Data being transmitted through a communications network can be protected by cryptography. In a data processing environment, cryptography is implemented by an algorithm which utilizes a secret key, or sequence of bits. Any key-controlled cryptographic algorithm, such as the Data Encryption Standard, requires a protocol for the management of its cryptographic keys. The complexity of the key management protocol ultimately depends on the level of functional capability provided by the cryptographic system. This paper discusses a possible key management scheme that provides the support necessary to protect communications between individual end users (end-to-end encryption) and that also can be used to protect data stored or transported on removable media.

A cryptographic key management scheme for implementing the Data Encryption Standard

by W. F. Ehrsam, S. M. Matyas, C. H. Meyer, and W. L. Tuchman

Cryptography is the only known technologically feasible method of protecting stored data. Cryptography can be used to protect against the threats of passive wiretapping (recording or listening in on transmissions) and active wiretapping (modification of messages or insertion of false or stale messages). It can also prevent accidental or deliberate substitution of one device for another and the exposure of proprietary data by the accidental misrouting of message traffic. Further, cryptography can protect against theft and undetected interference with system resident data and data stored on removable media.

definitions and background

Cryptography deals with the methods involved in preparing *cryptograms*—messages or writings intended to be incomprehensible to all except those who legitimately possess the means to recover the original information. The designer of a cryptographic system, or *cryptosystem*, is a *cryptographer*.

Notice to readers of the IBM Systems Journal

The paper entitled, "A cryptographic key management scheme for implementing the Data Encryption Standard" by W. F. Ehrsam, S. M. Matyas, C. H. Meyer, and W. L. Tuchman, *IBM Systems Journal*, Volume 17, Number 2, pages 106–125, contains a typographical error on page 106. Two lines of type were omitted from the first paragraph. The reverse of this notice is a replacement for page 106, in which the error has been corrected. Please insert it in your copy of Volume 17, Number 2.

Additional copies of the replacement page can be ordered by telephone from the Editor (914-686-5680) or the Publications Manager (914-686-5585), or by writing to the address below.

The *Systems Journal* regrets the error.

C. A. Thiel, Editor
IBM Systems Journal
Dept. 10-786, Box 3-30, Third Floor
44 South Broadway
White Plains, New York 10601

The opponent or antagonist of a cryptosystem is a *cryptanalyst*. *Cryptanalysis* is concerned with techniques used to penetrate communications and recover the original information by means other than those available to the legitimate recipient.

Cryptology is the science of disguised or secret communications. It embraces both cryptography and cryptanalysis.

The basic challenge in cryptography is to devise a method that transforms messages (known as *plaintext*) into cryptograms (known as *ciphertext*) in a cryptographically secure way—that is, the method must withstand intense efforts at cryptanalysis. Plaintext can be protected by either of two techniques: it can be *encoded* using a code system, or it can be *enciphered* (encrypted) using a cipher system.

Code systems require a code book or dictionary that relates the words, phrases, and sentences of the vocabulary (the plaintext) to its equivalent code group (the ciphertext), and vice versa. The number of plaintext messages that can be encoded depends on the number of combinations of phrases that can be obtained from the code book. Although that number may be large, not every combination or pattern of bits can be encoded. Hence the versatility and usefulness of code systems is limited, especially in computer applications.

Cipher systems require two basic elements: a set of rules or steps—that is, an algorithm—constituting the basic cryptographic procedure, which is constant in character and agreed upon in advance, and a specific cryptographic key, selected from a large set of possible keys, which is known only by the communicators.

A cryptographic algorithm can be represented as an extremely large number of possible mathematical procedures called transformations. Each transformation defines how sequences of intelligible data (representing a message) are changed into sequences of apparently random noise (gibberish) that are unintelligible to humans or machines. The cryptographic key is a secretly held sequence of numbers or characters, relatively short, which identifies the transformation to be used.

To be useful, the algorithm should have, for each of its transformations, an inverse operation that changes the gibberish back into intelligible data. (This process is called decipherment or decryptment.) It is assumed here that encipherment and decipherment are performed using the same cryptographic key. Thus, a cryptographic algorithm will provide data security between two nodes of a data processing system if those two nodes have the algorithm installed (in hardware or software) and if both nodes have exact knowledge of the key.

It is important to note that, for good data security, only the key need be kept secret. The details of the algorithm are assumed to be known to everyone. The cryptographic key, therefore, is often considered analogous to the secretly held combination of a safe.

