Communication Security and File Security 192
NETWORK ENCRYPTION MODES 195
Figure 4-1. Data Processing Network 194
Figure 4-2. Link Encryption 195
Figure 4-3. Node Encryption 196
Figure 4-5. Message and Header Encryption 200
FUNDAMENTALS OF LINK ENCRYPTION ... 201
Figure 4-6a. DTE/DCE and Link Control..... 202
Figure 4-6b. DEE Placement-Link Encryption..... 202
Asynchronous 203
Byte-Synchronous..... 204
Figure 4-7. Example of Extent of Encryption, ... 205
Bit-Synchronous 206
AN OVERVIEW OF END-TO-END 206
Cryptographic Key Data Set 208
CIPHER KEY ALLOCATION 208
Specification of Cipher Keys 209
Figure 4-8. Terminal End User 209
Figure 4-9. Host End Users 210
Figure 4-10. Cryptographic Facility-General..... 211
Figure 4-11. Transmission of Enciphered..... 212
Figure 4-12. Shorthand Notation Representin... 213
Figure 4-13. Shorthand Notation Representin... 213
Figure 4-15. Communication Phase..... 214
Figure 4-16. Allocation of Secondary Keys 215
Figure 4-17. Allocation of Secondary 216
Figure 4-18. Allocation of Secondary 217
Figure 4-19. Allocation of Secondary File..... 218
Figure 4-20. Allocation of Secondary File..... 218
An Example of the Encryption of Transmitted..... 219
Figure 4-21. Initial Configuration 220
Figure 4-22. Session Key Generation/Encrypt .. 220
Figure 4-23. Session Key Transformation-Hos . 221
Figure 4-24. Session Key Recovery at..... 221
Figure 4-25. Initial Configuration 222
An Example of the Encryption of a Data File 222
THE CRYPTOGRAPHIC FACILITY 222
Figure 4-26. File Key Generation and Encipher..... 223
Figure 4-27. File Key Recovery and Decipher 223
Figure 4-28. Cryptographic Facility 224

Figure 4-29. Primitive Operations of 225
CIPHER KEY PROTECTION 226
Protection of Terminal Keys 226
Figure 4-30. Session Key Recovery at the 227
Multiple Master Keys 229
Protection of Host Keys 228
The Master Key Concept 228
Encrypted vs. Unencrypted Primary Keys 228
Master Key Variants 230
Figure 4-31. An Implementation Using 230
Figure 4-32. Implementation in which 231
Figure 4-33. Derivation of Variants within 232
Hierarchy of Cipher Keys 232
Figure 4-34. Host Cipher Key Protection-Su 233
Figure 4-35. Hierarchy of Key Protection 234
THE HOST CRYPTOGRAPHIC SYSTEMI 234
Figure 4-36. Summary of Cipher Keys 235
BASIC CRYPTOGRAPHIC OPERATIONS 237
Figure 4-37. Host Cryptographic System 237
Cryptographic Operations at a Terminal 239
LOAD KEY DIRECT 239
WRITE MASTER KEY 239
Figure 4-38. Load Key Direct Operation at 240
DECIPHER KEY 240
ENCIPHER 240
Figure 4-39. Write Master Key Operation at 241
DECIPHER 241
Cryptographic Operations at a Host 243
Figure 4-40. Decipher Key Operation at 242
Figure 4-41. Encipher Operation at Terminal 243
Figure 4-42. Decipher Operation at Terminal 244
ENCIPHER DATA 244
Figure 4-43. Encipher Data Operation at 245
DECIPHER DATA 245
Key Management Operations 246
SET MASTER KEY 246
ENCIPHER UNDER MASTER KEY 246
Figure 4-45. Set Master Key Operation at Host 247
Figure 4-46. Encipher Under Master Key 247
Figure 4-44. Decipher Data Operation at 246
REENCIPHER FROM MASTER KEY 248
REENCIPHER TO MASTER KEY 248

Partitioning of Cipher Keys..... 250
Figure 4-47. Reencipher From Master Key 248
Figure 4-48. Reencipher To Master Key 249
Figure 4-49. Encipherment and Decipherment.. 251
Figure 4-50. Hypothetical Scheme for the..... 252
Figure 4-51. Correct and Incorrect Use of..... 252
CIPHER MACRO INSTRUCTION..... 253
Figure 4-52. Ciphering Operation Using the 255
Figure 4-53. Ciphering Operation,Using the 256
Figure 4-54. Ciphering Operation Using the 258
Figure 4-55. Procedure for Computing OCV 258
Figure 4-56. Padding of Short Blocks 259
Figure 4-57. Ciphering a Short Block Using 259
Key Parity..... 249
Partitioning of Cipher Keys..... 250
Figure 4-49. Encipherment and Decipherment.. 251
Figure 4-50. Hypothetical Scheme for the..... 252
Figure 4-51. Correct and Incorrect Use of..... 252
CIPHER MACRO INSTRUCTION..... 253
Figure 4-52. Ciphering Operation Using the 255
Figure 4-53. Ciphering Operation,Using the 256
Figure 4-54. Ciphering Operation Using the 258
Figure 4-55. Procedure for Computing OCV 258
Figure 4-56. Padding of Short Blocks 259
Figure 4-57. Ciphering a Short Block Using 259
KEY MANAGEMENT MACRO INSTRUCTIO .. 260
GENKEY and RETKEY Macros 260
Figure 4-60. Session Key Translation at Host j..... 267
Table 4-1. Resources, Keys, and Key Storage 262
Table 4-2. Valid and Invalid Parameter..... 263
Table 4-3. Valid and Invalid Parameter..... 264
Using GENKEY and RETKEY..... 265
Figure 4-58. Initial Configuration 266
Figure 4-59. Session Key Generation at Host ... 266
Figure 4-60. Session Key Translation at Host ... 267
THE CRYPTOGRAPHIC KEY DATA SET 267
Table 4-4. CKDS Record Format..... ?
Figure 4-62. CKDS Entries..... ?
SUMMARY 269
REFERENCES 269
Other Publications that Treat Key 270

CHAPTER FOUR

Communication Security and File Security Using Cryptography[1]

Previous chapters have introduced the reader to the fundamentals of cryptography from a conceptual and, at times, abstract viewpoint. The discussion has centered around the types of cryptographic algorithms in use today, their salient properties, and some principles used in designing strong algorithms.

