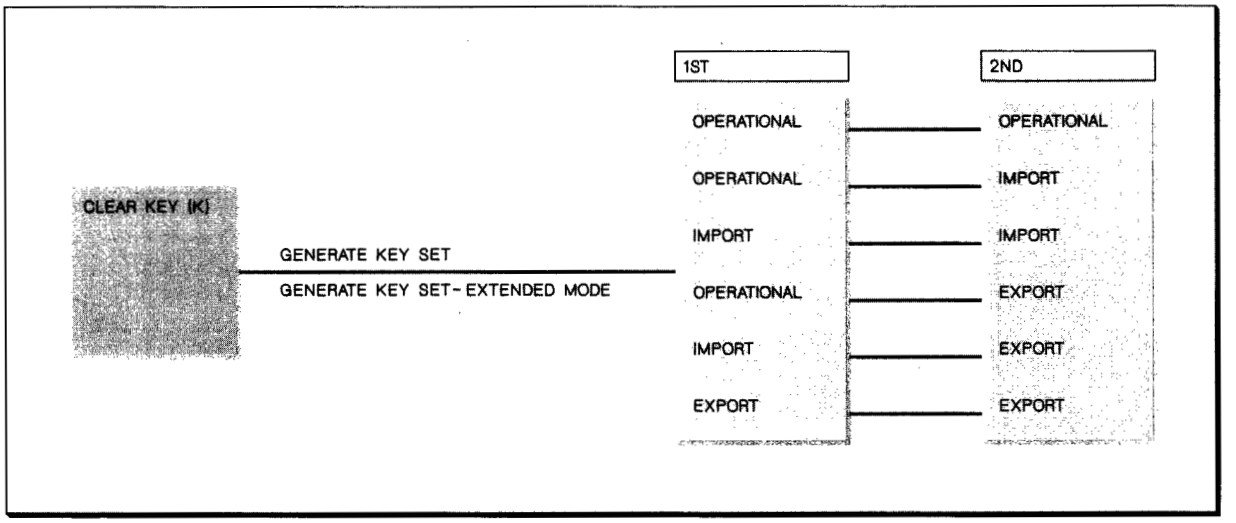
Export control via the RFMK instruction. In IBM-1, any data key of the form $e_{KM0}(K)$ can be re-enciphered to the form $e_{KK}(K)$, using the RFMK instruction. In IBM-2, an export control bit in the control vector is interrogated by the RFMK instruction to determine whether the key may or may not be exported (B'1' indicates export allowed and B'0' indicates export not allowed.) A Lower Export Authority (LEA) instruction permits the export control bit to be reset from export allowed to export not allowed, but not vice versa.

Now that the important new features of the Transaction Security System key-management scheme (IBM-2) have been highlighted, the topics of key generation and key distribution in IBM-2 can be discussed.

Key generation using the Generate Key Set instruction. The Generate Key Set (GKS) instruction generates two encrypted copies of a key, where the control vectors associated with each of the encrypted copies of the key may be equal or different. Each encrypted copy of the key may be encrypted with the master key (called an operational key or OP-key), with a key-encrypting sender key KESK (called an EX-key for export key), or with a key-encrypting receiver key KERK (called an IM-key for import key). The terms OP-key, EX-key, and IM-key may be abbreviated as OP, EX, and IM, respectively. An OP-key is a key in an encrypted state that can be processed directly by the cryptographic facility (CF). An EX-key is a key in an encrypted state that can be exported to another device, where it is imported. An IM-key is a key in an encrypted state that can be imported. The terms OP-key, EX-key, and IM-key are often convenient in explaining and understanding how keys are processed by the key management. An OP-key is exported by translating it to an EX-key. An IM-key is imported by translating it to an OP-key. The Translate Key instruction translates an inbound IM-key to an outbound EX-key. A key-distribution channel is established from device i to device j by installing a KESK at device i and installing a matching KERK at device j. Thus, an EX-key produced at device i automatically becomes an IM-key at device j.