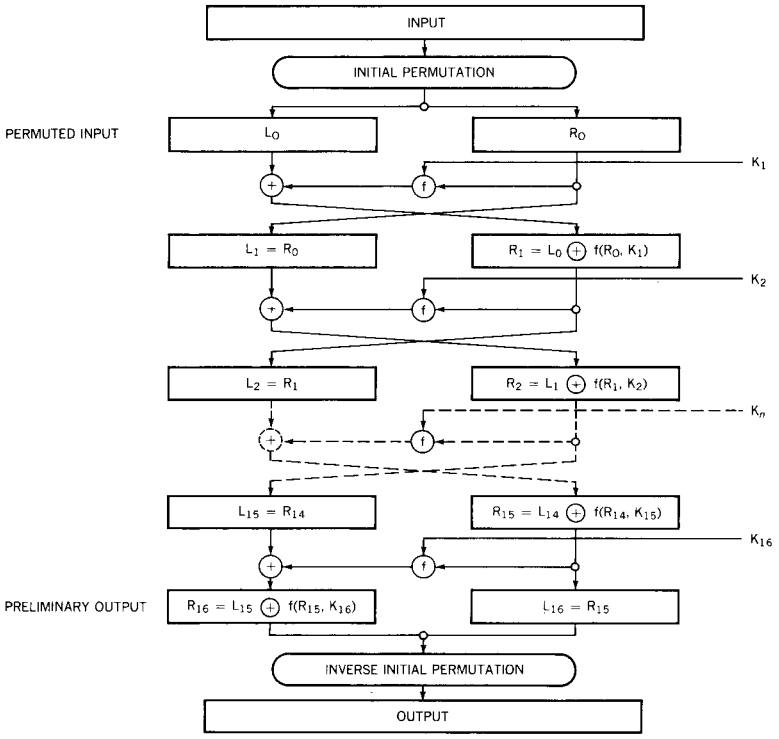
It is always possible to construct an unbreakable system if the number of characters in the key is equal to or greater than the number of characters of plaintext to be enciphered. It is required, however, that the key be randomly selected and used only once. This approach is impractical in data processing systems because of the large amount of message traffic. In a practical approach, the key must be of fixed length, relatively short, and capable of repeated use.

Basically there are three general classes of ciphers: *transposition ciphers*, *substitution ciphers*, and combinations of these called *product ciphers*. A transposition cipher involves the rearrangement or permutation of the plaintext letters without change in their identity. A substitution cipher involves the replacement of plaintext letters by one or more letters (or other symbols) without changing their sequence. Cryptographic research conducted during World War II showed that strong encryption systems could be obtained using alternate steps of substitution and transposition, resulting in a product cipher.[1]

Further research into the development of strong product ciphers was undertaken in the private sector in the late 1960's. During the period from 1968 to 1975, a cryptographic procedure consisting of 16 alternate steps (or rounds) of key-controlled substitution and fixed permutation, based on work done by Horst Feistel, was developed at IBM.[2] This algorithm was accepted as a standard by the National Bureau of Standards and became effective on July 15, 1977. It is known as the Data Encryption Standard (DES).[3]

The DES enciphers a 64-bit (eight-character) block of plaintext into a 64-bit block of ciphertext under the control of a 56-bit cryptographic key. The general process of encryption consists of 16 separate rounds of encipherment, each round using a product cipher approach, or *cipher function*. The interaction of data, cryptographic key K, and cipher function f is illustrated in Figure 1. The externally supplied key K consists of 64 bits: 56 bits are used by the algorithm and eight bits may be used for parity checking. A special shifting scheme on the original 56-bit key is used so that a subset of 48 key bits is used in each round. These subsets of key bits are denoted K_1, K_2 $\cdots$ K_{16}. During decipherment, the rounds are performed in reverse order (K_{16} is used in round one, K_{15} in round two, and so forth).

Figure 1 Enciphering computation (adapted from FIPS Publication 46, National Bureau of Standards[3])



In the following discussion of key management, let X represent a 64-bit block of input data (plaintext), Y a 64-bit block of output data (ciphertext), and K a 64-bit cryptographic key (eight bits may be used for parity checking, hence are not used by the algorithm itself). The notation $E_K(X) = Y$ means that the encipherment (E) of X under key K is equal to Y, and the notation $D_K(Y) = X$ means that the decipherment (D) of Y under key K is equal to X.

In the DES, a cryptographic relationship exists between the plaintext, ciphertext, and cryptographic keys on the one hand and the complements of those quantities on the other hand. That relationship, called the *complementary property* of the DES, can be expressed

$$E_K(X) = \overline{E_{\bar{K}}(\bar{X})}$$

where the bars represent complementation, or bit inversion.

Because of the complementary property of the DES, if an analyst could obtain $E_K(X)$ and $E_K(\bar{X})$ for an arbitrary X, it would be possible to reduce the key space from 2^{56} to 2^{55}. Therefore the key space could be exhausted in 2^{55} trials instead of 2^{56} trials. However, depending on the implementation, it may be impractical for

an opponent to obtain plaintext (X) and its complement ($\bar{X}$) enciphered under the unknown cipher key. Moreover, it has been suggested that in unusual cases, when a longer key is desired, multiple encryption methods (superencipherment) can be used as a means of increasing the work factor (a measure of the strength of a cryptographic procedure). The work factor is an expression of the time and resources required for a competent cryptanalyst, using known techniques and equipment, to solve for the particular key in use, given full knowledge of the algorithm used and large amounts of plaintext (which the cryptanalyst may specify) and corresponding ciphertext. At this writing, the authors are unaware of any demonstrated method of solving for a single key bit, other than by key exhaustion (trying each possible key).

It must be remembered that if the key required for decrypting data is lost or unknown, the data cannot be recovered from the ciphertext. It is just as difficult for the authorized user to decrypt this data when the key is unknown as it is for the opponent.

The DES can be used to obtain either a *block cipher* or a *stream cipher*. In a block cipher, each 64-bit input block is enciphered into a corresponding 64-bit output block of ciphertext. In a stream cipher, a 64-bit pseudorandom initializing vector is used to start a process that produces a long pseudorandom bit stream which can then be added to the plaintext, using modulo 2 addition, to produce the desired ciphertext. Only the block cipher mode will be treated here.