Beginning with this chapter and continuing throughout most of the book, cryptographic applications are described that provide data privacy, data integrity, or both. Communication security (**COMSEC**), file security (**FILESEC**), personal verification, message authentication, and digital signatures are discussed. This chapter deals mainly with methods for incorporating a conventional cryptographic algorithm, such as DES, in a data processing network to provide communication security. Also described are methods of using DES to secure data stored on removable media, such as tapes and disks. Methods for implementing public-key algorithms are not described, although some of the material may be applied to the design of public-key cryptographic systems.

For COMSEC, end-to-end encryption—the encipherment of data at their point of origin and decipherment only at their final destination—is emphasized because it provides the most security. When properly implemented, a host processor's encryption capability can be used for communication security, file security, and other applications involving cryptography. Before a step-by-step discussion of how end-to-end encryption can be implemented in a data processing network, an overview of network configurations and the various ways that cryptography can be implemented is appropriate.

NETWORKS

Soon after the first computers were developed for scientific data processing applications, people realized that they might be applied to accounting tasks. System designers and manufacturers then tried to extend the benefits of the

[1] The material contained in this chapter elaborates and extends the ideas contained in References 1 and 2.

computer to more people and businesses. The application of computers to communications was inevitable.[2]

At first, typewriter-like devices were designed and attached to the computer in the same way that punched card readers, magnetic tape devices, and line printers were. Through the keyboard the user could request programming services without punching cards. Similarly, program execution was performed immediately, or in real time, as opposed to the delay normally experienced between submission of a job for batch processing and receiving the output, or results. Systems programs called operating systems made program execution in real time possible. Useful results and reports of errors were returned to the user with almost no noticeable delay.

While such devices were satisfactory for users near the computer, they did not serve remote users. This problem was solved when it became possible to send computer data over voice-grade analog telephone lines.

The development of the transmission control unit, a device capable of controlling the telephone line and attached devices, contributed to this breakthrough. System programming at the computer formatted data and managed the control unit and device through standardized protocols: the agreements reached between two parties on the format and meaning of control messages and the sequence of control messages to be exchanged between the two parties. Unfortunately, application development was inhibited by the need for device-dependent support in each application program for each type of device. Architectures were subsequently developed to unify network operations (see Incorporation of Cryptography into a Communications Architecture, Chapter 7).

At the same time, advances in technology, specifically the development of microprocessors, led to the introduction of programmable communications control units and programmable device control units. These units assumed line and device control functions previously performed by application programs. As a result, a major portion of the network management responsibility was relocated to various network devices thus allowing the host processor to perform other functions. Gradually, data communications between hosts as well as between a host and its attached devices (terminals, printers, facsimile machines, etc.), over switched (public) or nonswitched (dedicated) lines, became generalized such that any node (terminal, communications controller, or host processor) could communicate with any other node.

The microprocessor revolution also had an effect on the computational capability of terminals and control units. In addition to the fixed functions of line and device control, particular applications could be performed by specialized devices because sufficient computing power was available for additional functions. As the capacity of microprocessors increased, additional functions were off-loaded from host processors and performed by various microprocessor-driven devices. Systems evolved wherein data processing functions were performed by devices situated in different places and connected by transmission facilities. Thus data could be partially or wholly processed

[2] A short history of this development can be found in Reference 3.

at any number of network nodes—a concept that became known as *distributed data processing*. The common element in this development was the network architecture which established the basis for device attachment and specified the protocols necessary for device interaction.

A *data processing network* (Figure 4-1) is a configuration of data processing devices, such as processors, control units, and terminals, which normally are connected by data links for the purpose of data processing and information exchange. The links may be processor channels, satellite and microwave links, or switched and nonswitched communications lines. A network may be de-

