CHAPTER FIVE

The Host System Cryptographic Operations[1]

Since an opponent masquerading as an authorized user could access the host's cryptographic facility, the cryptographic operations at a host (Chapter 4) must be such that keys are not exposed as a result of exercising these operations in some chosen manner. For this reason, the operations are discussed in depth, providing the reader with additional insight into their design.

The host's basic cryptographic operations are invoked via a programmed interface to the host's cryptographic facility. However, a *set master key* operation, used to store a host master key in the cryptographic facility, is enabled via a physical key lock that operates an associated electromechanical lock. The key lock isolates the set master key operation from the other cryptographic operations, thus affording maximum protection to the master key.

In a data communications network, a host system's domain refers to the set of resources managed by that host system. Data communications involving only a single host system are referred to as *single-domain communications*, whereas those involving more than one host system are referred to as *multiple-domain communications.*

One master key provides adequate security for single-domain communications when pregenerated data-encrypting keys are used. Two master keys are necessary if dynamically generated data-encrypting keys are desired. And if file security (the storage of ciphered data) is also desired, three master keys are required. Also, three master keys are sufficient to provide for single-domain and multiple-domain COMSEC and FILESEC using dynamically generated data-encrypting keys.

SINGLE-DOMAIN COMMUNICATION SECURITY USING PREGENERATED PRIMARY KEYS

Pregenerated primary keys are primary keys generated under secure conditions in a single large group before they are needed, and stored within the

[1] © 1978 IBM Corporation. Reprinted in part from *IBM Systems Journal,* **17**, No. 2 (1978) [1].

system for later use. A cryptographic system with one master key permits single-domain communication security using pregenerated primary keys.

Let KC1, KC2, . . . , KCn denote the pregenerated primary communication keys used for ciphering data. If there is only one master key, KM0, then KM0 is used to protect all primary communication keys stored at the host system for use in subsequent data ciphering operations. Keys are stored in the form

$$E_{KM0}(KC1), E_{KM0}(KC2), \ldots, E_{KM0}(KCn)$$

The host master key KM0 is inserted into the master key storage of the host's cryptographic facility via a *set master key* (SMK) operation, as follows:

$$SMK: \{KM0\}$$

It is assumed that the primary communication keys are generated ahead of time, and that these keys are enciphered under the host master key KM0 with the *encipher under master key* (EMK) operation, as follows:

$$EMK: \{KC\} \longrightarrow E_{KM0}(KC)$$

The *encipher data* (ECPH) and *decipher data* (DCPH) operations at the host system are defined as

$$ECPH: \{E_{KM0}(Key), Data\} \longrightarrow E_{KEY}(Data)$$
$$DCPH: \{E_{KM0}(Key), E_{Key}(Data)\} \longrightarrow Data$$

and hence the enciphered primary communication keys stored at the host system are in a form that can be used directly as inputs to these operations. The ECPH and DCPH operations are defined in such a way that a primary communication key (KC) can be used for ciphering data only after the quantity $E_{KM0}(KC)$ has been deciphered under KM0 and the resulting value of KC transferred to the working key storage of the host's cryptographic facility. This complies with the requirement that KC should not appear in clear form outside the cryptographic facility, except when it is generated and initially enciphered under KM0.

Each terminal in the domain of the host system has its own unique secondary communication key, also called the terminal master key (KMT). The terminal master keys are generated under secure conditions at the host system, and each key is distributed in a secure manner (as by courier) to its respective terminal where it is installed in the master key storage element of the terminal's cryptographic facility.

Once installed, the terminal master key is used to protect the primary communication keys as they are sent from the host system to the terminal. At the terminal, the enciphered primary communication key is first deciphered under the terminal master key, and the resultant value of KC is transferred to the working key storage of the terminal's cryptographic facility (where it can be used for subsequent ciphering operations).

Also, as part of the initialization, the list of primary communication keys is divided into as many separate groups of keys as there are terminals. Each group of keys is then enciphered under a different terminal master key. For example, if there were 5 terminals and 5000 primary communication keys, then the table of enciphered keys would be shown by Table 5-1.

This table of enciphered primary communication keys is stored at the host system, and individual keys are selected from the table and sent to their respective terminals as needed. Alternatively, the keys in each row of the table could be stored at their respective terminals. Individual keys could then be selected, as needed, from this locally stored list.

The table of enciphered primary communication keys is produced by using both the *encipher under master key* (EMK) operation and the *encipher data* (ECPH) operation. A terminal master key (KMT) is first enciphered under the host master key (KM0) using an EMK operation. The quantity $E_{KM0}(KMT)$ is then used in an ECPH operation to encipher a primary communication key (KC) under the terminal master key (KMT):

$$\text{EMK: } \{KMT\} \longrightarrow E_{KM0}(KMT)$$

$$\text{ECPH: } \{E_{KM0}(KMT), KC\} \longrightarrow E_{KMT}(KC)$$

After the table of enciphered primary communication keys has been produced, all quantities used in its generation are erased from the host system's main storage.

To implement the approach described here, four basic cryptographic operations are needed at the host system: *set master key* (SMK), *encipher under master key* (EMK), *encipher data* (ECPH), and *decipher data* (DCPH). (See Cryptographic Operations at a Host, Chapter 4.) The approach requires, in addition, only one master key (KM0).

The disadvantages of this approach are that the number of primary communication keys needed by the cryptographic system must be determined in advance, and the storage space for these keys must be provided by the host system. There is also a danger that one or more of these enciphered keys may become known to an opponent. In that case, the quantity $E_{KM0}(KC)$ could be used by an opponent directly as input to a *decipher data* operation

Terminal 1	$E_{KMT1}(KC1)$,	$E_{KMT1}(KC2)$,	..., $E_{KMT1}(KC1000)$
Terminal 2	$E_{KMT2}(KC1001)$,	$E_{KMT2}(KC1002)$,	..., $E_{KMT2}(KC2000)$
⋮	⋮	⋮	⋮
Terminal 5	$E_{KMT5}(KC4001)$,	$E_{KMT5}(KC4002)$,	..., $E_{KMT5}(KC5000)$

Table 5-1. Table of Primary Communication Keys Enciphered Under Terminal Master Keys

to decipher intercepted ciphertext, provided that access to the host system could be obtained.

SINGLE-DOMAIN COMMUNICATION SECURITY USING DYNAMICALLY GENERATED PRIMARY KEYS

The previous section described a protocol in which one host master key was needed to protect the pregenerated keys used for communication security in a single domain. The question now arises whether one master key is sufficient if dynamically generated session keys are used.

Let KS1, KS2, . . . , KSn denote the dynamically generated primary communication keys used for ciphering data. Each KS is operational only for the duration of a communications session, and hence is called a *session key* (see Cipher Key Allocation, Chapter 4). Since there is only one host master key, KM0, session keys are maintained at the host system in the form

$$E_{KM0}(KS1), E_{KM0}(KS2), \ldots, E_{KM0}(KSn)$$

Session keys required by the cryptographic system are generated at the host processor. This is because a single host facility is more economical than multiple facilities duplicating the same function at several terminals.

To satisfy the condition that no clear key occurs outside the cryptographic facility, and yet avoid a requirement to generate KS directly within the cryptographic facility, an indirect method of generating session keys is adopted. A 64-bit pseudo-random number (RN) is generated (Chapter 6) and defined to be the session key enciphered under the requesting node's host master key (KM0):

$$RN = E_{KM0}(KS)$$

Session keys are therefore produced as a sequence of pseudo-random numbers:

$$RN1, RN2, \ldots, RNn$$

where the ith pseudo-random number (RNi) corresponds to the ith encrypted session key (i.e., $RNi = E_{KM0}(KSi)$).

This method for generating session keys also has the advantage that the quantity RN can be used directly at the host system to encipher and decipher data:

$$\text{ECPH: } \{RN, \text{Data}\} \longrightarrow E_{D_{KM0(RN)}}(\text{Data})$$

$$\text{DCPH: } \{RN, E_{D_{KM0(RN)}}(\text{Data})\} \longrightarrow \text{Data}$$

where

$$D_{KM0}(RN) = KS$$

Because session keys are generated in enciphered form, it is not possible to encipher them directly under a terminal master key (KMT) by using the EMK and ECPH operations (see Single Domain Communication Security Using Pregenerated Primary Keys). Thus to obtain $E_{KMT}(KS)$, which is required at the terminal, a cryptographic operation is needed to transform KS from encipherment under KM0 to encipherment under KMT. This transformation is accomplished by deciphering $E_{KM0}(KS)$ with the value of KM0 stored in the host's cryptographic facility and reenciphering KS with the terminal master key (KMT). (KMT is stored at the host system and is provided as an input parameter to the host's cryptographic facility as needed.)