The GKS instruction generates two encrypted copies of a 64-bit odd parity adjusted random key K, where K may be a 64-bit key, the leftmost 64 bits of a 128-bit key, or the rightmost 64 bits of a 128-bit key. A mode parameter specified to the instruction permits the encrypted copies of the key to be produced in the following pairwise states: OP-OP, OP-IM, IM-IM, OP-EX, IM-EX, and EX-EX, as illustrated in Figure 5. A pair of control vectors (C1,C2) is also provided as an input to the GKS instruction, where C1 specifies the attributes granted to the first encrypted copy of K, and C2 specifies the attributes granted to the second encrypted copy of K. For example, GKS mode = 1 produces an OP-key and EX-key pair, where the OP-key can be a *data-privacy key*, with an encipher attribute, and the EX-key can be a *data-privacy-translate key*, with a *translate-in* attribute. Definitions for the data-privacy and data-privacy-

Figure 5 Encrypted states produced by the GKS and GKS-E instructions from clear key K



translate keys are given in a companion paper in this issue.[9] The translate-in attribute permits the key to decipher data in the CF only for the purpose of immediately re-enciphering it under a different data-privacy-translate key, with a *translate-out* attribute. A full discussion of the control vector pairs (C1,C2) that may be specified to the GKS instruction is not possible in this short paper.

The GKS-E instruction is functionally the same as the GKS instruction, except that it accepts a different set of input control vector pairs (C1,C2).

The cryptographic facility access program (CFAP) uses the GKS and GKS-E instructions to generate 64- and 128-bit keys. A 128-bit key is produced by executing the GKS instruction twice and storing both encrypted key outputs in a single key token. In situations where, for reasons of compatibility, it is necessary to generate a 128-bit key with the leftmost 64 bits equal to the rightmost 64 bits, a Replicate Key (REPK) instruction can be used to generate an encrypted output representing the rightmost 64 bits of a key from an encrypted input representing the leftmost 64 bits of a key. In that case, the GKS or GKS-E instruction is used to generate the leftmost 64 bits of the key and the REPK instruction is used to generate the rightmost 64 bits of the key from the leftmost 64 bits of the key.

Key-distribution environments. The key-management scheme in IBM-2 is purposely designed to support different key-distribution environments. Of course, these key-distribution features implemented within the CF require supporting application program services, e.g., a key server program interfacing to the CFAP with a capability to dynamically generate and serve keys to network cryptographic devices and application programs. The key-management architecture supports end-to-end encryption in a (1) *peer-to-peer* environment, (2) *key-distribution-center* environment, and (3) *key-translation-center* environment.

Peer to peer. A peer-to-peer environment is set up as follows: A serves keys to B, as illustrated in Figure 6, or B serves keys to A. A key-distribution channel is first established from A to B, using nonelectronic methods, by installing a key-encrypting sender key, KESK1, at A and matching key-encrypting receiver key, KERK1, at B. Then A generates a matching pair, KERK2, KESK2, as OP-key and EX-key, respectively, keeping OP-key KERK2 and serving EX-key KESK2 to B. A GKS instruction operating in the OP-EX mode is used at A to generate keys, as required by the protocol. At B the EX-key KESK2 is imported by translating it to OP-key KESK2, thus establishing a key-distribution channel from B to A. An RTMK instruction is used at B to import keys, as required by the protocol.

Key-distribution center. In a key-distribution-center environment, a key-distribution center C

generates keys for A and B, as illustrated in Figure 7. For example, A serves a key to B by making a request for keys from C. In response, C generates a pair of keys and returns them to A. Whereupon, A keeps one of the keys and serves the other to B. If C is permitted by the key-distribution protocol to generate key-encrypting keys for A and B, then C can be used to establish key-distribution channels from A to B and from B to A. Thereafter, A and B can use a peer-to-peer key-distribution protocol, provided they have a key-generation capability.

To establish a key-distribution-center environment, key-distribution channels are established from C to A and from C to B using nonelectronic methods. That is, a KESK1 is installed at C and matching KERK1 is installed at A. Likewise, a KESK2 is installed at C and matching KERK2 is installed at B. A GKS instruction operating in the EX-EX mode, is used at C to generate keys for A and B, as required by the protocol. The EX-EX mode generates a pair of keys as EX- and EX-keys.