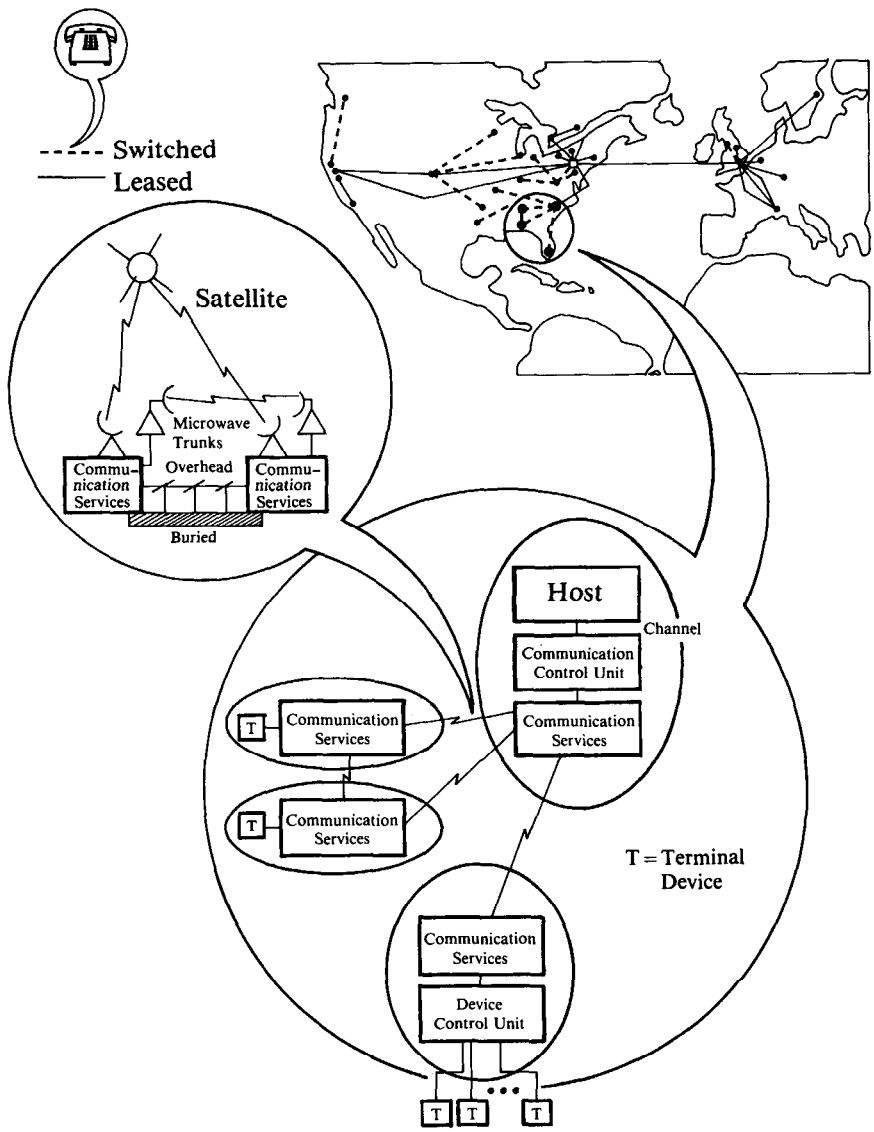


Figure 4-1. Data Processing Network

scribed, in terms of the patterns formed by its links, as a star, loop, tree, or mesh. In any case, the network's basic function is to provide *access paths* by which a *user* (person or program) at one location can communicate with another user at some other location [4].

As more network facilities are used to process and transmit data, there is an increased dependency on communication facilities provided by a communication common carrier for hire by the general public. Likewise, the opportunity and ease with which data can be intercepted increase. An architecture for networks must therefore provide a capability to implement appropriate security measures should they be required.

NETWORK ENCRYPTION MODES

There are three ways to incorporate cryptography into a communications system: *link*, *node*, and *end-to-end* encryption.

Link encryption (Figure 4-2) protects data between adjacent network nodes. The algorithm is implemented in cryptographic devices that bracket (i.e., are situated at opposite ends of) a communication line between two network nodes. The two devices are positioned between their respective nodes and associated modems (modulators/demodulators) and are equipped with identical keys.

Node encryption (Figure 4-3) is similar to link encryption in that each pair of nodes shares a key to protect data communicated between them. However, data passing through an intermediate node are not in the clear, as would be the case with link encryption. Rather, at an intermediate node, the enciphered data are transformed from encipherment under one key to encipherment under another key (deciphered and reenciphered) within a security module or protected peripheral device attached to the node. (Node encryption is defined solely for completeness and discussed only in this section.)

End-to-end encryption (Figure 4-4) continuously protects data during transmission between users. Unlike link and node encryption, end-to-end encryption permits each user to have several keys, one key for each user who uses encryption. Data are deciphered only at their final destination—they never appear in clear form at intermediate nodes or their associated security modules.

With link and node encryption, the user is normally not aware that messages are receiving cryptographic protection (i.e., the cryptographic function is provided by the network, and is transparent to the user). A user-transparent form of end-to-end encryption occurs if the cryptographic function is provided automatically through system services. If the user makes specific requests for cryptography, its use is not transparent. This latter case is referred to as *private* cryptography.[3]

One system service required to support transparent end-to-end encryption involves the selection or assignment of a cryptographic key for enciphering

[3] Private cryptography is not the opposite of, nor should it be confused with, cryptography using a public-key algorithm. For a description of public-key algorithms, see Cryptographic Algorithms and Public-Key Algorithms, Chapter 2.