A cryptographic system, like the one being discussed, could be constructed with one master key if KMT were stored enciphered under KM0. Such a system, however, would expose session keys, since using $E_{KM0}(KMT)$ and $E_{KMT}(KS)$ as inputs to a *decipher data* operation would yield a clear value of KS:

$$\text{DCPH: } \{E_{KM0}(KMT), E_{KMT}(KS)\} \longrightarrow KS$$

This condition violates the stated requirement that *it must not be possible to recover keys in the clear outside a designated physically secure area, such as a cryptographic facility.*

Two Master Keys

The situation described above can be avoided by defining a second master key, KM1. In this case, session keys are maintained at the host system in the form

$$E_{KM0}(KS1), E_{KM0}(KS2), \ldots, E_{KM0}(KSn)$$

and terminal master keys are maintained at the host system in the form

$$E_{KM1}(KMT1), E_{KM1}(KMT2), \ldots, E_{KM1}(KMTn)$$

The terminal master keys are generated under secure conditions at the host system, and each key is distributed in a secure manner (as by courier) to its respective terminal where it is installed in the master key storage of the terminal's cryptographic facility. This approach, however, requires a translation capability, defined as the *reencipher from master key* operation:

$$\text{RFMK: } \{E_{KM1}(KMT), E_{KM0}(KS)\} \longrightarrow E_{KMT}(KS)$$

Even though the relationship between KM0 and KM1 is publicly known, this information is not enough to permit the algorithm to be broken. For instance, $E_{KM0}(KMT)$ cannot be deduced from $E_{KM1}(KMT)$ for a strong cryptographic algorithm (see Protection of Host Keys, Chapter 4).

The attack described above for recovering session keys is thwarted when KMT is stored enciphered under KM1. Neither is it possible to deduce

$E_{KM0}(KMT)$ from $E_{KM1}(KMT)$, nor is it possible to enter $E_{KM1}(KMT)$ and $E_{KMT}(KS)$ as inputs to a *decipher data* operation and recover a clear session key (KS):

$$\text{DCPH: } \{E_{KM1}(KMT), E_{KMT}(KS)\} \longrightarrow D_k(E_{KMT}(KS)) \neq KS$$

because

$$K = D_{KM0}(E_{KM1}(KMT)) \neq KMT$$

The host master key (KM0) is inserted into the master key storage of the host's cryptographic facility with an SMK operation (as previously discussed). Because session keys are generated in a form enciphered under the host master key (KM0), the EMK operation is not needed to encipher dynamically generated primary keys as it is for pregenerated primary keys.

Encipherment Under KM1

Encipherment of the terminal master keys under KM1 can be accomplished under secure conditions by a combination of cryptographic operations already discussed.[2] The steps involved in this procedure are as follows. The host master key (KM0) is first read into the main storage of the host system, where the variant KM1 is derived by inversion of appropriate bits of KM0. The intermediate quantity $E_{KM1}(KM1)$ is then derived as follows:

$$\text{EMK: } \{KM1\} \longrightarrow E_{KM0}(KM1)$$

$$\text{ECPH: } \{E_{KM0}(KM1), KM1\} \longrightarrow E_{KM1}(KM1)$$

Using the intermediate quantity $E_{KM1}(KM1)$, each terminal master key (KMT) is then enciphered under KM1 as follows:

$$\text{EMK: } \{KMT\} \longrightarrow E_{KM0}(KMT)$$

$$\text{RFMK: } \{E_{KM1}(KM1), E_{KM0}(KMT)\} \longrightarrow E_{KM1}(KMT)$$

Once the list of terminal master keys is enciphered, all intermediate values used in the computation are erased from the host's main storage.

An Example of Communications Encryption

A communication session between a terminal and an application program in a host is initiated as follows. A 64-bit pseudo-random number (RN) is generated at the host system and is defined to be the session key (KS) enciphered under the host's master key (KM0), that is, $RN = E_{KM0}(KS)$. Since the host's master key (KM0) is unavailable at the terminal, $E_{KM0}(KS)$ must be transformed into a form usable at the terminal, that is, into the form $E_{KMT}(KS)$.

[2] Another approach is to use a new cryptographic operation (see Encipherment of Keys Under the Master Key's Variants, Chapter 6).

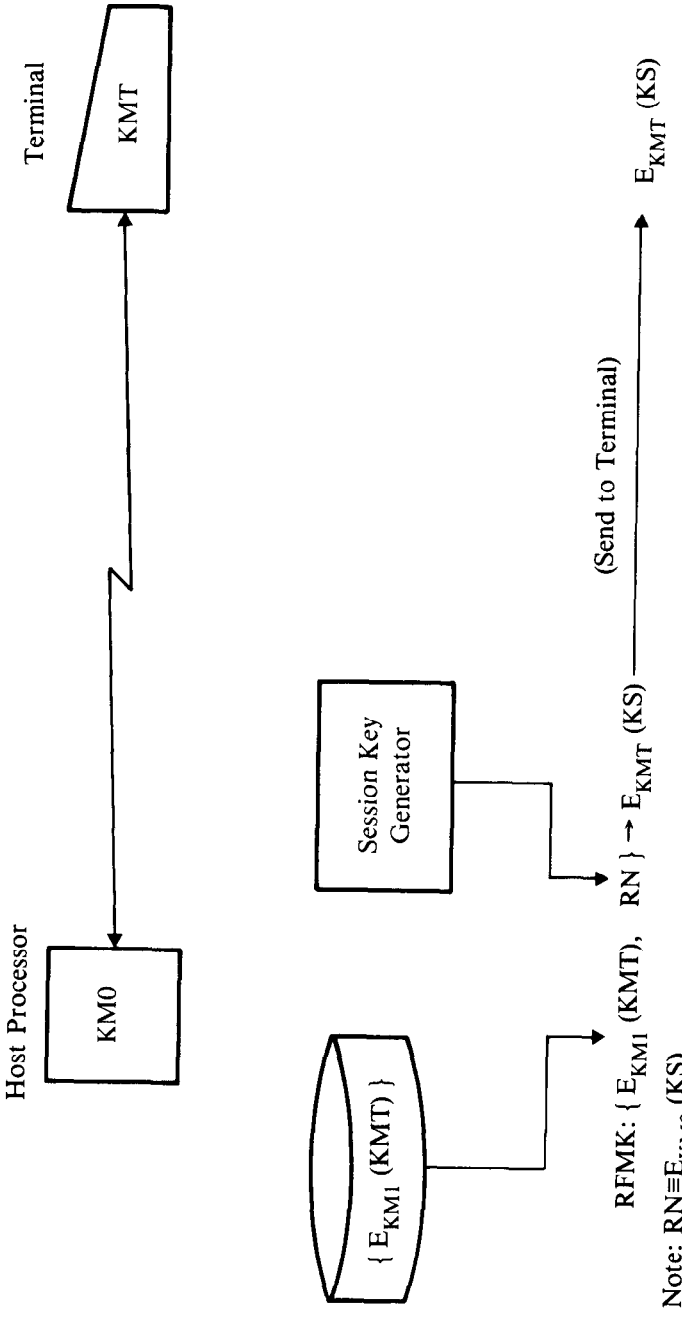


Figure 5-1. *Initiation of a Dynamically Generated Session Key (KS) Between a Terminal and Host Application Program in a Single Domain*

This is accomplished through the use of the RFMK operation, as described above. $E_{KMT}(KS)$ is then transmitted to the terminal, where KS is recovered and transferred to the working key storage in the terminal's cryptographic facility. $E_{KM0}(KS)$ is given to the application program in the host. At this point, both the terminal and the application program have identical session keys (KS) that can be used for ciphering data (Figure 5-1).

Requirements

In the approach described here, five basic cryptographic operations are needed at the host system: *set master key* (SMK), *encipher under master key* (EMK), *encipher data* (ECPH), *decipher data* (DCPH), and *reencipher from master key* (RFMK). The approach also requires two master keys: KM0 and KM1.

SINGLE-DOMAIN COMMUNICATION SECURITY AND FILE SECURITY USING DYNAMICALLY GENERATED PRIMARY KEYS

A cryptographic system with two master keys permits single-domain communication security using dynamically generated primary keys. If stored data must be protected as well, then a third master key is needed.

Problems Associated with Storing Enciphered Data

The previous section described a protocol for single-domain communication security using dynamic session keys. It therefore seems natural to ask if the same scheme could be adapted for use in file security to protect stored data.

Suppose one wishes to protect stored data in the same way that communicated data are protected, that is, one wishes to use a session key in the form $E_{KM0}(KS)$ as an input parameter to the *encipher data* and *decipher data* operations for the purpose of creating and recovering data files, respectively. However, for this approach to be workable (i.e., to be able to recover data with a DCPH operation), either $E_{KM0}(KS)$ must be saved for later use or else it must be possible to recreate it when it is needed (Figure 5-2).

If $E_{KM0}(KS)$ is stored within the system, especially for long periods, it must be protected by a suitable method of controlled access, since knowledge of $E_{KM0}(KS)$ would allow data to be recovered directly with a *decipher data* operation. This difficulty could be avoided, of course, by using the quantity $E_{KM0}(KS)$ as a personal key and therefore not storing it within the system. However, the advantage of a personal key must be weighed against that of cryptographic transparency (where the user is relieved of any responsibility for handling keys). When stored information is shared among many users, the use of personal keys may be impractical. For example, if there are 10 different users and 10 different data files, such that the first user must have access to all but the first data file, the second user must have access to all but the second file, and so forth, then each file must be enciphered under a different key and each user must be given nine different personal keys to manage.