Alternatively, key-distribution channels are first established from A to C and B to C using nonelectronic methods. Then A generates a matching pair (KERK1, KESK1), as OP- and EX-key, respectively, keeping OP-key KERK1 and serving EX-key KESK1 to C. At C, the EX-key KESK1 is imported by translating it to OP-key KESK1, thus establishing a key-distribution channel from C to A. In like manner, B generates a matching pair (KERK2, KESK2) and follows the same procedure to establish a key-distribution channel from C to B. The advantage here is that the attributes in the control vectors associated with KESK1 and KESK2, which prescribe the usage of KESK1 and KESK2 at C, are entirely under the control of A and B, respectively. Thus, A and B can grant rights to C to use KESK1 and KESK2 only for generating key pairs for A and B, as required by the protocol. Keys generated at C on behalf of A and B, i.e., when C acts as a key-distribution center, could not also exist as OP-keys at C. This, for example, prevents C from eavesdropping on encrypted sessions between A and B.

Key- translation center. In a key-translation-center environment, A cannot serve keys to B directly. Instead, A generates a pair of keys, keeps one and sends the other to a key-translation center T, as illustrated in Figure 8. T translates the key into a form that can be imported at B and returns it to A. A then serves the translated key to B.

Figure 6 Peer to peer

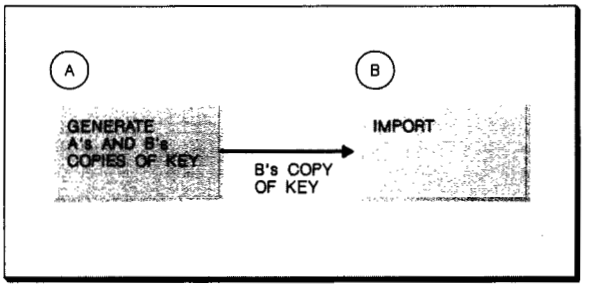
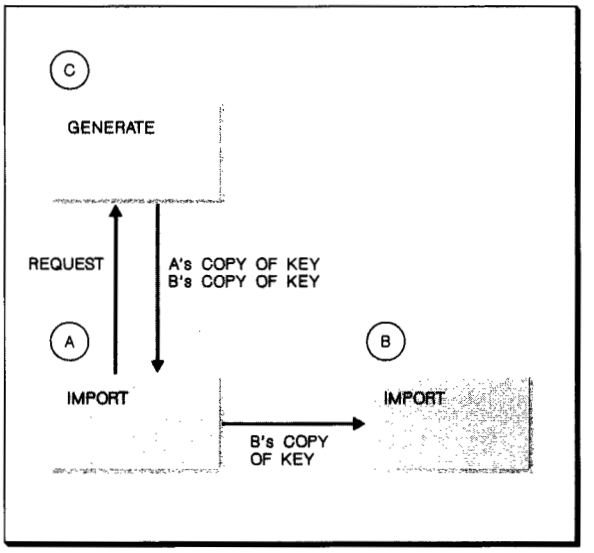


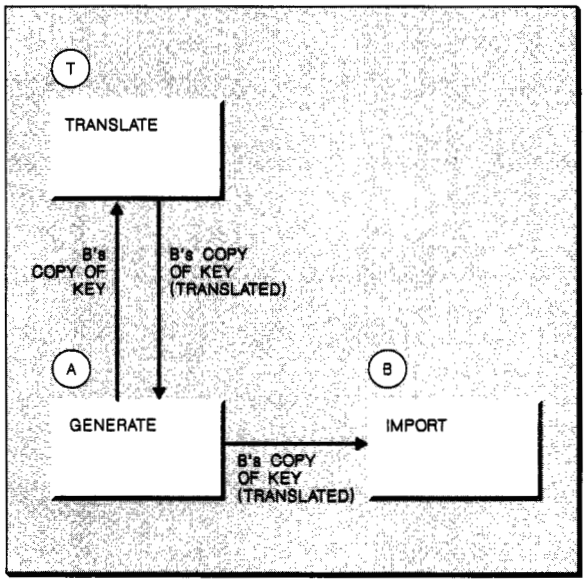
Figure 7 Key-distribution center



To set up a key-translation-center environment is a bit more complicated. Therefore, the example shows only how A, B, and T cooperate to set up a key-translation channel from A to B. One can see that it is fairly easy to extend the protocol to handle keys in the opposite direction.