The block cipher can be used in either of two modes of operation: block encryption and chained block encryption. When block encryption is used, each 64-bit block of data is enciphered separately. In chained block encryption, the encipherment of each block is also made dependent on prior information (plaintext, ciphertext, or the like) that is available when the block is enciphered. Two important chained block encryption techniques are *ciphertext feedback* and *plaintext-ciphertext feedback*, as defined below.

Let $X_1, X_2 \cdots X_n$ denote blocks of plaintext to be chained using key K and nonsecret initializing vector Y_0, and let $Y_1, Y_2 \cdots Y_n$ denote the blocks of ciphertext produced. When ciphertext feedback is used, the following relationship holds:

$$Y_i = \mathrm{E}_K(X_i \oplus Y_{i-1}) \quad \text{for} \quad i \geq 1$$

where $\oplus$ represents modulo 2 addition. When plaintext-ciphertext feedback is used, the following relationships hold:

$$Y_1 = \mathrm{E}_K(X_1 \oplus Y_0)$$

and

$$Y_i = \mathrm{E}_K(X_i \oplus Y_{i-1} \oplus X_{i-1}) \quad \text{for} \quad i > 1.$$

Chained block encryption (or block chaining) can be used to extend the strong intersymbol dependence in a single block of ciphertext to include all chained blocks. (Strong intersymbol dependence is a property of the DES, since each ciphertext bit is a complicated function of all plaintext bits and all key bits.) Hence, block chaining can be used to overcome difficulties with highly redundant or structured data.

Plaintext-ciphertext feedback has the additional property of error propagation. Corruption of a single bit of ciphertext will cause each subsequent bit of recovered plaintext to be in error with probability 0.5. By appending a known pattern of bits to the end of the plaintext prior to encryption, and comparing that value to the value recovered, the error propagation feature can be used for checking the true content of a message.

Chaining techniques are useful for encrypting data, and the block cipher (with no chaining) is useful for key transformation operations. To simplify the discussion that follows, chaining methods are not considered.

There are three approaches to incorporating encryption into a communications system: link-by-link, node-by-node, and end-to-end encryption.

In *link-by-link* encryption, data is encrypted across the medium connecting two directly communicating nodes. Link-by-link encryption is logically independent of the system and does not necessarily imply that the cryptographic capability is integrated into the communicating nodes. It can be thought of as implemented by a pair of cryptographic devices bracketing the line between two communicating nodes and situated between the nodes and their modems (modulators or demodulators).

Node-by-node encryption is logically similar to link-by-link encryption in that each link is protected by a unique key. However, the translation from one key to another occurs within a security module, which may be a peripheral device attached to the node. Moreover, plaintext occurs only within the security module, not within the node.

In *end-to-end* encryption, data encrypted at the originating node is not decrypted until it arrives at its final destination. The cryptographic capability is integrated into the participating nodes to the extent that the system can control the setting of the keys and turning the cryptographic capability on and off. The cryptographic capability present at a host processor node could be provided either by a programmed implementation of the DES algorithm or by special hardware integrated into the central processing unit, into a front-end processor, or into the channel.

end-to-end encryption

For end-to-end encryption, the common key that must be present at each node for encryption and decryption can be provided in either of two ways. It can be a private *personal key* associated with and manually entered into the node by each individual user, or it can be a secret *device key* resident within the node, initially installed by authorized personnel, maintained in a nonvolatile store, and shared by many system users. In the latter case, the keys are managed by the system, and the implementation provides *cryptographic transparency*.

Any key-controlled cryptographic algorithm, such as the DES, requires a protocol for safely handling and controlling its cryptographic keys. This aspect of cryptography is called *key management*. Its complexity depends on the level of functional capability provided by the cryptosystem.

The key management scheme presented here enables the DES to be integrated into data processing systems to provide protection for communications between individual end users (end-to-end encryption). With end-to-end encryption, data can be deciphered only at its final intended destination. Data is never exposed at intermediary nodes. Moreover, there is no danger from misdirected messages since each end user has a unique data-encrypting key.

By integrating cryptography into the host system, end-to-end encryption provides a means for encrypting and decrypting stored data. For this reason, the key management scheme can be used to protect data stored and transported on removable media.

The key management scheme involves host processor nodes as well as terminal nodes. Each terminal has a unique terminal master key stored in the clear within the terminal's cryptographic device. The terminal master key is stored also at the host processor to which the terminal is attached and is protected through the use of a special host resident key called the host master key. The master key is the only key stored in clear form at the host processor; all other keys are enciphered.

End-to-end encryption between individual end users is achieved by the use of a common data encrypting key (defined as the primary communication key), which is generated dynamically and remains operable for the duration of the communications session. Hence this key is also called a *session key* (*KS*). When referring to file security, the data encrypting key is defined as the *primary file key*, or *file key* (*KF*).

The session key is a time-variant quantity that is generated at the host processor. It is transmitted to the appropriate terminal under the encipherment of that terminal's master key, where it is recovered and used for protecting data communications.

This basic idea—transmitting or storing a data-encrypting key under the encipherment of a key-encrypting key available at the destination node—can be extended so that session keys and file keys are generated at one host processor and recovered at another. In that case, the data-encrypting key is enciphered under a common key held only by the two participating nodes. When applied to communication security, the common key is called a *secondary communication key* (*KNC*), and when applied to file security, the key is called a *secondary file key* (*KNF*).