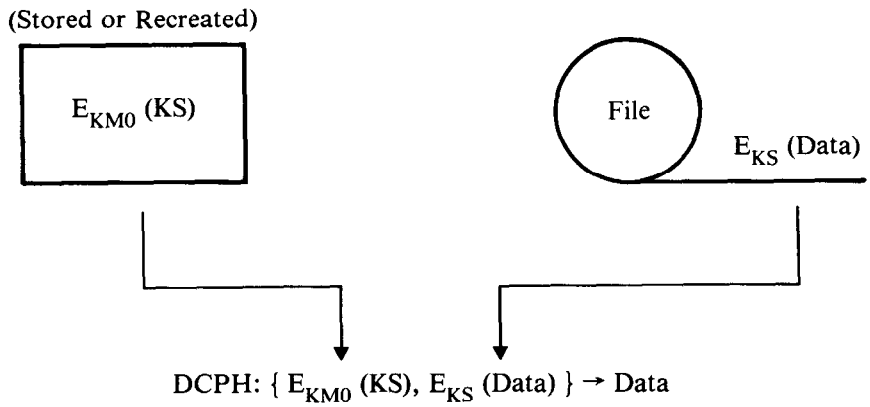


Figure 5-2. Recovery of Data Using the DCPH Operation

Whether $E_{KM0}(KS)$ is stored in the system or used as a personal key, key management must permit the host master key (KM0) to be changed periodically. Either there must be a method for recovering KS in clear form so that it can be reenciphered under the new host 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 session keys.

Still another disadvantage of basing a file recovery strategy on the stored quantity $E_{KM0}(KS)$ is that recovery at a different host system would not be possible unless KM0 was shared with that other host system. *But no good key management scheme would require a host master key to be shared with another host system.*

Because of these disadvantages, $E_{KM0}(KS)$ should not be the quantity saved for later use in file recovery operations. The disadvantages are overcome, however, if KS is stored enciphered under a *secondary file key* (KNF), rather than under KM0.

In the described approach, quantity $E_{KNF}(KS)$ is saved in a place that will be accessible during later file recovery operations, such as in the file header or in a separate file. At the host system, KNF is stored under the encipherment of some suitable key-encrypting key. (The choice of this key-encrypting key is discussed below.) Recovery of data is accomplished by reading $E_{KNF}(KS)$ from its saved location, obtaining access to the value of KNF stored at the host system, and regenerating the quantity $E_{KM0}(KS)$ using an appropriate translation operation.

The question now arises as to what key KNF can safely be enciphered under during its period of storage. Suppose KNF is stored enciphered under KM0. If it were, then a *decipher data* operation could be used to obtain a clear session key:

$$\text{DCPH: } \{E_{KM0}(KNF), E_{KNF}(KS)\} \longrightarrow KS$$

Again, this would violate the requirement that it must not be possible to recover keys in the clear outside the cryptographic facility.

If, on the other hand, KNF is stored enciphered under KM1, then the cryptographic operation that permits $E_{KM0}(KS)$ to be recovered from $E_{KM1}(KNF)$ and $E_{KNF}(KS)$ would also permit $E_{KM0}(KS)$ to be recovered from $E_{KM1}(KMT)$ and $E_{KMT}(KS)$. The said cryptographic operation would permit an attack against ciphered data by an opponent who could gain access to the system.

The attack is as follows. During a communication session between a user at a terminal and an application program in a host, the opponent obtains the quantities $E_{KMT}(KS)$ and $E_{KS}(Data)$ via a wiretap. By gaining access to the host system, the opponent first recovers $E_{KM0}(KS)$ by entering the quantities $E_{KM1}(KMT)$ and $E_{KMT}(KS)$ as inputs to the said cryptographic operation. Data is then recovered by entering the quantities $E_{KM0}(KS)$ and $E_{KS}(Data)$ as inputs to a DCPH operation.

Incorporation of file security within a cryptographic system, as described here, weakens the protection afforded by communication security. This is because the cryptographic operation that permits $E_{KM0}(KS)$ to be recovered from $E_{KNF}(KS)$ also permits $E_{KM0}(KS)$ to be recovered from $E_{KMT}(KS)$. Such a condition should be avoided; COMSEC and FILESEC applications should be cryptographically separated (see Partitioning of Cipher Keys, Chapter 4).

Three Master Keys

Enciphering the terminal master keys and secondary file keys under different host master keys permits the file security system to be implemented without affecting the integrity of the communication security system. The terminal master keys (which are secondary communication keys) are stored enciphered under a second variant of the host master key (KM1), and the secondary file keys are stored enciphered under a second variant of the host master key (KM2). KM2 is derived from KM0 in a manner similar to that used for deriving KM1: by inverting (different) selected bits in KM0. (A precise specification for KM2 is not important to the present discussion).

In file security, the primary key used for ciphering data is called a *file key* (KF). Let KF1, KF2, . . . , KFn denote the dynamically generated primary file keys used for ciphering stored data. It is assumed that KF is operational for the life of the enciphered file, that is, until the file is no longer maintained enciphered under KF.

The cryptographic operation that allows a file key (KF) to be transformed from encipherment under a secondary file key (KNF) to encipherment under the host master key (KM0), defined as the *reencipher to master key* (RTMK) operation, is given by

$$\text{RTMK: } \{E_{KM2}(KNF), E_{KNF}(KF)\} \longrightarrow E_{KM0}(KF)$$

Even though the relationship between the host master key and its first and second variants is publicly known, this information is not enough to permit

the algorithm to be broken. For instance, $E_{KM0}(KNF)$ and $E_{KM1}(KNF)$ cannot be deduced from $E_{KM2}(KNF)$ for a strong cryptographic algorithm (see Protection of Host Keys, Chapter 4).

The attack described above for recovering session keys is thwarted when KNF is stored enciphered under KM2. That is, it is not possible to enter $E_{KM2}(KNF)$ and $E_{KNF}(KF)$ as inputs to a *decipher data* operation and recover a clear file key (KF):

$$\text{DCPH: } \{E_{KM2}(KNF), E_{KNF}(KF)\} \longrightarrow D_K(E_{KNF}(KF)) \neq KF$$

because

$$K = D_{KM0}(E_{KM2}(KNF)) \neq KNF$$

In like manner, it is not possible to enter $E_{KM1}(KMT)$ and $E_{KMT}(KS)$ as inputs to a *reencipher to master key* operation and recover the quantity $E_{KM0}(KS)$:

$$\text{RTMK: } \{E_{KM1}(KMT), E_{KMT}(KS)\} \longrightarrow E_{KM0}(D_K(E_{KMT}(KS))) \neq E_{KM0}(KS)$$

because

$$K = D_{KM2}(E_{KM1}(KMT)) \neq KMT$$

Host Key Protection

Having justified the need for three master keys, the intended use of each of these keys is now summarized. KM0 is used to protect both file keys and session keys. Thus primary keys are maintained at the host system in the form

$$E_{KM0}(KS1), E_{KM0}(KS2), \ldots, E_{KM0}(KSn)$$
$$E_{KM0}(KF1), E_{KM0}(KF2), \ldots, E_{KM0}(KFn)$$

KM1 is used to protect terminal master keys. Terminal master keys are maintained at the host system in the form

$$E_{KM1}(KMT1), E_{KM1}(KMT2), \ldots, E_{KM1}(KMTn)$$

KM2 is used to protect secondary file keys. Secondary file keys are maintained at the host system in the form

$$E_{KM2}(KNF1), E_{KM2}(KNF2), \ldots, E_{KM2}(KNFn)$$

Encipherment under KM1 and KM2

Encipherment of the secondary file keys under KM1 is performed under secure conditions by a combination of the cryptographic operations already described (see Encipherment Under KM1). The host master key (KM0) is

first entered into the main storage of the host system, where the variant KM2 is derived by inverting appropriate bits in KM0. (It is assumed that KM0 has already been inserted into the master key storage of the host's cryptographic facility using an SMK operation.) The intermediate quantity $E_{KM1}(KM2)$ is then derived as follows:

$$\text{EMK: } \{KM1\} \longrightarrow E_{KM0}(KM1)$$

$$\text{ECPH: } \{E_{KM0}(KM1), KM2\} \longrightarrow E_{KM1}(KM2)$$

Through the use of the intermediate quantity $E_{KM1}(KM2)$, each secondary file key (KNF) is then enciphered under KM2 as follows:

$$\text{EMK: } \{KNF\} \longrightarrow E_{KM0}(KNF)$$

$$\downarrow$$

$$\text{RFMK: } \{E_{KM1}(KM2), E_{KM0}(KNF)\} \longrightarrow E_{KM2}(KNF)$$

Once the list of secondary file keys has been enciphered, all intermediate values used in the computation are erased from the host's main memory.

File Key Generation

To satisfy the condition that no clear key occurs outside the cryptographic facility, and yet avoid a requirement to generate KF directly within the cryptographic facility, an indirect method of generating file keys is adopted. A 64-bit pseudo-random number (RN) is generated and defined to be the file key enciphered under the secondary file key associated with the named file:

$$RN = E_{KNF}(KF)$$

The encrypted file keys required by the cryptographic system are therefore produced as a sequence of pseudo-random numbers

$$RN1, RN2, \ldots, RNn$$

where the ith pseudo-random number (RNi) corresponds to the ith encrypted file key (i.e., $RNi = E_{KNF}(KFi)$).

The rationale for defining RN as the file key (KF) enciphered under the secondary file key (KNF), rather than as the file key enciphered under the host master key (KM0), is as follows. If RN were to be defined as the quantity $E_{KM0}(KF)$, then the RFMK operation would have to be used at file creation to derive $E_{KNF}(KF)$ from $E_{KM0}(KF)$. Recall that $E_{KNF}(KF)$ is saved in a location that will be accessible during later file recovery operations, whereas $E_{KM0}(KF)$ is used with an ECPH operation to encipher the file. But at file recovery, the RTMK operation would have to be used to derive $E_{KM0}(KF)$ from $E_{KNF}(KF)$. Thus both the RFMK and RTMK operations would be

needed for file security. This in turn would require that KNF be stored enciphered under both KM1 and KM2. An undesirable reversibility would then exist between RFMK and RTMK.