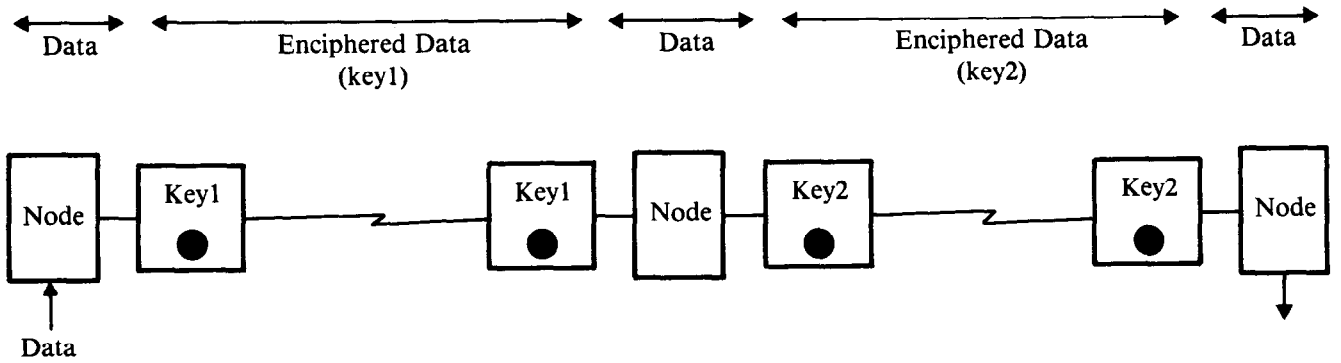


Figure 4-2. Link Encryption

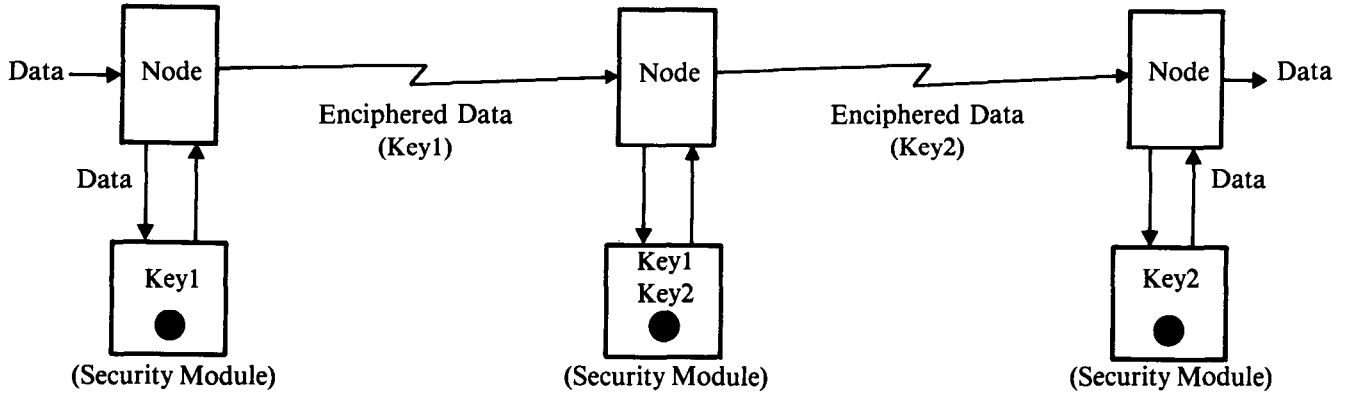


Figure 4-3. Node Encryption

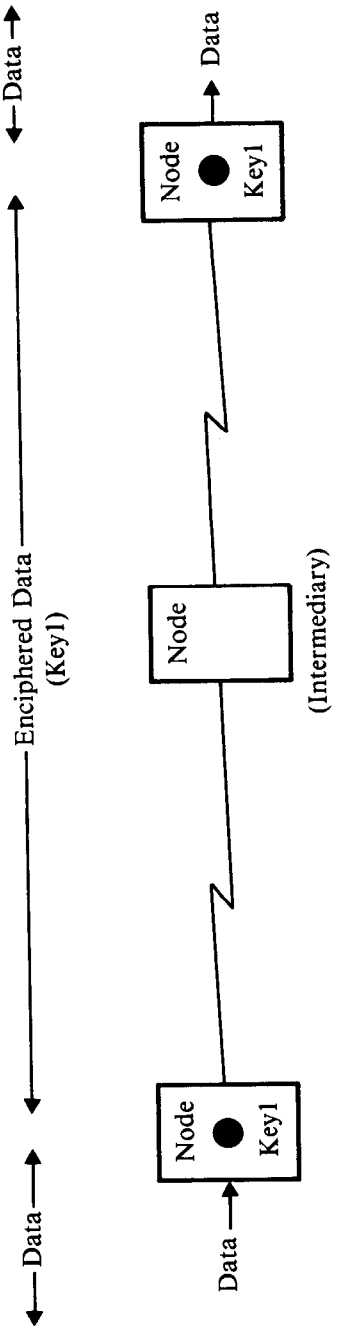


Figure 4-4. End-to-End Encryption

and deciphering data between communicating users (or the network nodes at which the users are located). If one or more nodes in the system cannot use encryption, or if encryption of only selected messages is desired, an additional system mechanism is required to enable and disable the encryption function. In either case, if cryptography is to be used by a network node, then the encryption capability must be integrated (logically or physically) into the node. In effect, this means that key management for end-to-end encryption is not affected by the way a node is attached to the communication channel.

A host processor's encryption capability can be integrated into the central processing unit (CPU), into a front-end processor, or into a separate unit attached to a CPU's channel–a device that connects the CPU and main storage with input/output control units. There are many trade-offs between cost and performance that must be considered in selecting an approach. However, the latter approach, being compatible with many CPU designs, has the advantage that only a single device need be designed.

With link encryption, every node in the selected path over which encrypted data pass must have a standalone cryptographic device connected to its input and output ports. With node encryption, every node in the selected path over which encrypted data pass must have its own security module (integrated or attached). With end-to-end encryption, only those nodes that originate or receive encrypted messages require an encryption capability; this can significantly reduce the places where cryptography, or cryptographic devices, must be used in the network.

In general, the information communicated between network nodes consists of a message, or the data to be exchanged between users, and a message header. Generally, a header contains routing information; for example, the intended destination, the message's sequence number, the identity of the message's source, indicators denoting the start and end of text, the classification of the message (i.e., whether it contains control commands, data, or both), and possibly its format.

With link encryption, both a message and its header could be encrypted (Figure 4-5). With either end-to-end or node encryption, only the message, but not the routing information, can be encrypted. This is because each intermediate node in the communications path must examine the routing information in order to direct the message to its intended destination. Of course, if link encryption and either end-to-end or node encryption were implemented in the same network, messages would be doubly encrypted and headers would therefore be encrypted on a link basis.

If clear and encrypted messages are intermixed, a mechanism must exist to separate them. In one approach, a bit in the message's header is used to indicate whether the message is encrypted. In another approach, consistent with some line protocols, special control messages specify when to start and stop encryption. Finally, a high security mode of operation is possible if the mechanism for turning encryption on and off is ignored and all message traffic is encrypted.