The main purpose of the session key is to establish a common data-encrypting key between end nodes that have different master keys. A session may involve several different end users at the same time. Session keys are also useful in limiting the amount of communicated data enciphered under any single key within the system. The limitation reduces the threat that an opponent could successfully build a dictionary of plaintext and ciphertext equivalents which could be used to attack the system.

The host master key concept

Because the DES is a key-controlled algorithm, the protection achieved through encryption ultimately depends on the secrecy of the cryptographic key. If the cipher keys cannot be adequately protected, the use of cryptography does not enhance security; it does little more than create a nuisance factor for the opponent. Consequently, the effectiveness of any cryptographic algorithm, such as the DES, is highly dependent on the techniques used for the selection, handling, and protection of the cryptographic keys used in the ciphering process.

protection of cipher keys

One way to ensure the secrecy of cryptographic keys would be to keep them in a protected area of storage accessible only to the cryptographic algorithm. However, to provide secrecy for these keys when in use by the algorithm, and to prevent intermediate results of each round from being exposed, the algorithm itself would have to be kept in a protected area.

A random access memory (RAM), addressable only by the cryptographic algorithm, would provide a protected area for cipher keys. However, the keys must be available to users authorized to employ them. Access to the keys must be controlled, therefore, by means other than cryptography, such as store-and-fetch protection features or privileged operations.

Because the RAM approach does not provide a means for controlling access to cipher keys, it can be ruled out in favor of an equally secure approach in which a RAM is not needed. In this approach only a single key, the *master key*, is stored in clear form

at the host system. All other keys are protected through encipherment using the master key. Hence the problem of providing secrecy for cipher keys is reduced to providing secrecy for only one key—the master key.

The inaccessible area containing the master key and the cryptographic algorithm is called the *cryptographic facility*. It is assumed that the master key is in nonvolatile storage, so that it need be loaded into the cryptographic facility only once. The master key can be routed to the cryptographic facility by reading it into main memory and exercising a *set master key* operation, which moves the master key from main memory to a nonvolatile storage element in the cryptographic facility, or it can be entered into nonvolatile storage by a manual process using mechanical switches or dials.[4]

The use of encrypted keys within the host system has the advantage, compared with the use of clear keys, that even if an opponent obtains an encrypted key, any intercepted ciphertext can be deciphered only on the system possessing the appropriate master key. (For security purposes, no two master keys will ever be the same, except by pure chance.)

The distinction between encrypted keys and clear keys is important in that the ability of an opponent to obtain an encrypted key does not authorize him to use the cryptographic facility or gain access to the system. With clear keys, on the other hand, a system can be attacked externally. Information obtained by wiretapping, for example, could be decrypted at any computer using a programmed version of the DES algorithm.

As indicated previously, the cryptographic algorithm and the master key are contained in the protected area of the cryptographic facility. This strategy permits the master key, other cipher keys used by the ciphering algorithm, and the results of intermediate rounds of encipherment or decipherment to be kept secret. It is also assumed that the cryptographic facility has an input port for data (which may include clear or encrypted keys), an output port, and a control line to indicate the desired operation. The cryptographic operations necessary to provide key management are developed around these assumptions.

Communication security

time variant keys

Let KS_1, $KS_2 \cdots KS_n$ represent the time-variant, dynamically changing data-encrypting keys used for enciphering and deciphering data. It is assumed that KS is operational for the duration of a communications session (KS is the primary communication key, or session key, as discussed before). Let KMT represent the mas-

ter key of the terminal, and *KMH0* the master key of the host processor. A common session key (*KS*) between two nodes is established by generating *KS* at one node and sending it to the other node in enciphered form. This requires that both nodes share a common key—in this case, *KMT*.

The *encipher* (ECPH) and *decipher* (DCPH) operations available at the host processor are defined as follows:

ECPH: $\{E_{KMH0}(\text{key}), \text{data}\} \rightarrow E_{\text{key}}(\text{data})$

DCPH: $\{E_{KMH0}(\text{key}), E_{\text{key}}(\text{data})\} \rightarrow \text{data}.$

The symbols within brackets indicate the cryptographic facility input, and the arrow points to the result.

Since the terminal can be in communication with only one node at a time, the terminal's *encipher* and *decipher* operations are handled differently than those for the host processor. When $E_{KMT}(KS)$ is received at the terminal, *KS* is first recovered by exercising a *decipher under master key* (DMK) operation:

DMK: $\{E_{KMT}(KS)\} \rightarrow KS.$

This is accomplished by deciphering $E_{KMT}(KS)$ with the current value of the terminal master key stored within the cryptographic facility. In a strict sense, DMK is not really a decipher-under-master-key operation, since the result never leaves the cryptographic facility but is stored in the working key register, where it remains until changed by a key management operation or until power to the terminal is turned off.

The *encipher* and *decipher* operations available at the terminal are defined as follows:

ECPH: $\{\text{data}\} \rightarrow E_{KS}(\text{data})$

DCPH: $\{E_{KS}(\text{data})\} \rightarrow \text{data}.$

They operate under the key value stored in the working key register. By protocol, that value is the communication key *KS*.