To establish a key-translation-center environment, key-distribution channels are established from A to T and from T to B using nonelectronic methods. That is, a KESK1 is installed at A and a matching KERK1 is installed at T. Likewise, a KESK2 is installed at T and matching KERK2 is

Figure 8 Key-translation center



installed at B. A Translate Key (XLTKEY) instruction is used at T to translate received IM-keys to EX-keys (i.e., to re-encrypt keys from encryption under KERK1 to encryption under KESK2). This permits T on behalf of A to translate keys so that A can serve them to B, as required by the protocol.

Alternatively, key-distribution channels are first established from A to T and B to T using non-electronic methods. Then A generates a matching pair (KESK1, KERK1), as OP-key and EX-key, respectively, keeping OP-key KESK1 and serving EX-key KERK1 to T. At T the EX-key KERK1 is imported by translating it to OP-key KERK1, thus establishing a key-distribution channel from A to T. Similarly, B generates a matching pair (KERK2, KESK2), as OP- and EX-key, respectively, keeping OP-key KERK2 and serving EX-key KESK2 to T. At T, the EX-key KESK2 is imported by translating it to OP-key KESK2, thus establishing a key-distribution channel from T to B. The advantage here is that the attributes in the control vectors associated with KERK1 and KESK2, which prescribe the usage of KERK1 and KESK2 at T, are entirely under the control of A and B, respectively. Thus, A and B can grant rights to T to use KERK1 and KESK2 only for translating keys from encryption under KERK1 to encryption under KESK2, as required by the protocol. Keys translated at T on behalf of A and B, i.e., when T acts as a key-translation center, could not also exist as OP-keys at T. This imposes a level of integrity on the key-translation process, that is similar to that described above for the key-distribution center.

KESK and KERK control vectors. The control vector is the means whereby an installation implements its own key-management policy and rules for governing the generation, manipulation, and processing of keys. The granularity in the control vector permits an installation to set up its key-encrypting keys to support key distribution in different key-distribution environments (i.e., peer to peer, key-distribution center, and key-translation center). In effect, an installation selects the type of key distribution it wants and then implements that decision through selectable CFAP options that determine how the control vector is encoded.

The *usage control* fields in the key-encrypting sender key control vector and key-encrypting receiver key control vector contain subfields for controlling key usage. The usage subfields in the KESK control vector are shown in Table 1. The usage subfields in the KERK control vector are shown in Table 2. For convenience, usage attributes are listed using the following notation:

KESK⟨...attributes...⟩
KERK⟨...attributes...⟩

Here the presence of an attribute name means that it has a value of 1 and the absence of an attribute name means that it has a value of 0. Thus, (KESK⟨gks op-ex⟩, KERK⟨rtmk⟩) denotes a (KESK, KERK) pair, such that: (a) the control vector for KESK has the gks op-ex attribute set equal to 1 and the remainder of the attributes set equal to 0; and, (b) the control vector for KERK has the rtmk attribute set equal to 1 and the remainder of the attributes set equal to 0.

The minimal configuration to support peer-to-peer key distribution, e.g., between cryptosystems A and B, is this:

A : KESK1⟨gks op-ex⟩ B : KERK1⟨rtmk⟩
KERK2⟨rtmk⟩ KESK2⟨gks op-ex⟩

The pair (KESK1, KERK1) provides a key-distribution channel from A to B and the pair (KERK2, KESK2) provides a key-distribution channel from B to A. To serve a key, A executes the GKS in-

Table 1 The usage subfields in the KESK control vector

GKS IM-EX	::	1 : KESK can be used with GKS IM-EX to produce the EX-key.
		0 : cannot
GKS OP-EX	::	1 : KESK can be used with GKS OP-EX to produce the EX-key.
		0 : cannot
GKS EX-EX	::	1 : KESK can be used with GKS EX-EX to produce the first EX-key or second EX-key.
		0 : cannot
RFMK	::	1 : KESK can be used with RFMK to encrypt an intermediate clear key (CL key) to produce an output EX-key.
		0 : cannot
XLTKEY OUT	::	1 : KESK can be used with XLTKEY to encrypt an intermediate clear key (CL key) to produce an output EX-key.
		0 : cannot