Since with end-to-end and node encryption, and in some cases with link encryption, the header is in clear form, data communications are susceptible to a form of *traffic analysis* wherein an opponent obtains statistics related to

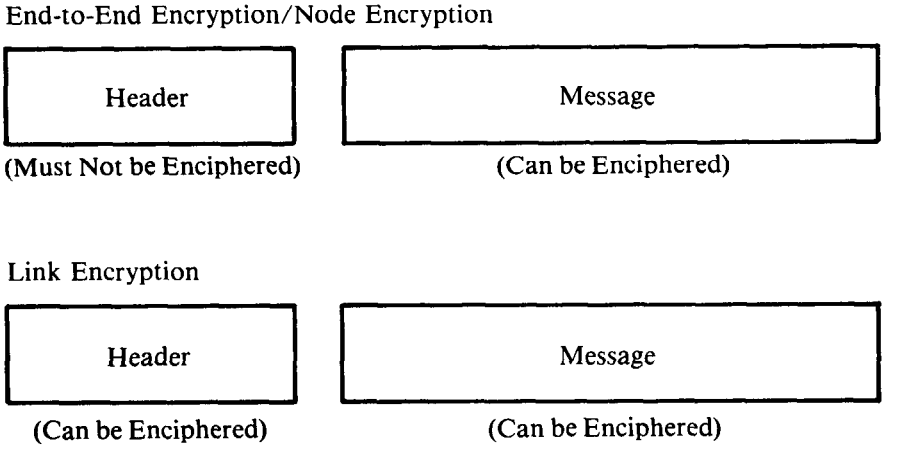


Figure 4-5. Message and Header Encryption

the number of messages transmitted to or from a given node. Other forms of traffic analysis are possible when headers are encrypted. Used by the military, traffic analysis has often given advance warning of enemy activity in a particular sector or combat zone. Traffic analysis can be prevented easily (at the expense of efficiency) by deliberately keeping a line active with bogus message traffic, thus masking the occasions when an unusual amount of meaningful traffic is transmitted. However, since traffic analysis is less of a threat in commercial applications, it is not discussed.[4]

In terms of cost, flexibility, and security, end-to-end encryption appears most attractive for systems requiring many protected links. Thus a cryptographic architecture—the definition and allocation of keys, cryptographic operations, macro instructions, and the like—for end-to-end encryption in a communications network is the subject of most of the remainder of this chapter. Based on this architecture, methods are also developed for the encryption of data files. (The communication architecture to support such a system is discussed in Chapter 7.)

However, link encryption may be more attractive than end-to-end encryption for certain networks and teleprocessing configurations. For example, where the number of links requiring protection is small, only a few link encryption devices are necessary, and therefore the cost of protection is low. Link encryption devices operate transparent to existing programs, and they require no operator action. Moreover, most link encryption devices operate at line speed, thus causing no noticeable degradation in transmission performance. Finally, some teleprocessing devices, because of their design or the way they are managed by programming, will not support end-to-end encryption. For these reasons, link encryption is discussed in greater detail before developing the subject of end-to-end encryption.

[4] The interested reader may wish to consult Baran [5] and Chaum [6], who address the traffic analysis problem.

FUNDAMENTALS OF LINK ENCRYPTION

Whereas the objective of end-to-end encryption is to protect data over the entire path it must traverse from a source node to a destination node, the objective of link encryption is to protect only the portion of the total path, or link(s), where the opportunity for interception exists or is the greatest.[5] The definition of link includes terrestrial (telephone wires, microwave, optical fiber, cable television, etc.) and satellite communication services which (1) accept a signal from a portion of a data processing system (or business machine), (2) transport that signal to a distant point, and (3) deliver the signal with exactly the same bit sequence it had when it was received. Such services are provided by communication common carriers (Western Union Telegraph Co., American Telephone and Telegraph Co., General Telephone and Electronics Co., and numerous independent telephone companies), specialized carriers, satellite carriers in the United States, and the Postal Telephone and Telegraph Administrations (PTTs) in many other countries. Excluded from the definition are the CPU's channel and the input/output (I/O) bus used to attach I/O control units (i.e., devices not connected to a communication channel).

From a practical point of view, an encryption device operating independently of the communication channel is independent of, and does not require redesign of, existing data processing and communications equipment (Figure 4-2). However, the placement of the encryption device need not be external to a terminal or its modem. Encryption apparatus could, in fact, be included "under the covers," along with a modem (should the design of either piece of equipment so specify). The ultimate decision rests with the manufacturer of the equipment and is based on the requirements of the market.

Regardless of implementation, however, the data encryption equipment (DEE) must conform to the established interface between system components, generally called data terminal equipment (DTE) and data circuit-terminating equipment (DCE).[6] (This terminology has been accepted by the International Consultative Committee for Telegraph and Telephone (CCITT), the International Organization for Standardization (ISO), the American National Standards Institute (ANSI), and the Electronic Industries Association (EIA).) Included are the electrical signal characteristics, interface mechanical characteristics, and the functional description of interchange circuits needed to establish, maintain, and disconnect the physical connection between a DTE and DCE or between two DTEs.

In some cases, improved link efficiency can be realized if the DEE is designed to a specific data link control protocol. These rules regulate the initiation, checking, and retransmission (if any) of each data unit presented to the link for transmission, and are independent of the media used. This improves the DEE's ability to determine when to start and stop encrypting/decrypting.

[5] Much of the information in this section regarding link protocols was taken from Chapter 11 in Reference 7. Interested readers are referred to this text for supplemental reading.

[6] A DTE corresponds to a data terminal or communications controller. A DCE corresponds to a modem or its functional equivalent in a public data network.