In contrast, if RN is defined as the quantity $E_{KNF}(KF)$, then the RTMK operation can be used to derive $E_{KM0}(KF)$, both when the file is created and when the file is recovered, and KNF need only be stored enciphered under KM2.

An Example of File Encryption

The following example describes how a host application program can encipher a file to be stored on a secondary storage medium, and how this file can later be recovered. Recovery is at the same host where the file was originally enciphered. (Enciphering a file at one host and deciphering it at another host is discussed in Multiple-Domain Encryption.) A 64-bit pseudo-random number (RN) is generated at the host system and defined to be the file key (KF) enciphered under the secondary file key (KNF) of the named file (i.e., $RN = E_{KNF}(KF)$). In this example, RN is written in the file header. The RTMK operation is used by the key manager to transform RN into the quantity $E_{KM0}(KF)$. The quantity $E_{KM0}(KF)$ is then returned by the key manager to the host application program, whereupon the program uses it as an input parameter to the CIPHER (FNC = ENCPHR) macro instruction for the purpose of enciphering the file. During recovery, the quantity RN is read from the file header. The RTMK operation is used by the key manager to transform RN into the quantity $E_{KM0}(KF)$. Again, the quantity $E_{KM0}(KF)$ is returned by the key manager to the application program, whereupon the program uses it as an input parameter to the CIPHER (FNC = DECPHR) macro instruction for the purpose of deciphering the file. Figure 5-3 illustrates the procedures for creating and recovering a file.

Because of the level of indirection provided by the secondary file key (KNF), the quantity $E_{KNF}(KF)$ does not depend on the host master key. Therefore, changing the host master key will not require that $E_{KNF}(KF)$ be reenciphered, as would be the case if $E_{KM0}(KF)$ were written in the file header. The only change that must be made when a new host master key is installed is that the secondary keys stored at the host system are reenciphered under the new variants of the master key, KM1′ and KM2′ (where ′ indicates new). Moreover, since the secondary file key (KNF) can be disclosed to other host systems, recovery is also possible at remote locations.

If cryptographic transparency is desired, the quantity $E_{KNF}(KF)$ can be written in the header of the data file. Access to the data, in this case, can be controlled by controlling the use of the RETKEY macro and thereby controlling access to the quantity $E_{KM2}(KNF)$, which is stored at the host system. In an alternate approach, $E_{KNF}(KF)$ can be treated as a personal key and stored outside the system. In this case, access to the data requires also that the secret quantity $E_{KNF}(KF)$ be provided to the system at the time data are to be deciphered.

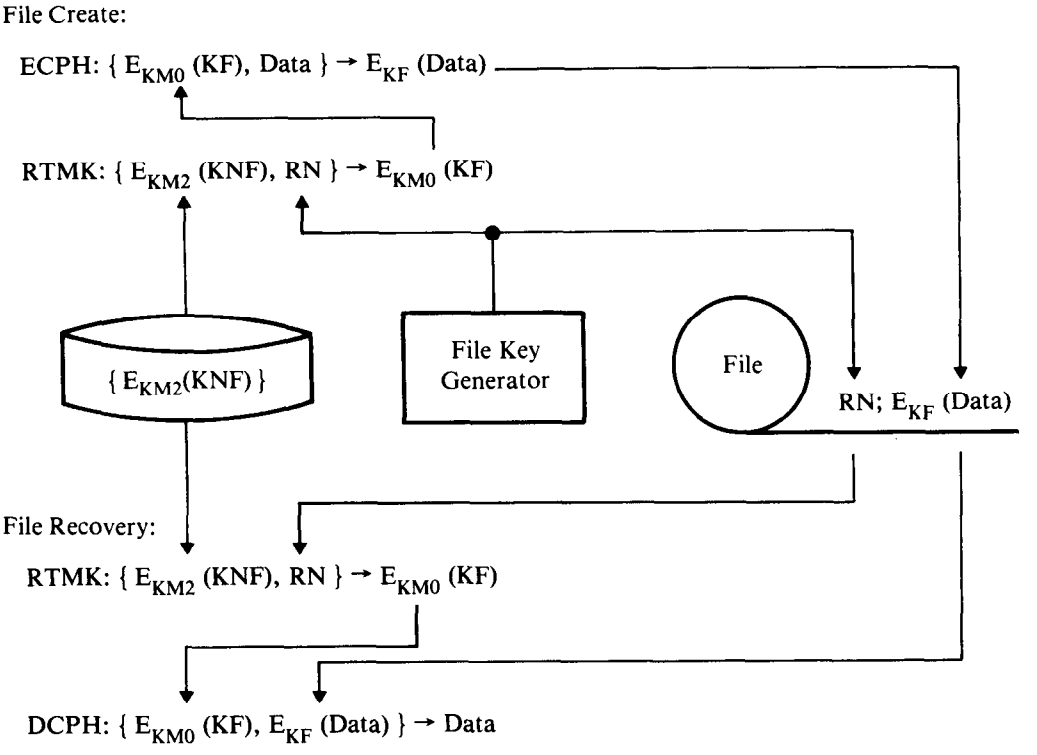


Figure 5-3. Single-Domain File Security Using a Dynamically Generated File Key (KF)

Requirements

In the approach described here, six basic cryptographic operations are needed at the host system: *set master key* (SMK), *encipher under master key* (EMK), *encipher data* (ECPH), *decipher data* (DCPH), *reencipher from master key* (RFMK), and *reencipher to master key* (RTMK). The approach also requires three master keys: KM0, KM1, and KM2.

MULTIPLE-DOMAIN ENCRYPTION

The set of basic cryptographic operations defined in the previous section permits communication security and file security to be achieved within a single-domain network, that is, in a communications network consisting of a single host node and one or more terminal nodes. In this section, it is shown that this same set of cryptographic operations is sufficient to achieve communication security and file security within a multidomain network, that is, in a communications network with many host nodes.

A Protocol For Communication Security

Let i and j denote host nodes whose master keys are KM0i and KM0j, respectively. To permit establishment of a common KS between domains i and j,[3] the two host systems must first share a common key. However, *the common key shared by each pair of host systems should not be the host master key of either system.* Instead, the host systems should share a special key that is 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 discussed, the following secondary communication keys are defined:

KNCii Known only by host node i; permits a session key generated at host node i to be established between two nodes within domain i. KNCjj is similarly defined at host node j.

KNCij Shared by host nodes i and j; permits a session key generated at host node i to be transmitted to and recovered at host node j. A similar key KNCji is available at host nodes j and i to permit a session key to be transmitted in the reverse direction.

Generally speaking, there should be only one KNCij key and one KNCji key, but possibly many KNCii and KNCjj keys. Within the domain of host node i, the secondary communication keys are used as a means for establishing a common KS between the host and one of its (n) terminals, or between two of its (n) terminals. Hence the set {KNCii} actually represents the set {KMTi1, KMTi2, . . . , KMTin}, where i refers to domain i and the numbers 1 through n denote the specific terminal.

At host node i, KNCij is stored enciphered under KM1i, thus allowing KS to be transmitted to host node j. At host node j, KNCij is stored enciphered under KM2j, thus allowing KS to be reenciphered under KM0j. A symmetrical specification also exists for KNCji. At host node j, KNCji is stored enciphered under KM1j, thus allowing KS to be transmitted to host node i. At host node i, KNCji is stored enciphered under KM2i, thus allowing KS to be reenciphered under KM0i. The form in which KNCij and KNCji are stored at host nodes i and j is shown in Figure 5-4.

The method for establishing a common session key (KS) between two domains, say domain i and domain j, is shown in Figure 5-5.

At host node i, a pseudo-random number (RN) is generated and defined as

$$RN = E_{KM0i}(KS)$$

RN can be used directly in the ECPH and DCPH operations for ciphering data or it can be used in an RFMK operation to transform KS from encipherment

[3] The collection of nodes consisting of host node i and all its logically connected terminal nodes is defined as domain i. A similar domain is defined for host node j.

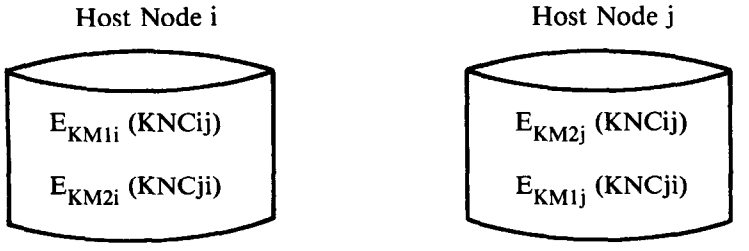


Figure 5-4. Storage of Secondary Communication Keys Shared by Domain i and Domain j

under KM0i to encipherment under a terminal master key (a key in the set {KMTi1, KMTi2, . . . , KMTin}). To send KS to domain j, the RFMK operation is used to transform KS from encipherment under KM0i to encipherment under the appropriate secondary communication key (KNCij):

$$\text{RFMK: } \{E_{KM1i}(KNCij), E_{KM0i}(KS)\} \longrightarrow E_{KNCij}(KS)$$

The quantity $E_{KNCij}(KS)$ is then transmitted to host node j, where the RTMK operation is used to transform KS from encipherment under KNCij to encipherment under KM0j:

$$\text{RTMK: } \{E_{KM2j}(KNCij), E_{KNCij}(KS)\} \longrightarrow E_{KM0j}(KS)$$

The quantity $E_{KM0j}(KS)$ can then be used directly in the ECPH and DCPH operations for ciphering data at host node j, or it can be used in an RFMK operation to transform KS from encipherment under KM0j to encipherment under a terminal master key (a key in the set {KMTj1, KMTj2, . . . , KMTjn}).