Since *KS* is generated dynamically for each communications session, $E_{KMT}(KS)$ also is a dynamic quantity. By storing *KMT* at the host processor where *KS* can be readily generated, it is possible for $E_{KMT}(KS)$ to be produced. A capability for session key generation at a single host processor is more economical than duplicating the same function across many terminals.

To satisfy the condition that no clear key occur outside the facility, at the same time avoiding the need to generate *KS* directly within this secure area, the following method for session key generation is adopted: a 64-bit pseudorandom number *RN* is gener-

ated* and is defined to be the session key enciphered under the master key of the requesting node (host processor H in the present example). At host processor H, *RN* is thus defined as

$$RN \equiv E_{KMH0}(KS).$$

The quantity *RN* is used directly at H to encipher and decipher data. To obtain $E_{KMT}(KS)$, which is required at terminal T, a transformation must be applied to $E_{KMH0}(KS)$. This transformation is accomplished by deciphering $E_{KMH0}(KS)$ with the value of *KMH*0 stored in the cryptographic facility and reenciphering *KS* with *KMT*. As previously stipulated, *KMT* is stored at the host processor for the purpose of accomplishing this translation.

Since *KMT* must not be stored in clear form at H, it would appear that storing *KMT* under *KMH*0 encipherment might solve the problem. That procedure, however, would create an exposure to *KS*, since entering $E_{KMH0}(KMT)$ and $E_{KMT}(KS)$ into the *decipher* operation would yield

$$\text{DCPH: } \{E_{KMH0}(KMT), E_{KMT}(KS)\} \rightarrow KS.$$

This condition violates the stipulation that clear keys should not occur outside the cryptographic facility. The quantity $E_{KMT}(KS)$ could be obtained, for example, through a wiretap, and the quantity $E_{KMH0}(KMT)$ could become exposed during storage at host processor H.

two master keys

The situation described above can be avoided by defining a second master key, *KMH*1. Instead of storing *KMT* under *KMH*0 encipherment, it is stored under *KMH*1 encipherment. Thus the translation from $E_{KMH0}(KS)$ to $E_{KMT}(KS)$ is accomplished by using $E_{KMH1}(KMT)$. This procedure requires a new translation capability, defined as the *reencipher from master key* (RFMK) operation:

$$\text{RFMK: } \{E_{KMH1}(KMT), E_{KMH0}(KS)\} \rightarrow E_{KMT}(KS).$$

In a practical implementation of the two-master-key approach, only one key actually resides in clear form within the cryptographic facility (*KMH*0 in the present example), so the user sees only a one-master-key system. The second master key can be derived internally within the cryptographic facility, for example by selected inversion of bits within *KMH*0. (*KMH*1 is defined, in this case, to be the first variant of the master key *KMH*0.)

It must be realized that the use of two keys that do not differ much from each other is tolerable only if their use does not result in exploitable correlations of the ciphertext produced when the same plaintext is enciphered with each of the keys. Since, for the DES, even a single bit change in the key has a drastic effect on the

*The generation of cryptologically secure keys requires a careful technical procedure. See the discussion of key generation in S. M. Matyas and C. H. Meyer, "Generation, distribution, and installation of cryptographic keys," *IBM Systems Journal*, this issue, page 126.

ciphertext,[5] there is no practical way to compute $E_{KMH0}(KMT)$ from $E_{KMH1}(KMT)$; hence the proposed scheme is cryptographically strong.

File security

problems in storing enciphered data

The previous section described a key management scheme for protecting data communications. It seems natural to ask if the key management scheme for communication security can be adapted easily for use with file security, the protection of stored data.

Suppose we want to protect stored data in the same way that communicated data is protected; that is, we want to use a session key in the form $E_{KMH0}(KS)$ in conjunction with the *encipher* and *decipher* operations to respectively create and recover a file. For data to be recoverable, the quantity $E_{KMH0}(KS)$ must be saved for later use or else recreated when needed. If $E_{KMH0}(KS)$ is stored within the system, especially for long periods, it must be protected by a suitable method of controlled access because knowledge of $E_{KMH0}(KS)$ would allow data to be recovered directly with the *decipher* operation. The difficulty could be avoided, of course, by using this quantity as a personal key and not storing it within the system. However, the advantage of a personal key must be weighed against that of cryptographic transparency, in which the user is relieved of any responsibility for handling keys. When stored information is shared among many users, the system-managed key may be the only pragmatic solution.

It must be anticipated that the master key may eventually change, whether $E_{KMH0}(KS)$ is stored in the system or used as a personal key. Therefore a method must exist for recovering *KS* in the clear so it can be enciphered under the new master key, or else there must be a method for translating *KS* directly from encipherment under the old master key to encipherment under the new master key. In either case, the procedure would be cumbersome because of the many different *KS* values.

Still another disadvantage in basing a file recovery strategy on the stored quantity $E_{KMH0}(KS)$ is that recovery at a different host processor would not be practical since it would require that *KMH*0 be revealed to the other host, and the master key is too important to be shared with another host processor.

These disadvantages mitigate against the use of $E_{KMH0}(KS)$ as the quantity that should be saved for later use in file recovery operations. The disadvantages can be overcome, however, by the use of a *secondary file key* (*KNF*) so that *KS* can be stored under *KNF* encipherment rather than *KMH*0 encipherment.