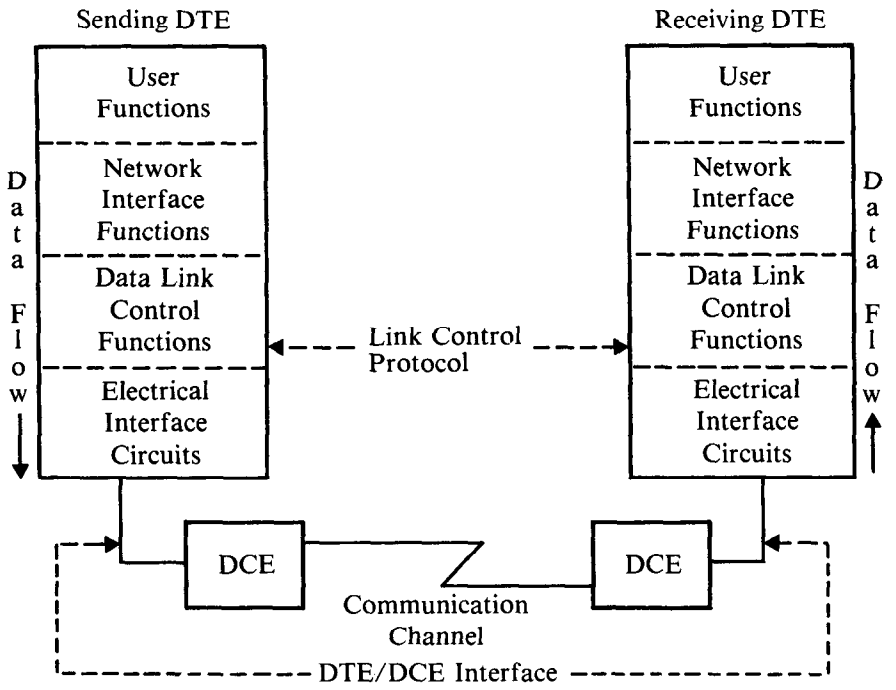


Figure 4-6a. DTE/DCE and Link Control Interface

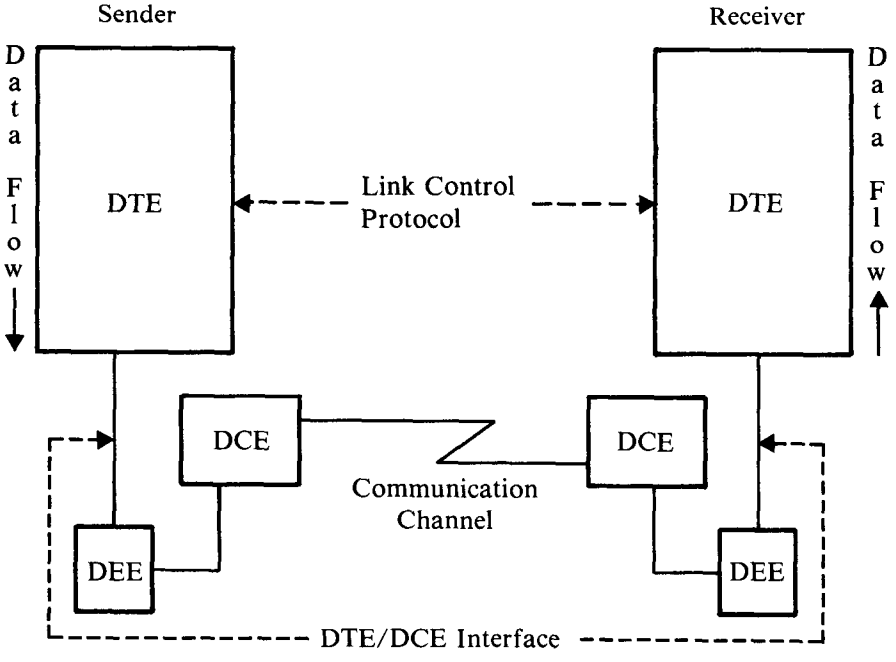


Figure 4-6b. DEE Placement—Link Encryption

Implementations which are not link protocol-dependent also exist. In these cases *every* bit is cryptographically processed without consideration by the DEE as to the format of the data unit being transmitted. Figure 4-6a illustrates the DTE/DCE interface and the arrangement of functions within DTEs.

DTE/DCE interfaces between data processing equipment and communication equipment are standardized in the United States through the efforts of the EIA, and worldwide through CCITT. Almost all manufacturers adhere to the EIA and CCITT standards so that products of different manufacturers can communicate with one another.

For the same reason, link protocols are standardized in the United States by ANSI, and worldwide through ISO. Practically then, link encryption devices must be implemented in conformance with these standards. Thus the presence of the DEE, residing on the DTE/DCE interface (Figure 4-6b), is transparent to both DTE and DCE.

Link protocols applicable to character-coded information generally fall into two broad classes: *asynchronous* and *synchronous*. The difference is related to the mechanism used to provide (bit) synchronization of the transmitted data character(s).

With an asynchronous protocol, the bits within each character, or block of characters, are sent at distinct time intervals as determined by clock pulses generated by the DTE. The receiving DTE receives the transmitted bits at the same rate, having the ability to produce clock pulses identical to those of the sender. But the first bit of each character, or block of characters, can be sent at any time. By contrast, the time of occurrence of each bit or byte transmitted synchronously, including the start bit or start character, is related to clock pulses synchronized between the sending and receiving DTEs. The protocols commonly used for asynchronous and synchronous transmissions are discussed below. A method for incorporating cryptography in each is suggested.

Asynchronous

With this protocol, each data character is preceded by a *start* bit and followed by at least one *stop* bit (popularly referred to as start/stop protocol). The start and stop bits delimit each character being sent. Depending on the character code/set in use, a character could be represented by 5, 6, 7, or 8 bits.

The start bit must always be sent in the clear. Encryption can, and usually does, begin with the first data bit and normally ends with the last data bit. The stop bit may or may not be enciphered, although in most present designs it is sent in the clear. A DEE supporting this protocol could be designed to operate with a specific character code, for example, 7-bit ASCII (American National Standard Code for Information Interchange), or with multiple codes. In this latter case, the device must be initialized via some external input to operate with one code during one period and then be reinitialized to operate with another code during a different period. Implementation options, such as these, may vary from one manufacturer to another.