Because the RFMK and RTMK operations are designed to use, respectively, only the first and second variants of the host master key, a unidirectional process of transformation involving secondary keys (secondary communication keys in the present example) is achieved. A unidirectional transformation process is one that is irreversible at the sender's location, that is, recovery of the primary key (session key in the present example) can only be done at a predefined receiver.

In the example above (of host-to-host communication), unidirectionality from the first host to the second host is achieved because the secondary communication key (KNCij) is known to the first host only in the form $E_{KM1i}(KNCij)$, and the output of the sender's RFMK operation, $E_{KNCij}(KS)$, is usable only by the intended receiver's RTMK operation. In other words, the first host can produce $E_{KNCij}(KS)$ from $E_{KM0i}(KS)$ because $E_{KM1i}(KNCij)$ is available, but it cannot retrieve $E_{KM0i}(KS)$ from $E_{KNCij}(KS)$ because $E_{KM2i}(KNCij)$ is not available. Conversely, the second host can retrieve $E_{KM0j}(KS)$ from $E_{KNCij}(KS)$ because $E_{KM2j}(KNCij)$ is available, but it cannot produce $E_{KNCij}(KS)$ from $E_{KM0j}(KS)$ because $E_{KM1j}(KNCij)$ is not available.

Thus the unidirectionality property ensures that an opponent who recovers

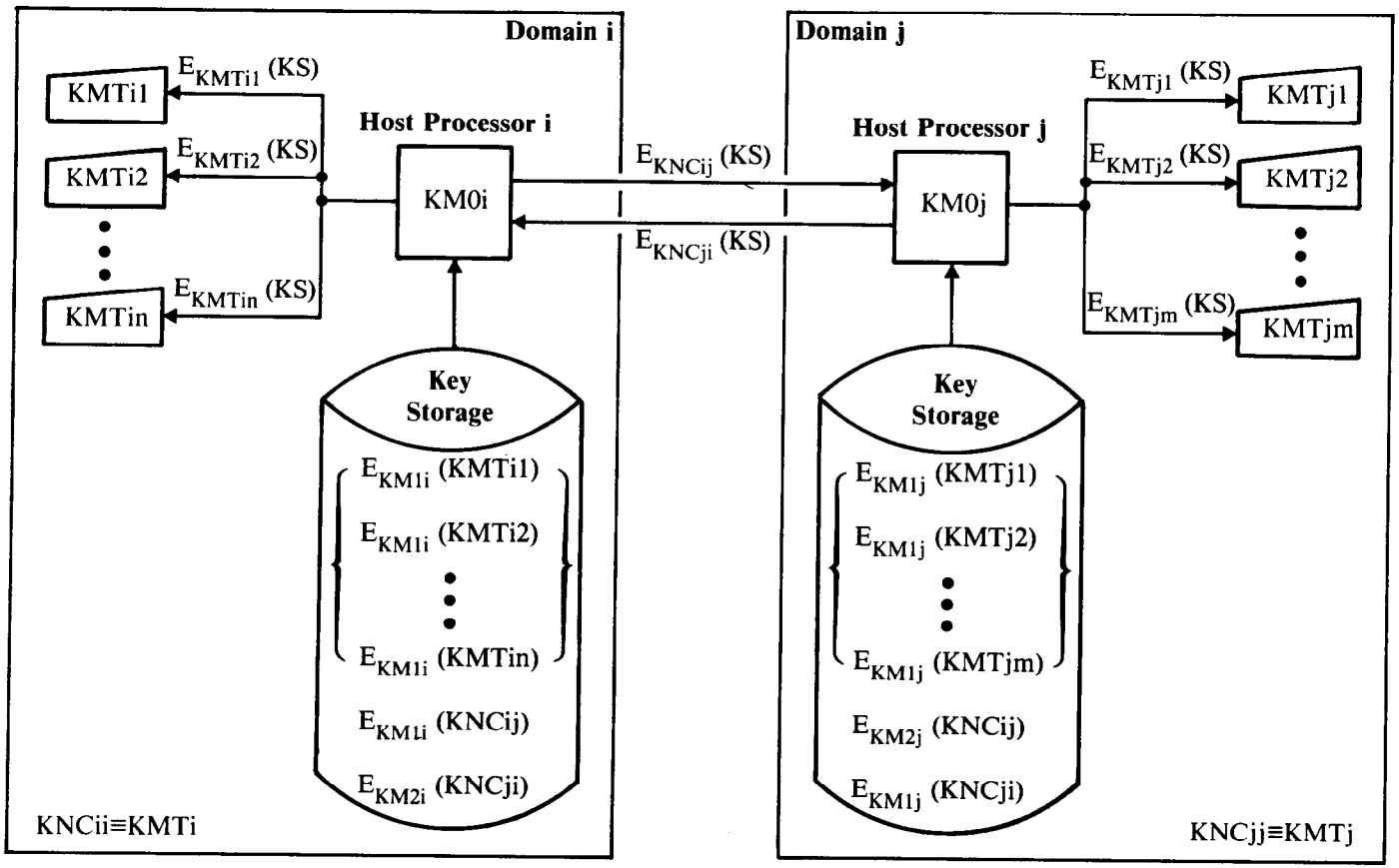


Figure 5-5. Secondary Communication Key Protocol Between Domain i and Domain j

the quantity $E_{KNCij}(KS)$ via a wiretap cannot make use of this quantity at host node i. At host node j, however, the same protocol that permits host node j to recover $E_{KM0j}(KS)$ from $E_{KNCij}(KS)$ can potentially be misused by an opponent. For this reason, the RTMK operation should be privileged, and its use controlled by the key manager.

In summary, then, when a secondary communication key is used to transmit a session key from one domain to another, recovery of that session key at the destination host must be controlled by means other than cryptography.

A Protocol For File Security

To permit establishment of a common KF between domains i and j, the host systems must first share a common key. The cryptographic key used for this purpose is called a secondary file key (KNF).

In the protocol discussed, the following secondary file keys are defined:

KNFii Known only to host node i; permits a file key generated at host node i to be recovered at host node i. KNFjj is similarly defined at host node j.

KNFij Shared by host nodes i and j; permits a file key generated at host node i to be safely stored and later recovered at host node j. A similar key KNFji is available at host nodes j and i to permit a file key to be transmitted in the reverse direction.

At host node i, KNFij is stored enciphered under KM1i, thus allowing KF to be transmitted to host node j. At host node j, KNFij is stored enciphered under KM2j, thus allowing KF to be reenciphered under KM0j. Conversely, at host node j, KNFji is stored enciphered under KM1j, thus allowing KF to be transmitted to host node i. And at host node i, KNFji is stored enciphered under KM2i, thus allowing KF to be reenciphered under KM0i. Note the similarity between KNCij and KNFij, and between KNCji and KNFji. The form in which KNFij and KNFji are stored at host nodes i and j is shown in Figure 5-6.

Transporting a New File

The method for generating a file key (KF) at one domain (say domain i) which can be recovered at another domain (say domain j) is shown in Figure 5-7.

At host node i, a pseudo-random number (RN) is generated and defined as

$$RN = E_{KM0i}(KF)$$

RN can be used directly in the ECPH operation to encipher the file, as follows:

$$ECPH: \{RN, Data\} \longrightarrow E_{KF}(Data)$$

To send KF to domain j, the RFMK operation is first used to transform KF

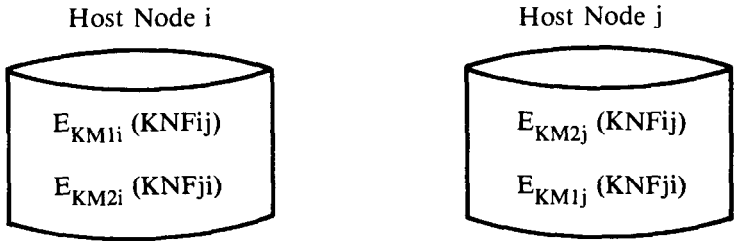


Figure 5-6. Storage of Secondary File Keys Shared by Domain i and Domain j

from encipherment under KM0i to encipherment under the secondary file key (KNFij):

$$\text{RFMK: } \{E_{KM1i}(KNFij), E_{KM0i}(KF)\} \longrightarrow E_{KNFij}(KF)$$

The quantity $E_{KNFij}(KF)$ is, for this example, recorded in the file header. The enciphered file is then sent to host node j. At host node j, the RTMK operation is used to transform KF from encipherment under KNFij to encipherment under KM0j:

$$\text{RTMK: } \{E_{KM2j}(KNFij), E_{KNFij}(KF)\} \longrightarrow E_{KM0j}(KF)$$

The quantity $E_{KM0j}(KF)$ can then be used directly in the DCPH operation to recover data:

$$\text{DCPH: } \{E_{KM0j}(KF), E_{KF}(Data)\} \longrightarrow Data$$

Transporting an Existing File

The protocol described here has the advantage that a file created and intended to be recovered at host node i can easily be sent to and recovered at host node j *without the data having to be deciphered and reenciphered under a new KF.* This is accomplished by using an RTMK operation to recover $E_{KM0i}(KF)$ from $E_{KNFii}(KF)$, and then using an RFMK operation to produce $E_{KNFij}(KF)$ from $E_{KM0i}(KF)$:

$$\text{RTMK: } \{E_{KM2i}(KNFii), E_{KNFii}(KF)\} \longrightarrow E_{KM0i}(KF)$$

$$\text{RFMK: } \{E_{KM1i}(KNFij), E_{KM0i}(KF)\} \longrightarrow E_{KNFij}(KF)$$

Hence the file can be sent to host node j by merely replacing $E_{KNFii}(KF)$ with $E_{KNFij}(KF)$ in the file header. (Note that the procedure may require the file to be copied to another volume.)