Table 2 The usage subfields in the KERK control vector

GKS IM-EX	::	1 : KERK can be used with GKS IM-EX to produce the IM-key.
		0 : cannot
GKS OP-IM	::	1 : KERK can be used with GKS OP-IM to produce the IM-key.
		0 : cannot
GKS IM-IM	::	1 : KERK can be used with GKS IM-IM to produce the first IM-key or second IM-key.
		0 : cannot
RTMK	::	1 : KERK can be used with RTMK to decrypt an IM key to recover an intermediate clear key (CL key).
		0 : cannot
XLTKEY IN	::	1 : KERK can be used with XLTKEY to decrypt an input IM key to recover an intermediate clear key (CL key).
		0 : cannot

struction in OP-EX mode to generate an OP-key which it keeps and an EX-key which it sends to B. B executes an RTMK instruction to re-encrypt the received IM-key to an OP-key. B serves keys to A in a like manner.

The minimal configuration to support a key-distribution center, e.g., consisting of cryptosystems A and B and key-distribution center C, is this:

A : KERK1⟨rtmk⟩ C : KESK1⟨gks ex-ex⟩
B : KERK2⟨rtmk⟩ KESK2⟨gks ex-ex⟩

The pair (KERK1, KESK1) provides a key-distribution channel from C to A, and the pair (KERK2, KESK2) provides a key-distribution channel from C to B. In response to a request for keys, e.g., from A, C executes the GKS instruction in EX-EX mode to generate a first EX-key (encrypted under KESK1) and a second EX-key (encrypted under KESK2). Upon receipt of the two keys at A, which are IM-keys from the standpoint of both A and B, A keeps the IM-key encrypted under KERK1 and serves the other IM-key encrypted under KERK2 to B. Both A and B import their respective IM-keys by executing an RTMK instruction.

Since C is only granted rights to use KESK1 and KESK2 with mode EX-EX of the GKS instruction, there is no opportunity for C to violate the integrity of the key-distribution protocol. In effect, it is not possible for C to give A or B a key that C also has use of in its own cryptosystem (i.e., a key that C has stored as an OP-key). Of course, this is not the case if C is totally free to select the usage control attributes of KESK1 and KESK2. For example, suppose that C establishes a configuration that looks like this:

A : KERK1⟨rtmk⟩ C : KESK1⟨gks op-ex, rfmk⟩
B : KERK2⟨rtmk⟩ KESK2⟨gks op-ex, rfmk⟩

In that case, in response to a request for keys from A, C executes the GKS instruction in OP-EX mode to generate an OP-key and an EX-key (en-

crypted under KESK2). C then executes the RFMK instruction to translate the OP-key to an EX-key (encrypted under KESK1). C sends the two EX-keys to A, as before, but keeps the OP-key. Another variation on this theme is to generate an IM-key at C, use an RTMK instruction to import the IM-key, and use a Translate Key instruction also to translate the IM-key to an EX-key suitable to be given to A.

The minimal configuration to support a key-translation center, e.g., consisting of cryptosystems A and B and key-translation center T, is this:

A : KESK1⟨gks op-ex⟩	T : KERK1⟨xltkey in⟩
KERK2⟨rtmk⟩	KESK2⟨xltkey out⟩
B : KESK3⟨gks op-ex⟩	KERK3⟨xltkey in⟩
KERK4⟨rtmk⟩	KESK4⟨xltkey out⟩

The reader can trace the steps necessary to serve keys from A to B, and vice versa. IBM-2 also provides a compatibility mode for electronic key distribution. This topic is discussed next.

Compatibility

The Transaction Security System provides instructions for achieving levels of compatibility with other IBM and non-IBM systems not implementing control vectors. Chief among these is the Translate Control Vector (XLTCV) instruction, which provides a general mechanism for adding, deleting, and remapping control vector values. The XLTCV instruction permits compatibility with any DEA-based key-management scheme where keys are distributed in one of the following forms:

$e_{KK}(K)$

$e^*_{KK}(K)$

$e_{KK \oplus X}(K)$

$e^*_{KK \oplus Y}(K)$

where K is a 64-bit key, the leftmost 64 bits of a 128-bit key, or the rightmost 64 bits of a 128-bit key; KK is a 64-bit key (in the forms $e_{KK}(K)$ and $e_{KK \oplus X}(K)$) or a 128-bit key (in the forms $e^*_{KK}(K)$ and $e^*_{KK \oplus Y}(K)$); X is any 64-bit value; Y is any 128-bit value; and $\oplus$ denotes the exclusive-OR operation. Many existing IBM and non-IBM systems implement key distribution using one of the listed forms. (The most notable exception is ANSI X9.17, which uses a form of key notarization and key offsetting described in another paper[9] in this issue.)