Byte-Synchronous

This protocol is suitable for 7- and 8-bit ASCII and 8-bit EBCDIC (Extended Binary-Coded Decimal Interchange Code). Characters entered at a keyboard are first collected in a buffer. The entire message is then transmitted at intervals determined by the clocking mechanism supported by the two DTEs. Rather than rely on start and stop bits for character synchronization, the DTE uses a different source of timing. For example, some DCE's maintain a master clock signal (referred to as modem clocking). However, it is still desirable (as in the start/stop protocol) to derive the bit-synchronizing signal from the 0-to-1 and 1-to-0 transitions in the bit-stream itself. This is accomplished in the DTE with a bit-clock, the primary component of a business machine clock, since this avoids the consequences of any tendency of the DCE's clock to drift. In binary synchronous communication (BSC), for example, sufficiently frequent 0-to-1 bit transitions are provided through the use of the special PAD and SYN characters.

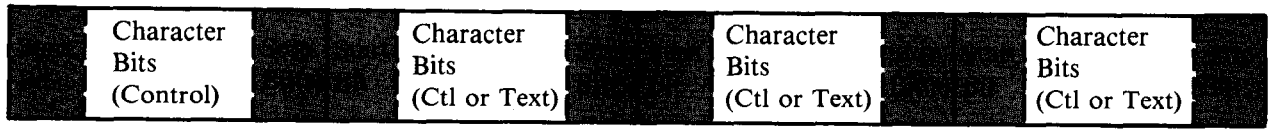
The PAD character (a set of alternating 1s and 0s) helps establish bit synchronization between the DTE and DCE. The SYN character, which may repeatedly appear in the data, helps the receiving DTE maintain character (byte) synchronization with the sending DTE. For example, initial synchronization, following a line turnaround, is assisted by the control-character sequence of PAD SYN SYN. Additional synchronizing (SYN) characters in the data stream not only ensure that the receiving DTE stays in bit synchronization but also establish which bits are the first and last in each character (i.e., they provide character synchronization as well). The content of the message is delineated by the following additional control characters:

SOH (start of header)
STX (start of text)
ETX (end of text)

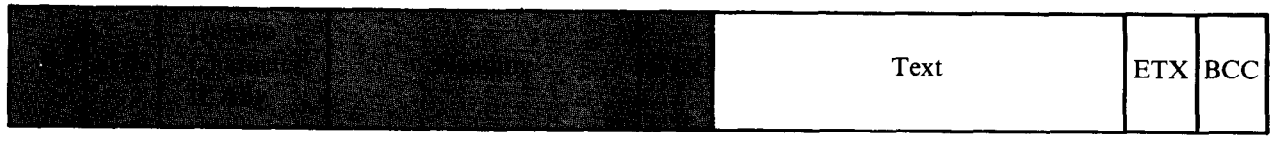
With this very brief explanation, the use of cryptography to protect a message can now be understood. In one approach, intended to protect only the text portion of the message, the DEE scans the data stream for an STX character, used to signal the beginning of encryption/decryption. Encryption/decryption terminates when a corresponding ETX character is encountered. (ETX may or may not be encrypted.) Alternatively, the SOH character could signal the DEE to begin encrypting/decrypting, if it was desired to protect header information. But note that in this case the entire message is encrypted/decrypted. Therefore, throughput is adversely affected since encryption is always applied, even for messages containing no text. (In BSC, control messages are used to convey link commands.)

Ordinarily, a group of bits (called the block check character) is added to the message after the ETX. These bits, which are a function of all the bits in the message (including the header), are used to detect errors that can occur during transmission. When encryption is employed, the block check character can and may be encrypted, in which case its recomputation by the receiver is performed after decryption.

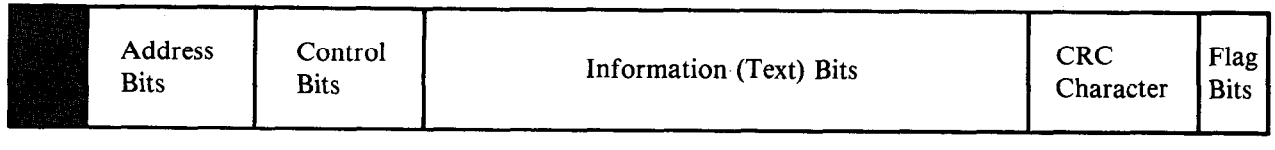
Start-Stop: Encrypts Everything but Start and Stop Bits



BSC: Encrypts Only Text and Ending Characters



SDLC: Encrypts Everything but the Initial Flag



= "In the Clear"

Figure 4-7. Example of Extent of Encryption, by Protocol

Bit-Synchronous

Some link protocols are bit-oriented, rather than character-oriented.[7] Therefore, the bit pattern of a particular character is ignored by the DTE and the protocol is said to be character-code independent. Each transmission is synchronized by a unique delimiter called a *flag* (F). Transmitted data are identified by leading and trailing flags, and a unit of transmission is referred to as a *frame*. The generalized format of a frame is:

[Link Header] [Information] [Link Trailer]

or

[F] [Information] [F]

When a series of frames is sent as a group, the trailing flag of one frame is also the leading flag of the next frame (i.e., the frames are separated by a single flag). By convention, sequences of eight or more consecutive 1-bits are never transmitted except to indicate an idle line condition.

Even though the discussion above is brief, a generalized approach to encryption can be inferred therefrom by means of an example using IBM's Synchronous Data Link Control (SDLC). Because link encryption is effected between two adjacent nodes, in which case routing information must be unencrypted at the nodes but not on the link, SDLC encryption can commence with the detection of a leading F (where F is the bit string 01111110) and terminate after the trailing F (F followed by eight or more 1-bits). Decryption follows the same rule. Of course, the trailing flag would have to be identified after decryption to delimit the frame. Calculation and comparison of the block check character, which is included in the information segment of the frame, likewise is performed after decryption. Figure 4-7 illustrates one of many possible encryption implementations to achieve link data protection and serves to summarize this subject.