Again, because of the property of unidirectionality (see A Protocol for

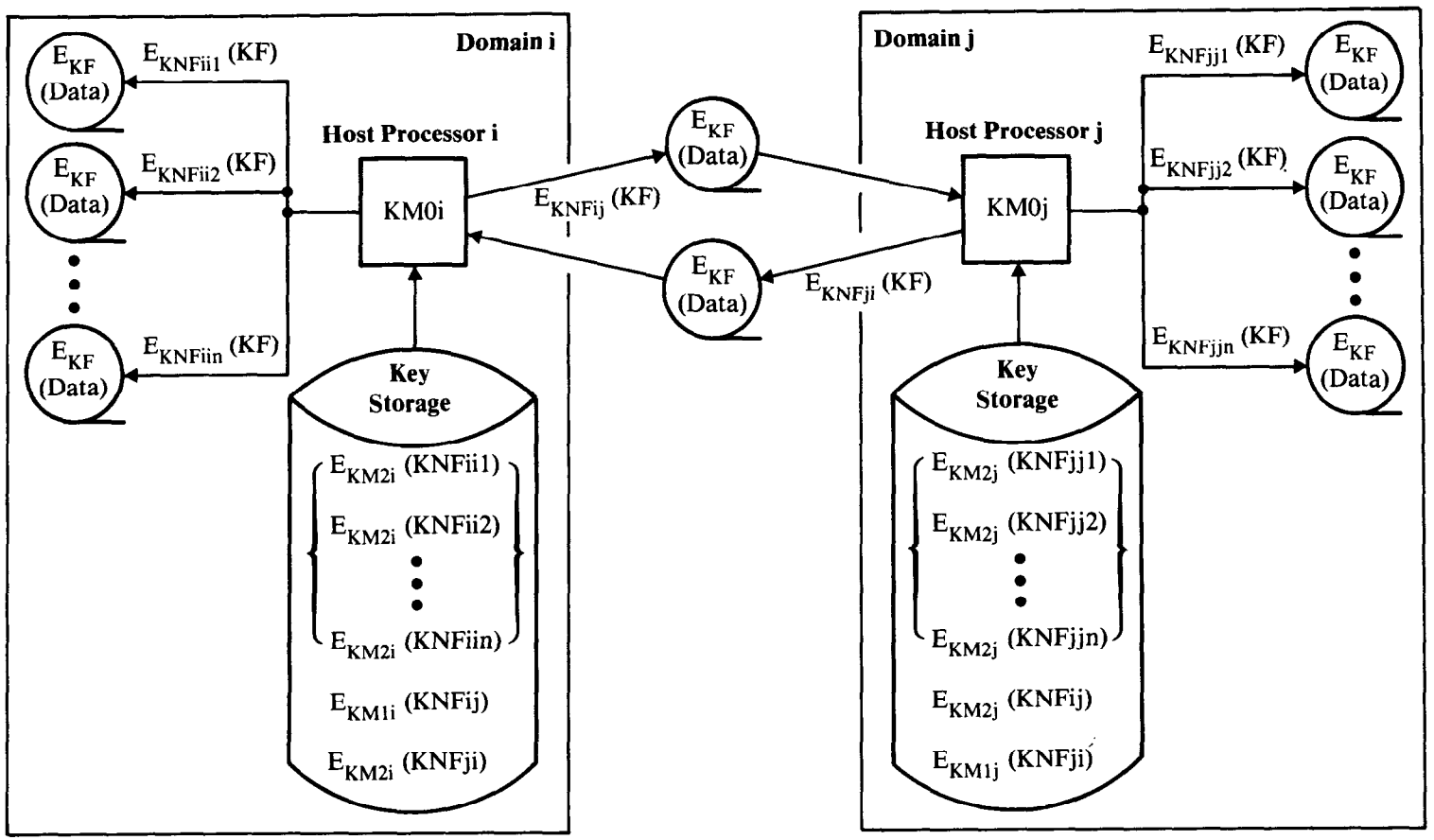


Figure 5-7. Secondary File Key Protocol Between Domain i and Domain j

Communication Security), an opponent who obtains the quantity $E_{KNFij}(KF)$ from the file header can make no use of it at host node i. However, at host node j the RTMK operation could be used to recover $E_{KM0j}(KF)$ from $E_{KNFij}(KF)$, provided that access to the system is obtained. Hence the RTMK operation should be privileged, and its use controlled by the key manager.

ADDITIONAL CONSIDERATIONS

It is assumed that only installation-specified cryptographic keys are stored enciphered under KM1 or KM2, and that these keys are secret and known only to authorized installation personnel.

If an opponent could encipher under the host master key variants (KM1 and KM2), then it might be possible to replace existing installation-specified keys with those selected by the opponent. For example, to attack a terminal an opponent could do the following: encipher an alien terminal master key (KMT′) under KM1, replace KMT with KMT′ at the terminal, and replace $E_{KM1}(KMT)$ with $E_{KM1}(KMT')$ in the host's key table. The opponent could then recover session keys sent to the terminal in the form $E_{KMT'}(KS)$, and thus recover data sent to or from the terminal in the form $E_{KS}(Data)$.

Variations of the above attack include the following: requesting or causing installation personnel to encipher, under KM1 or KM2, a value which the opponent knows, or compromising the security of the system by obtaining $E_{KM1}(KMT)$ from the host's key table, and KMT from the terminal.

If an opponent can encipher a known value X under KM1, or can cause installation personnel to encipher a known value X under KM1, or can discover the value of an unknown quantity enciphered under KM1, then an attack can recover primary keys in clear form. Let X, $E_{KM1}(X)$, and $E_{KM0}(K)$ be the values known to an opponent, where K is the unknown primary key to be recovered. The operations involved in the attack are shown below.[4]

$$\text{RFMK: } \{E_{KM1}(X), E_{KM0}(K)\} \longrightarrow E_X(K)$$

$$\text{EMK: } \{X\} \longrightarrow E_{KM0}(X)$$

$$\text{DCPH: } \{E_{KM0}(X), E_X(K)\} \longrightarrow K$$

Elements in the set $\{E_{KMx}(KMy) : x, y = 0, 1, 2; x = y \neq 2\}$ must be kept secret from any potential opponent, since they could be used as parameters with the available cryptographic operations to encipher and/or decipher under one or more of the keys: KM0, KM1, and KM2, as indicated in Table 5-2.

The quantities $E_{KM0}(KM0)$, $E_{KM0}(KM1)$, and $E_{KM0}(KM2)$ could be used with an ECPH operation to encipher under KM0, KM1, and KM2, respectively,

[4] In a sense, the attack is academic because knowledge of $E_{KM0}(K)$ (where K = KS) would allow data enciphered under K to be recovered with the ECPH operation.

Function	Operation	Key Used For Ciphering KM0	KM1	KM2
Encipher	ECPH	$E_{KM0}(KM0)$	$E_{KM0}(KM1)$	$E_{KM0}(KM2)$
Decipher	DCPH	$E_{KM0}(KM0)$	$E_{KM0}(KM1)$	$E_{KM0}(KM2)$
Encipher	RFMK	$E_{KM1}(KM0)$	$E_{KM1}(KM1)$	$E_{KM1}(KM2)$
Decipher	RTMK	———	$E_{KM2}(KM1)$	$E_{KM2}(KM2)$

Table 5-2. Special Quantities that Permit Encipherment and Decipherment Under KM0, KM1, and KM2

or they could be used with a DCPH operation to decipher under KM0, KM1, and KM2, respectively. For example, an arbitrary quantity X could be enciphered and deciphered under KM1 as follows:

$$\text{ECPH: } \{E_{KM0}(KM1), X\} \longrightarrow E_{KM1}(X)$$
$$\text{DCPH: } \{E_{KM0}(KM1), X\} \longrightarrow D_{KM1}(X)$$

The quantities $E_{KM1}(KM0)$, $E_{KM1}(KM1)$, and $E_{KM1}(KM2)$ could be used with an RFMK operation to encipher quantity X under KM0, KM1, and KM2, respectively:

$$\text{EMK: } \{X\} \longrightarrow E_{KM0}(X)$$
$$\text{RFMK: } \{E_{KM1}(KM1), E_{KM0}(X)\} \longrightarrow E_{KM1}(X)$$

Finally, the quantities $E_{KM2}(KM1)$ and $E_{KM2}(KM2)$ could be used with an RTMK operation to decipher quantity X under KM1 and KM2, respectively, provided that decipherment under KM0 is also available:

$$\text{RTMK: } \{E_{KM2}(KM1), X\} \longrightarrow E_{KM0}(D_{KM1}(X))$$

$D_{KM1}(X)$ is recovered by deciphering $E_{KM0}(D_{KM1}(X))$ under KM0.

EXTENDED CRYPTOGRAPHIC OPERATIONS

Additional cryptographic operations can be defined to satisfy specific security requirements. In each case, the primitive operations of encipher and decipher act as building blocks to achieve the desired goal. Similarly, additional variants of the host master key can be defined to isolate further the desired functions served by a particular operation. In this manner, the property of

irreversibility is built-in, thus preventing an opponent from manipulating an operation to reverse and defeat its intended purpose.