One way to incorporate the idea of a secondary file key would be to store the quantity $E_{KNF}(KS)$ in the file header. At the host processor, *KNF* would be stored under the encipherment of some key-encrypting key (the choice of this key-encrypting key will be discussed later). Recovery of data would be accomplished by reading $E_{KNF}(KS)$ from the file header, obtaining access to the value of *KNF* stored at the host processor, and regenerating the quantity $E_{KMH0}(KS)$ by an appropriate translation operation.

Two questions left unanswered are what key can *KNF* be safely enciphered under, and what type of translation operation will allow *KS* to be translated from encipherment under *KNF* to encipherment under *KMH*0.

With the dual master key approach, in which *KMH*0 and *KMH*1 are available, *KNF* should not be stored under *KMH*0 encipherment. If it were, the *decipher* operation could be used to obtain a clear session key, as follows:

DCPH: $\{E_{KMH0}(KNF), E_{KNF}(KS)\} \rightarrow KS$.

Again, this violates the strategic principle that no cryptographic operation should allow clear keys to be recovered from any cryptographic quantities that are routinely stored or routed through the cryptosystem.

There is no such exposure in storing *KNF* under *KMH*1 encipherment. However, if both *KNF* and *KMT* are enciphered under *KMH*1, an opponent who has access to the host processor still would be able to recover $E_{KMH0}(KS)$ from $E_{KMT}(KS)$, and therefore could decipher intercepted ciphertext. Therefore a better isolation between communication security and file security is desirable. Stated more simply, the cryptographic operation that allows $E_{KMH0}(KS)$ to be recovered from $E_{KNF}(KS)$ should not allow $E_{KMH0}(KS)$ to be recovered from $E_{KMT}(KS)$.

file keys

One way to achieve separation between communication security and file security is to encipher the terminal master keys and secondary file keys under different variants of the host master key. This can be accomplished by storing terminal master keys under *KMH*1 encipherment (the first variant of *KMH*0), and storing secondary file keys under *KMH*2 encipherment (the second variant of *KMH*0). *KMH*2 is derived from *KMH*0 in a manner similar to that of *KMH*1, by inverting selected bits in *KMH*0. (A precise specification for *KMH*2 is not important to our discussion.)

The *reencipher to master key* (RTMK) operation therefore is defined as

RTMK: $\{E_{KMH2}(KNF), E_{KNF}(KS)\} \rightarrow E_{KMH0}(KS)$.

Since the key *KS* normally is associated with communications sessions, the key *KF* will be used to denote encipherment within file security applications. *KF*, in this case, is called a *primary file key*, or *file key* for short. Using this new notation, the quantities stored on the file become $E_{KNF}(KF)$ and E_{KF}(data). The protocol for generating *KF* is slightly different from that for *KS*. The pseudorandom number *RN* is defined to be equivalent to $E_{KNF}(KF)$ instead of $E_{KMH0}(KF)$. Thus the RTMK operation is used to produce $E_{KMH0}(KF)$ from $E_{KNF}(KF)$. This quantity in turn is used with the ECPH operation to encipher data.

Since the quantity $E_{KNF}(KF)$ is not dependent on the master key, a change of master keys will not require that $E_{KNF}(KF)$ be retranslated. Therefore, storing $E_{KNF}(KF)$ on the file header has the advantage that a change in master keys does not require that the file header be changed. The only change required is that, at the host system, *KNF* must be reenciphered from its encipherment under the old value of *KMH*2 to encipherment under the new value of *KMH*2. Moreover, since *KNF* does not take on the importance of a master key, a recovery protocol at other nodes is possible (assuming that *KNF* is given to the other nodes).

If cryptographic transparency is desired, the quantity $E_{KNF}(KF)$ can be written on the data file together wih the encrypted data (as mentioned before). Access to the data, in this case, can be enforced by making the RTMK operation privileged and by controlling read access to the quantity $E_{KMH2}(KNF)$. In an alternate approach, $E_{KNF}(KF)$ can be treated as a personal key and not stored within the system or written on the data file. Under these circumstances, access to data additionally requires that this (secret) quantity be provided to the system at the time data is to be recovered.

Encryption between host nodes

In the previous section, a key management scheme was described that allows communication security and file security to be achieved when a single host processor node and a multiplicity of terminal nodes define the communications network. In this section, the key management scheme is extended to include many host processor nodes. Note that instead of using *KMH*0, *KMH*1, and *KMH*2, the shortened notation *KM*0, *KM*1, and *KM*2 is now employed.

a protocol for file security

Let i and j denote two host nodes whose master keys are $KM0^i$ and $KM0^j$, respectively. The following secondary file keys are then defined:

- KNF^{ii}—known only by node i; allows files enciphered at i to be recovered only at i (a similar key KNF^{ij} is available at j);
- KNF^{ij}—shared by nodes i and j; allows files enciphered at i to be recovered only at j (a similar key KNF^{ji} is available at i and j for data flow in the reverse direction).

It is important that the secondary file keys be stored (enciphered) under the proper variant of the master key. Basically, encipherment under the first variant allows recovery at a different node, whereas encipherment under the second variant allows recovery at the same node.