To export a key to a device that does not implement control vectors, IBM-2 uses the XLTCV instruction to translate an encrypted key of the form $e^*_{KK \oplus C}(K)$ to one of the compatibility forms. That is, XLTCV remaps C to a 64-bit vector of zeros, a 128-bit vector of zeros, a 64-bit vector X, or a 128-bit vector Y. To import a key originating with a device not supporting control vectors, one uses the reverse process. In the expression $e^*_{KK \oplus C}(K)$, C is a 128-bit control vector,[10] KK is either a key-encrypting sender key (KESK) or a key-encrypting receiver key (KERK). Also, the XLTCV instruction has a parity adjust option, so that an input key K can be adjusted for odd parity on output.

The mapping process is controlled by a control vector translation table (CVTT) created in advance by authorized installation personnel and supplied to the XLTCV instruction as an instruction input. The CVTT entries are partitioned into the following two groups: (1) those associated with key-encrypting sender keys and (2) those associated with key-encrypting receiver keys. As part of the CVTT creation process, the CVTT is encrypted with a cryptovariable-encrypting key available only to authorized installation personnel. That is, the operation of the XLTCV instruction is indirectly controlled by the special authorization needed to create the CVTT. Once the CVTT has been created, the XLTCV instruction executes without any special authorization. This means that the XLTCV instruction can provide routine compatibility support for electronic key-distribution protocols involving devices that do not implement control vectors.

The XLTCV instruction provides great flexibility for achieving compatibility with devices that do not implement control vectors. The instruction is highly granular, thus allowing an installation to remap control vectors only of the types needed—and no more. However, the decision to use the XLTCV instruction is one that each installation can make on its own, according to its own requirement for compatibility with other devices not implementing control vectors.

Super keys. Compatibility among Transaction Security System devices and other IBM and non-IBM devices not implementing control vectors is handled in the Transaction Security System through

the use of *super keys*. A super key is a collection of encrypted keys of the form ($e^*_{KK\oplus C1}$(K), C1), ($e^*_{KK\oplus C2}$(K), C2), ..., ($e^*_{KK\oplus Cn}$(K), C*n*), where the sum of the permitted uses granted to K by control vectors C1 through C*n* is equal to (or perhaps greater than) the uses otherwise granted to K if it were installed, as usual, in a device not implementing control vectors. A super key is a key-management construct implemented within the CFAP, i.e., the CF is unaware of the super key. The super key has a single *key name* or *key label*. Thus, from the point of view of the application program, the super key is just a single key. In practice, the number of different control vectors needed to cover the permitted uses of a key originating with a device not supporting control vectors is small. As a consequence, storage requirements for super keys are not burdensome to the CFAP. An application program can use a super key in the same way it uses any other key, by making a service request to the CFAP and specifying a super key to be used in satisfying that request. The CFAP selects the appropriate ($e^*_{KK\oplus Ci}$(K), Ci) from the super key and executes the appropriate CF instruction, passing ($e^*_{KK\oplus Ci}$(K), Ci) as an input. The word "appropriate" in this case means that CFAP must select ($e^*_{KK\oplus Ci}$(K), Ci) so that Ci grants to K the needed rights to be used in the CF instruction that CFAP must execute in order to satisfy the service request made by the application program. Another service request involving the same super key may result in CFAP accessing ($e^*_{KK\oplus Cj}$(K), Cj) and executing a different CF instruction. Obviously, the intent here is to keep the CFAP-level complexity associated with the super key hidden from the using application program.