Although the opportunity for designing special purpose cryptographic operations is virtually infinite, in the interest of economy only one example will be presented here—a method using composite keys to effect session key distribution. Through the use of a unique variant of the host master key, the technique, applicable to COMSEC applications, reduces the system's dependency on the RTMK operation. A second example extends the idea of additional master key variants to produce test patterns used in cryptographic authentication (see Implementing AF and AR, Chapter 8).

Cryptographic Key Distribution Using Composite Keys[5]

The key management scheme discussed thus far (Chapters 4 and 5) distinguishes between primary keys and secondary keys. The latter are used to encipher other keys and are defined as part of the process of initializing the cryptographic system. Primary keys, on the other hand, are dynamically generated each time a communications session is established and are referred to as session keys (KS). Session keys remain in existence only for the duration of the communications session, which is usually a relatively short time.

A simple method of establishing a session key between any two nodes within a communications system is to generate the key at one node and send it to the other. The key can be protected by enciphering it under a special secondary key that is unique to only these two nodes and is installed in advance. A retrieve function (e.g., the RTMK operation) must be used at the receiving node to transform the session key into a form suitable for enciphering and deciphering data. Thus someone who intercepts both an enciphered key and data enciphered under that key, and who also can gain access at any later time to the session key retrieve function, can transform KS into the form suitable for deciphering data.

In network nodes which provide a programmed interface to their respective cryptographic facility, use of the retrieve function (RTMK) can be controlled through physical means (denying access to the system to all but authorized users), and through logical means (making the designated function privileged and/or implementing an access control mechanism).

A *fixed function* node—one that does not provide user access to the cryptographic facility—already provides a level of protection in this regard. However, even here cryptographic authentication (see Handshaking, Chapter 8) could be required if the retrieve function is exposed to a "midnight" attack.

However, a different and more secure approach for protecting the retrieve function is achieved if composite keys are supported by the key distribution process. This is accomplished by defining the session key to be a composite of random data supplied by each node; for example, by combining a first

[5]© 1978 IEEE. Reprinted from *NTC 78 Conference Record,* December 3–6, 1978, Birmingham, Alabama [2].

random number (RN1) generated at the initiating node with a second random number (RN2) generated at the receiving node:

$$KS = RN1 \oplus RN2,$$

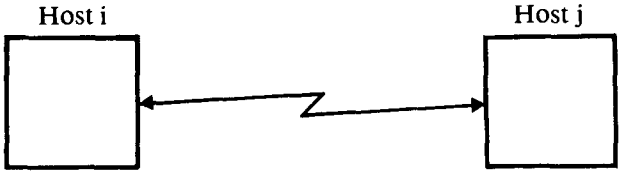
where $\oplus$ represents modulo 2 addition.[6]

The composite key protocol is such that interception of the enciphered values of RN1 and RN2 will not allow KS to be recovered in a form suitable for deciphering data *at either node*. However, the protocol for composite keys is more complex.

A Composite Key Protocol

A composite key protocol can be used to distribute keys between any two nodes in a communications network. However, because it is the intent here only to illustrate the concept of composite keys, the discussion will be limited to an example of host-to-host communications (Figure 5-8). In this example, an application program resident in one host (host i) requests a communications session with an application program resident in another host (host j). The composite key protocol is described as follows. At host i, a *generate session key 1* (GSK1) operation is used to

1. Generate a random number, RN1 (Figure 5-8).



1. Generate RN1.
2. Store RN1 at host i and send RN1 to host j.

RN1*

3. Generate RN2.
4. Combine RN2 with RN1 to produce KS.
5. Send RN2 to Host i.

RN2*

6. Retrieve RN1 and combine with RN2 to produce KS.
7. Erase stored copy of RN1.

*Protected by encryption during transmission

Figure 5-8. Overview: Composite Key Protocol

[6] Modulo 2 addition has been chosen for illustrative purposes only. Other suitable techniques for combining RN1 and RN2 exist.

2. Encipher RN1 under KNCij, which permits RN1 to be safely transmitted to host j (Figure 5-8).
3. Encipher RN1 under KM3i, which permits RN1 to be safely stored at host i (Figure 5-9).

A third variant of the host master key (KM3) is used to protect RN1 specifically so that no other parameter (except one enciphered under KM3) can be used in any meaningful way to produce KS. In effect, this isolates and protects the composite key functions from other functions in the cryptographic system. A fourth variant of the host master key (KM4) is defined for the purpose of generating pseudo-random numbers (see An Approach for Generating Keys with the Cryptographic Facility, Chapter 6).

At host j, a *generate session key 2* (GSK2) operation is used to

1. Generate a random number, RN2.
2. Encipher RN2 under KNCji, which permits RN2 to be safely transmitted to host i.
3. Merge RN2 with the value of RN1 received from host i.

Generate Session Key 1 (GSK1):

GSK1: $\{ E_{KM1i}(KNCij) \} \rightarrow E_{KNCij}(RN1), E_{KM3i}(RN1)$

where i designates the originating host of RN1.

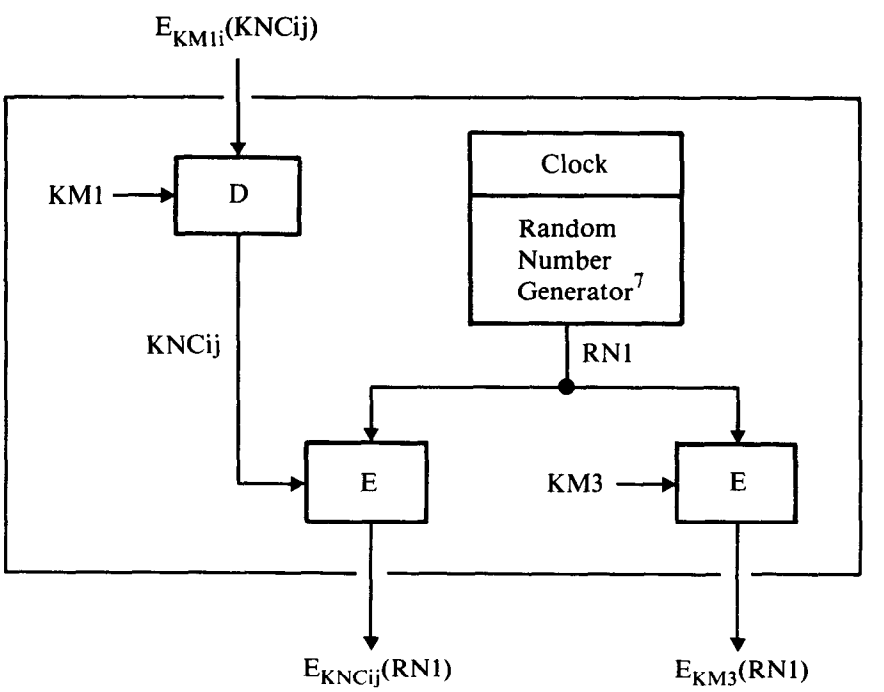


Figure 5-9. Generate Session Key 1

4. Encipher this merged value (KS) under KM0j, which is then in a form usable by an application program at host j for ciphering data.

(See Figure 5-10 and steps 3, 4, and 5 in Figure 5-8). The KS assigned to a particular session cannot be recreated even if someone obtains $E_{KNCij}(RN1)$ and executes GSK2. At each execution of GSK2 a nonresettable clock is read and is used as part of the random number generation process to create RN2.[7] Since KS is a function of RN2, and RN2 changes with each execution of GSK2, the ability to recreate KS is denied.

At host i, a *merge key* (MGK) operation is used to

1. Merge the saved copy of RN1 with the value of RN2 received from host j.
2. Encipher this merged value (KS) under KM0i, which is then in a form usable by an application program at host i for ciphering data.

Generate Session Key 2 (GSK2):

GSK2: { $E_{KM2j}(KNCij)$, $E_{KNCij}(RN1)$, $E_{KM1j}(KNCji)$ } → $E_{KM0j}(KS)$, $E_{KNCji}(RN2)$

where i designates the originating host of RN1 and j, the originating host of RN2.

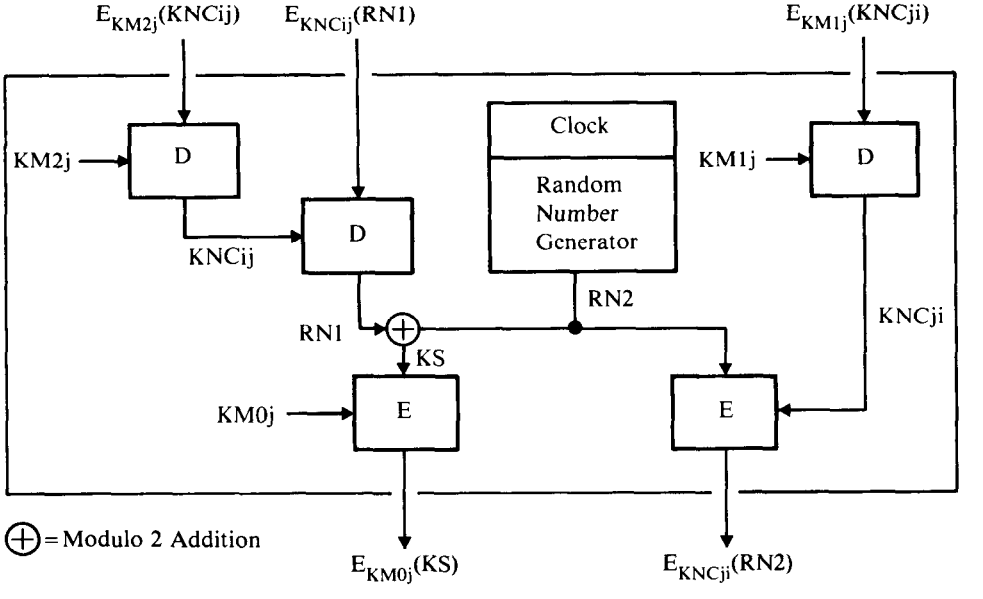


Figure 5-10. Generate Session Key 2

[7] An alternate approach would be to use a nonresettable counter as input to the random number generator and increment it at each GSK2 execution.