The protocol in which data is not shared with other nodes has already been discussed. At node i, a pseudorandom number *RN* is defined as

$$RN \equiv E_{KNF^{ii}}(KF).$$

Using the *reencipher to master key* (RTMK) operation, $E_{KM0^i}(KF)$ can be obtained from $E_{KM2^i}(KNF^{ii})$ by exercising

RTMK: $\{E_{KM2^i}(KNF^{ii}), RN\} \rightarrow E_{KM0^i}(KF)$.

Since *RN* is written on the encrypted file, $E_{KM0^i}(KF)$ is regenerated in the same manner during the recovery phase. Once $E_{KM0^i}(KF)$ is produced, the *encipher* and *decipher* operations can be used to encrypt and decrypt the file.

The protocol that allows files to be encrypted at node i and recovered at node i is described as follows: At node i, a pseudorandom number *RN* is defined as

$$RN \equiv E_{KM0^i}(KF).$$

RN can be used directly in the *encipher* (ECPH) operation to encrypt the file by exercising

ECPH: $\{RN, \text{data}\} \rightarrow E_{KF}(\text{data})$.

The *reencipher from master key* (RFMK) operation is then used to generate $E_{KNF^{ij}}(KF)$ by exercising

RFMK: $\{E_{KM1^i}(KNF^{ij}), E_{KM0^i}(KF)\} \rightarrow E_{KNF^{ij}}(KF)$.

The quantity $E_{KNF^{ij}}(KF)$ is recorded on the file so that recovery is possible at node j. The *reencipher to master key* (RTMK) operation is used at node j to generate $E_{KM0^j}(KF)$ by exercising

RTMK: $\{E_{KM2^j}(KNF^{ij}), E_{KNF^{ij}}(KF)\} \rightarrow E_{KM0^j}(KF)$.

The *decipher* (DCPH) operation can now be used to recover data by exercising

DCPH: $\{E_{KM0^j}(KF), E_{KF}(\text{data})\} \rightarrow \text{data}$.

The protocol being described has the nice feature that encrypted data files at node i can easily be sent to node j without the data

under KF having to be deciphered and reenciphered under a new KF. This is accomplished by using the RTMK operation to recover $E_{KM0^i}(KF)$ from $E_{KNF^{ii}}(KF)$, and using the RFMK operation to generate $E_{KNF^{ij}}(KF)$ from $E_{KM0^i}(KF)$, by exercising

RTMK: $\{E_{KM2^i}(KNF^{ij}), E_{KNF^{ii}}(KF)\} \rightarrow E_{KM0^i}(KF)$

RFMK: $\{E_{KM1^i}(KNF^{ij}), E_{KM0^i}(KF)\} \rightarrow E_{KNF^{ij}}(KF)$

and replacing $E_{KNF^{ii}}(KF)$ with $E_{KNF^{ij}}(KF)$ on the encrypted file. (This replacement may require that the file be copied to another volume.)

a protocol for communication security

Again, let i and j denote host nodes whose master keys are $KM0^i$ and $KM0^j$, respectively. The collection of nodes consisting of the host processor i and all its logically associated terminals will be defined as domain i. A similar domain can be defined for host processor j.

The problem at this point is how to establish a common session key KS between two domains, for example between a terminal in domain i and an application program in domain j.

To establish a common KS between domains i and j, the host processors must share a common key, which should not be the host master key of either system. Instead, the host processors should share a special key that can be used only for sending session keys from one domain to the other. The cryptographic key used for this purpose is called a *secondary communication key* (KNC).

In the protocol to be discussed, the following secondary communication keys are defined:

- KNC^{ii}—known only by host processor i; allows a session key generated at host processor i to be established between two nodes within domain i; a similar key KNC^{jj} is defined at host processor j;
- KNC^{ij}—shared by host processors i and j; allows a session key generated at host processor i to be transmitted and recovered at host processor j; a similar key KNC^{ji} is available at i and j to allow the session key to flow in the reverse direction.

The reader will notice the symmetry between secondary communication keys on the one hand, and secondary file keys on the other. Encipherment of either a secondary communication key or a secondary file key under the first variant of the host master key allows a data-encrypting key to be forwarded and recovered at another node, whereas encipherment of either a secondary communication key or a secondary file key under the second variant allows a data-encrypting key to be recovered at that same node.

Generally speaking, there will be only one KNC^{ij} key and one KNC^{ji} key, but many KNC^{ii} and KNC^{jj} keys. KNC^{ii} and KNC^{jj} are defined in order to emphasize the symmetry between communication security and file security. In terminals, the master key KMT performs the function of a secondary communication key; that is, it is used as a key under which the session key is transmitted to the terminal, but not as a key under which keys are enciphered for storage at the terminal. Hence the set $\{KNC^{ii}\}$ actually represents the set $\{KMT_1^i, KMT_2^i \cdots KMT_n^i\}$.

The method for establishing a common session key within a single domain—that is, between host processor and terminal—has already been covered. The key management protocol for single domain communications is not affected by the expanded protocol for cross domain communications; hence nothing more need be said about single domain communications.