Installation of super keys. The same key-management instructions used to install Transaction Security System keys can be used to install a super key. The difference is that with a super key the process must be repeated for each super key element ($e^*_{KK\oplus Ci}$(K), Ci) to be created. Of course, one would use a method that does not require K to be manually re-entered into the CF multiple times. One acceptable method is to prepare super keys at an off-line device where K, KK, and (C1, C2, ..., C*n*) are made available to a utility program that produces the super key. If one installs KK at the Transaction Security System device as a key-encrypting receiver key KERK with usage attribute ⟨rtmk⟩, i.e., KERK⟨rtmk⟩, the CFAP can import the off-line prepared super key, element by element, using the RTMK instruction.

Table 3 Example control vector translate table for KERK and KESK

For KERK:		For KESK:	
Input	Output	Input	Output
X	C1	X	0
X	C2		

Electronic generation and export of super keys. Let A represent a Transaction Security System device and B a device not supporting control vectors. A key-distribution channel is first established from A to B, using nonelectronic methods, by installing KESK1 at A and installing matching KERK1 at B. We assume that B can import keys encrypted in the form e^*_{KERK1}(K), where K can have permitted uses at B comparable to a super key at A of the form (($e^*_{KM\oplus C1}$(K), C1), ($e^*_{KM\oplus C2}$(K), C2)). (After one sees how to handle a super key with two elements, the method can be easily extended to handle any number of elements.) We further assume that KESK1 is stored at A in the form ($e^*_{KM\oplus C3}$(KESK1), C3), possibly as part of a super key, where C3 has attribute ⟨xltkey out⟩. We further assume that, in advance, A generates a key-encrypting receiver key KERK2 with attributes ⟨xltkey in, rtmk⟩. No matching KESK2 is generated, as KERK2 is used only by A to send keys to itself. We further assume that a control vector translate table (CVTT) is prepared in advance to permit the CV mappings shown in Table 3 to be made.

To generate and export a super key, a 64-bit random number RN is generated at A and defined to be equal to $e^*_{KERK2\oplus X}$(K), where K is the key (unadjusted for odd parity) that happens to be recovered when RN is decrypted with KERK2 using the CVD algorithm. The XLTCV instruction is now executed (with the parity adjust option specified) first to translate $e^*_{KERK2\oplus X}$(K) to $e^*_{KERK2\oplus C1}$(K) and second to translate $e^*_{KERK2\oplus X}$(K) to $e^*_{KERK2\oplus C2}$(K). The Translate Key instruction is executed to translate $e^*_{KERK2\oplus X}$(K) to $e^*_{KESK1\oplus X}$(K). The XLTCV instruction is executed to translate $e^*_{KESK1\oplus X}$(K) to $e^*_{KESK1\oplus 0}$(K). The RTMK instruction is executed first to import $e^*_{KERK2\oplus C1}$(K) as $e^*_{KM\oplus C1}$(K) and second to import $e^*_{KERK2\oplus C2}$(K) as $e^*_{KM\oplus C2}$(K). From this CFAP can build the super key (($e^*_{KM\oplus C1}$(K), C1),

Table 4 Example control vector translate table for KERK

For KERK:	
Input	Output
0	C1
0	C2

($e^*_{KM\oplus C2}(K)$, C2)). The encrypted key $e^*_{KESK1\oplus 0}(K)$ is sent to B where it is imported and initialized using procedures and protocols in accordance with the key-management rules implemented at device B.

Electronic import of super keys. Again, let A represent a Transaction Security System device and B a device not supporting control vectors. A key-distribution channel is first established from B to A using non-electronic methods, by installing KESK1 at B and installing matching KERK1 at A. We assume that B can export keys encrypted in the form $e^*_{KESK1}(K)$, where K can have permitted uses at B comparable to a super key of the form (($e^*_{KM\oplus C4}(K)$, C4), ($e^*_{KM\oplus C5}(K)$, C5)) at A. We further assume that KERK1 is stored at A in the form ($e^*_{KM\oplus C6}$(KERK1), C6), possibly as part of a super key, where C6 has attribute ⟨rtmk⟩. We further assume that a control vector translate table (CVTT) is prepared in advance to permit the CV mappings to be made as shown in Table 4.