Merge Key (MGK):

MGK: { E_{KM2i}(KNCji), E_{KNCji}(RN2), E_{KM3i}(RN1) } → E_{KM0i}(KS)

where i designates the originating host of RN1 and j, the originating host of RN2.

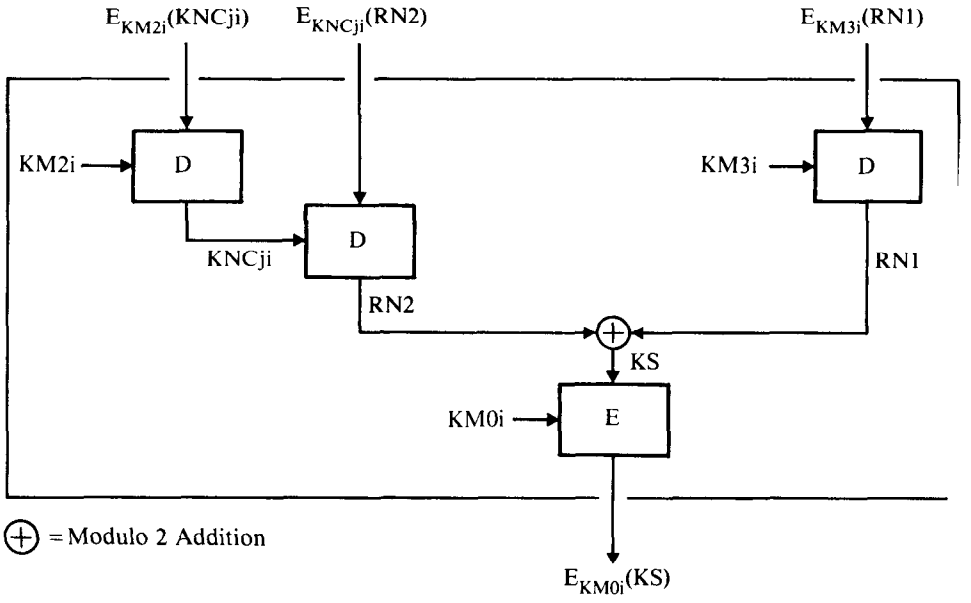


Figure 5-11. Merge Key

(See Figure 5-11 and steps 6 and 7 in Figure 5-8.)

Once the MGK operation is completed, E_{KM0i}(KS) is assigned to the application program, E_{KM3i}(RN1) is erased from host i's main storage, and E_{KM0i}(KS) is returned to the application program. For someone to gain any advantage from the MGK operation, the value E_{KM3i}(RN1) would have to be obtained during its brief period of storage at host i.

Although the present example discusses session initiation between two application programs, it can be coupled with the procedure specified for session key distribution in a single domain network (see Single-Domain Communication Security Using Dynamically Generated Primary Keys) so that a composite key (KS) produced at a host could then be sent to one of its terminals. In this case, E_{KM0i}(KS) is provided as input to an RFMK operation, and KS is transformed from encipherment under the host master key to encipherment under the terminal's master key (KMT):

$$\text{RFMK: } \{E_{KM1i}(KMT), E_{KM0i}(KS)\} \longrightarrow E_{KMT}(KS)$$

The quantity E_{KMT}(KS) is then sent to the terminal where KS is recovered and used for ciphering data. While this example illustrates a mixture of

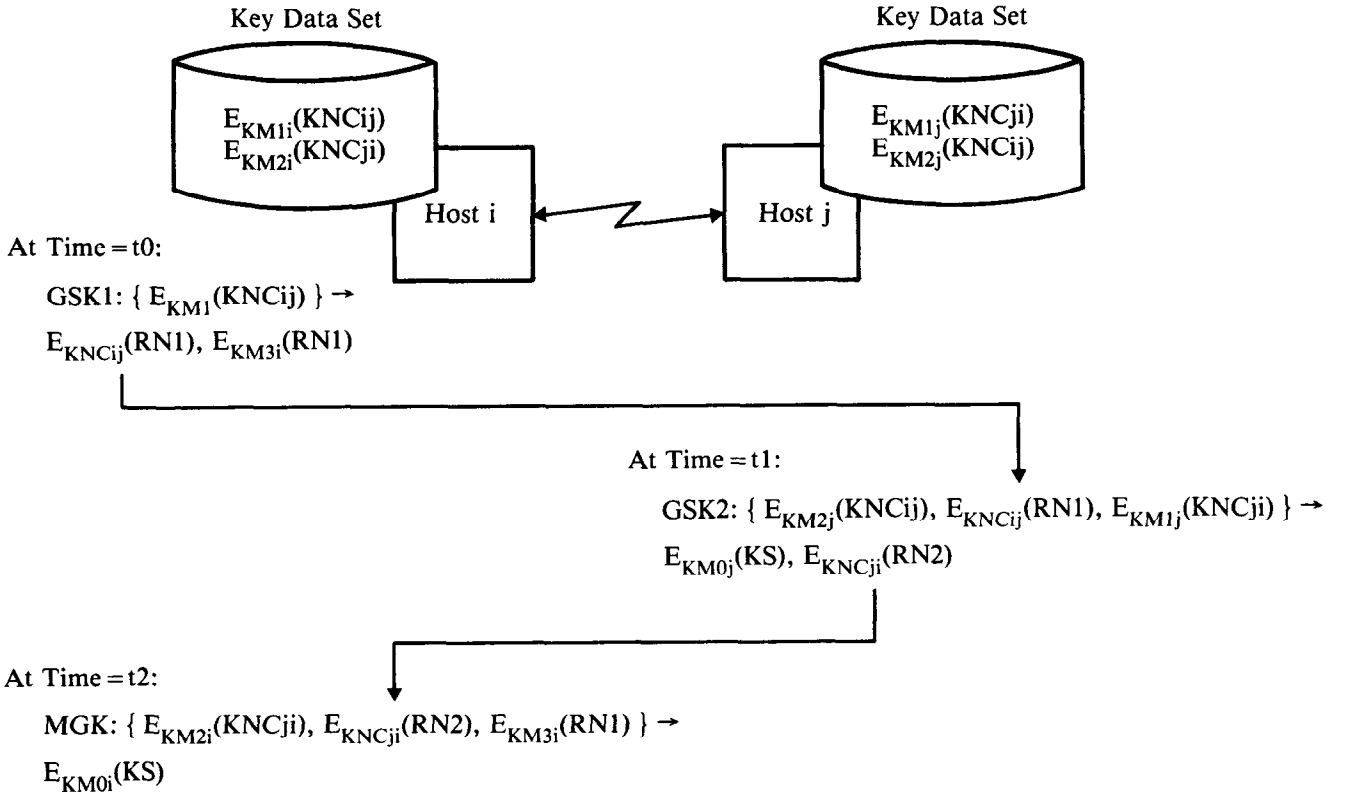


Figure 5-12. *Interdomain Exchange of KS*

protocols for session initiation, it should be obvious that the composite key protocol could be extended to include terminals.

To implement a composite key protocol, the cryptographic facilities at the affected nodes would have to support the operations GSK1, GSK2, and MGK, and the communications architecture would have to permit the exchange of RN1 and RN2. One such architecture, and the protocol it uses to send session keys between two nodes in a network, is described later (see Incorporation of Cryptography into a Communications Architecture, Chapter 7). Support for composite keys in this architecture could be accomplished by either modifying existing commands or creating new commands. Figure 5-12 illustrates host-to-host key distribution using composite keys.

Without composite keys, a session key could be retrieved and used without authorization provided that the enciphered value of that key and access to the receiving node is obtained. The protocol described here prevents this attack. Thus any attempt to bypass cryptographic protection must be made at the system level during the active session period and involve the retention of the session key in its final enciphered form or be directed at the clear data itself. For all practical purposes, then, no advantage is gained by intercepting RN1 and/or RN2.

SUMMARY

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, data is exposed through such external attacks as wiretapping and theft of removable storage. When cryptography is used, the opponent is forced to attack the system from within, requiring a penetration of the operating system.

Control over the execution of the defined cryptographic operations can be exercised by the following means: 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; and making selected cryptographic operations privileged.

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

REFERENCES

1. Ehrsam, W. F., Matyas, S. M., Meyer, C. H., and Tuchman, W. L., "A Cryptographic Key Management Scheme for Implementing the Data Encryption Standard (DES)," *IBM System Journal*, **17**, No. 2, 106–125 (1978).
2. Lennon, R. E. and Matyas, S. M., "Cryptographic Key Distribution Using Composite Keys," *NTC 78 Conference Record*, **2**, 26.1.1–26.1.6 (December 1978).