The method for establishing a common session key between two domains, say from domain i to domain j, is first to define a pseudorandom number RN at node i as follows:

$$RN \equiv E_{KM0^i}(KS).$$

RN can then be used directly in the ECPH or DCPH operations to encrypt or decrypt data, or it can be used with the RFMK operation to transform KS under the encipherment of a terminal master key belonging to domain i. To send KS to domain j, the RFMK operation is used at node i to generate $E_{KNC^{ij}}(KS)$ by exercising

$$\text{RFMK: } \{E_{KM1}(KNC^{ij}), E_{KM0^i}(KS)\} \rightarrow E_{KNC^{ij}}(KS).$$

The quantity $E_{KNC^{ij}}(KS)$ is then transmitted to node j, where the RTMK operation is used to recover $E_{KM0^j}(KS)$ by exercising

$$\text{RTMK: } \{E_{KM2^j}(KNC^{ij}), E_{KNC^{ij}}(KS)\} \rightarrow E_{KM0^j}(KS).$$

This quantity can then be used directly in the ECPH or DCPH operations at node j, or it can be used with an RFMK operation to transform KS under the encipherment of a terminal master key belonging to domain j.

Note that the quantity $E_{KNC^{ij}}(KS)$ is transmitted from node i to node j over an exposed path. An opponent who recovers this value by wiretapping can make no use of the quantity at node i since the value $E_{KM2^i}(KNC^{ij})$ is nowhere stored in the system. (The RTMK operation cannot be effectively used at node i.) On the other hand, at node j the opposite is true. The same protocol that must permit node j to recover $E_{KM0^i}(KS)$ can potentially be subverted by the opponent. Because of this, the RTMK operation should be made privileged and its execution carefully controlled by the system. In summary, when a secondary communication key is used in conjunction with the above described protocol to forward a session key from one domain to another, recovery of

the session key at the destination (host processor) must be protected by means other than cryptography.

A key management scheme could be defined, however, for which it would not be possible to recover the session key at either node i or j when only information obtained through wiretapping was used. This protocol would require that the session key be a composite of random data generated at each node. The disadvantage would be that additional cryptographic operations would have to be performed as part of the procedure for establishing the session key.

Summary

A key management protocol has been described that will allow the Data Encryption Standard (DES) to be integrated into electronic data processing systems for the purpose of obtaining communication security and file security. Several cryptographic keys have been defined that allow the desired key management protocol to be achieved. They are:

- Host master key (*KM0*)
- First variant of the host master key (*KM1*)
- Second variant of the host master key (*KM2*)
- Terminal master key (*KMT*)
- Secondary communication key (*KNC*)
- Secondary file key (*KNF*)
- Primary communication key, or session key (*KS*)
- Primary file key, or file key (*KF*).

The following cryptographic operations are used by the key management scheme at the host processor:

Set master key
SMK: {key}

Encipher under master key
EMK: {key} $\rightarrow E_{KM0}$(key)

Encipher
ECPH: {$E_{KM0}(KS)$, data} $\rightarrow E_{KS}$(data)

Decipher
DCPH: {$E_{KM0}(KS)$, E_{KS}(data)} $\rightarrow$ data

Reencipher from master key
RFMK: {$E_{KM1}(KMT)$, $E_{KM0}(KS)$} $\rightarrow E_{KMT}(KS)$

Reencipher to master key
RTMK: {$E_{KM2}(KNF)$, $E_{KNF}(KF)$} $\rightarrow E_{KM0}(KF)$.

The following cryptographic operations are used by the key management scheme at the terminal:

Decipher from master key
DMK: $\{E_{KMT}(KS)\} \rightarrow KS$

Encipher
ECPH: $\{\text{data}\} \rightarrow E_{KS}(\text{data})$

Decipher
DCPH: $\{E_{KS}(\text{data})\} \rightarrow \text{data}$.

Data-encrypting keys stored (or in use) at the host processor are protected by a host master key ($KM0$). This concept is expanded to include the storing of key-encrypting keys under host master key variants $KM1$ and $KM2$. The master key and its variants provide a means of enforcing the separation or stratification of cryptographic keys into groups that are both logically and functionally different.

Broadly speaking, cryptography reduces the problem of protecting data, in certain clearly defined situations, to that of protecting the secrecy and use of a small set of cryptographic keys. Without cryptography, a system is vulnerable to such external attacks as wiretapping and theft of removable storage. When cryptography is used, the opponent is forced to attack the system from within.

Control over the execution of the defined cryptographic operations can be exercised by:

- activating or deactivating certain cryptographic operations with a physical, key-operated switch;
- maintaining the secrecy of certain special cryptographic quantities used as input to cryptographic operations;
- making certain cryptographic operations privileged.

It must be emphasized that without system integrity, cryptography will not add significantly to the overall security of a system when the opponent is an authorized user or can gain entry to the system. Although cryptography can enhance the integrity of a computer system, it is not sufficient for integrity. When used in conjunction with other security features, cryptography does play an important and valuable role in a total security plan.

CITED REFERENCES

1. C. E. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal* **28,** 656–715 (1949).
2. H. Feistel, "Cryptography and computer privacy," *Scientific American* **228,** No. 5, 15–23 (May 1973).
3. *Data Encryption Standard*, Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington, DC (January 1977).
4. S. M. Matyas and C. H. Meyer, "Generation, distribution, and installation of cryptographic keys," *IBM Systems Journal* **17,** No. 2, 126–137 (1978, this issue).

5. C. H. Meyer, "Ciphertext/plaintext and ciphertext/key dependence vs number of rounds for the Data Encryption Standard," *AFIPS Conference Proceedings* **47** (June 1978, to be published).

Reprint Order No. G321-5066.