To import a super key, the XLTCV instruction is executed first to translate $e^*_{KERK1\oplus 0}(K)$ to $e^*_{KERK1\oplus C1}(K)$ and second to translate $e^*_{KERK1\oplus 0}(K)$ to $e^*_{KERK1\oplus C2}(K)$. The RTMK instruction is executed first to import $e^*_{KERK1\oplus C1}(K)$ as $e^*_{KM\oplus C1}(K)$ and second to import $e^*_{KERK1\oplus C2}(K)$ as $e^*_{KM\oplus C2}(K)$. From this CFAP can build the super key (($e^*_{KM\oplus C1}(K)$, C1), ($e^*_{KM\oplus C2}(K)$, C2)).

Summary

A key-management scheme based on the control vector has been described in which the complexity associated with key management is associated with the control vector (i.e., in data structures). This technique greatly simplifies the key-management functions and processing steps. Thus, a robust key-management scheme is obtained that can grow to provide new and improved cryptographic services.

Acknowledgments

The authors wish to acknowledge D. B. Johnson, R. K. Karne, R. Prymak, and J. D. Wilkins for their efforts in codeveloping the key-management design in the Transaction Security System.

Cited references and note

1. *Data Encryption Standard*, Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington (January 1977).
2. American National Standard X3.92-1981, *Data Encryption Algorithm*, American National Standards Institute, New York (December 31, 1981).
3. W. F. Ehrsam, S. M. Matyas, C. H. Meyer, and W. L. Tuchman, "A Cryptographic Key Management Scheme for Implementing the Data Encryption Standard," *IBM Systems Journal* **17**, No. 2, 106–125 (1978).
4. S. M. Matyas and C. H. Meyer, "Generation, Distribution, and Installation of Cryptographic Keys," *IBM Systems Journal* **17**, No. 2, 126–137 (1978).
5. R. E. Lennon, "Cryptography Architecture for Information Security," *IBM Systems Journal* **17**, No. 2, 138–150 (1978).
6. C. H. Meyer and S. M. Matyas, *Cryptography: A New Dimension in Computer Data Security*, John Wiley & Sons, Inc., New York (1982).
7. D. W. Davies and W. L. Price, *Security for Computer Networks*, Second Edition, John Wiley & Sons, Inc., New York (1989).
8. D. G. Abraham et al., "Transaction Security System," *IBM Systems Journal* **30**, No. 2, 206–229 (1991, this issue).
9. S. M. Matyas, "Key Handling with Control Vectors," *IBM Systems Journal* **30**, No. 2, 151–174 (1991, this issue).
10. In a companion paper on control vectors appearing in this issue,[9] the more general expression $e^*_{KK\oplus h(C)}(K)$ is used, where h is a hashing function applied to control vector C. However, for 128-bit control vectors, the more convenient expression $e^*_{KK\oplus C}(K)$ can be used in place of $e^*_{KK\oplus h(C)}(K)$.

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

of Iowa. He is the recipient of an Outstanding Innovation Award for his part in the development of the Common Cryptographic Architecture. He is presently an IBM Senior Technical Staff Member.

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

Dennis G. Abraham *IBM US Marketing & Services, 1001 W. T. Harris Boulevard, Charlotte, North Carolina 28257.* Mr. Abraham is a Senior Technical Staff Member in the security system architecture area where he has been a leader in establishing the architecture and function definitions for the IBM Transaction Security System. He attended Fairleigh Dickinson University, Rutherford, New Jersey, where he received his B.S.E.E. degree in 1964. He joined IBM in June 1964 at Endicott, New York, where he held assignments in various product and service groups, including circuit design, logic design, and a strong speciality in servomechanisms including a special expertise in stepper motor control and design. He received his M.S.E.E. from Syracuse University in 1972. He was the lead architect of the IBM 3890 optical character recognition machine. After moving to Charlotte in 1979, he has worked in developing image technology as it applies to check processing. After an assignment in the National Marketing Division headquarters, where he provided technical expertise for the marketing force, Mr. Abraham joined the advanced technology group and was assigned to develop a security strategy and architecture for the Consumer Systems Business Unit. This work lead to the development of the IBM Transaction Security System and the Common Cryptographic Architecture. He holds nine issued patents, ten patent applications on file, and 23 published invention disclosures.

Reprint Order No. G321-5429.