AN OVERVIEW OF END-TO-END ENCRYPTION

In a data communications network, assume that a person (end user) at a terminal is communicating with an application program (also an end user) through a processor of some type. These communicating end users share a common key, which may be a personal (private) key provided by and agreed upon by the users in advance or, for *transparent* cryptographic data protection, a key dynamically generated and assigned to these users by the system. This latter *data-encrypting* (or *data-decrypting*)[8] *key* is active only for the

[7] Several bit-synchronous protocols have evolved from ANSI's Advanced Data Communications Control Procedure (ADCCP), and the ISO equivalent, High Level Data Link Control (HDLC).

[8] Recall that a conventional algorithm is assumed.

duration of a single communications session, and therefore is called a *session key*.

For file security, applications programs are the participants (end users that encipher or decipher files). The data-encrypting key used to protect a file is called a *file key*. And a different file key, which is provided by either the end user or the system, may be assigned to each file. Subsequently, an encrypted file may be decrypted (recovered) at any host with an encryption capability, including the one at which it was created, provided that the file key is made available to that host.

For communication security, session keys are generated at a host and then transmitted to a receiving node (terminal or host) via a communications network (assumed nonsecure). The session key is kept secret by enciphering it under another key, defined a *key-encrypting* (or *key-decrypting*) *key,* which has been installed in advance at the receiving node. This approach allows each receiving node to have a unique key-encrypting key. Therefore, if a key-encrypting key at a terminal is compromised, the security exposure is localized to that specific terminal and does not jeopardize the security of the entire network—a highly desirable feature.

In the particular system under consideration, one set of key-encrypting keys is used to encipher session keys transmitted from host to host, and another set of key-encrypting keys is used to encipher session keys transmitted from host to terminal. Each host must therefore store the key-encrypting keys of each host and terminal that it communicates with. Potentially, many key-encrypting keys may be required. A terminal, on the other hand, is required to store only one key-encrypting key, called a *terminal master key*. Session keys are sent from a host to a terminal enciphered under the terminal's master key.

The terminal master key's secrecy is achieved by storing it in a protected area called the *cryptographic facility*. The cryptographic facility is a secure implementation containing the cryptographic algorithm (DES is assumed), and a nonvolatile memory where the master key is stored. It can be accessed only through inviolate interfaces (secure against intrusion, circumvention, and deception), which allow processing requests, key, and data parameters to be presented, and transformed output to be received. A similar cryptographic facility is available at the host.

Key-encrypting keys stored at a host processor are kept secret by enciphering them under a *host master key*. This method of protecting keys is referred to as the *master key concept*. The host master key, like the terminal master key, is kept secret by storing it in the host's cryptographic facility.

Because of the large number of cipher keys used at a host processor, automated procedures are required to generate and manage these keys. The key generator and the key manager are two host programs provided for this purpose. The key generator generates the key-encrypting keys that are required by the host. However, key-encrypting keys can also be specified by installation personnel (e.g., personal keys that have been created by individual users, or keys that have been generated at some other location by another key generator). In any case, the key-encrypting keys used by a host are stored in a key table. A second copy of each key-encrypting key is transmitted

securely (by courier or some other means) and installed in the respective terminal or host.

Due to the large number of keys required at a host, it is customary to store the key table on a disk or drum (i.e., in secondary storage rather than in the computers main memory). In this discussion the key table is called the *Cryptographic Key Data Set* (**CKDS**). It resides on secondary storage and is assumed to be accessible during normal system operation. The key generator has exclusive write privilege to this data set to add, change, and delete keys. The key manager and key generator have exclusive read privilege to the data set to accomplish the tasks of translating keys. Access to the CKDS is denied to all other programs.

The translation function of the key manager involves reenciphering a key from encipherment under one key to encipherment under another key. Requests for key translation are made via **GENKEY** (Generate Key) and **RETKEY** (Retrieve Key), two *programming calls* used to invoke the key manager function.

Another programming call, denoted **CIPHER**, invokes the encipher and decipher data function. It also provides a way for the calling program to specify extended options, such as block chaining (see Block Ciphers with Chaining, Chapter 2).

Six basic cryptographic operations are defined to the host's cryptographic facility. These operations, either alone or in combination, provide the cryptographic services requested by the GENKEY, RETKEY, and CIPHER calls, and all other key management functions required by the system. The implementation of the basic operations is such that a clear key cannot be recovered outside the cryptographic facility, regardless of the order in which the basic operations are exercised. Since this property is a requirement for the cryptographic system to be secure, let it be restated in a stronger form:

> *It must not be possible to recover keys in the clear outside a designated physically secure area, such as a cryptographic facility, regardless of the inherent security of the supporting host operating system.*

CIPHER KEY ALLOCATION

The formal treatment of end-to-end encryption begins with a discussion of cipher key allocation. To provide good overall security, different cryptographic applications require different types of cipher keys. (This will be demonstrated later when analyzing a particular implementation.) Thus an important step in the design of any cryptographic system is the initial specification of its cipher keys. This includes a statement or declaration of the intended purpose and use of each class or type of key and the procedures for their protection.

Technically, cryptographic keys are data. However, it is useful in a discussion involving cryptography to distinguish between keys (key data) and data that are not keys (nonkey data, or simply data).

Specification of Cipher Keys

The designer of a cryptographic system must answer the following questions:

1. What nodes in the system require cipher keys and how are these keys initialized or set into the nodes?
2. How often are cipher keys changed, that is, what is the expected life of a key?
3. Where in the system, and under what conditions, are cipher keys created?
4. How are data and cipher keys protected?

In the system to be described, both terminals and host processors can have encryption capabilities. The encryption capability (a *facility* capable of performing encryption and decryption) is invoked by a user located at the terminal or host processor.

For terminals capable of input and output operations, the end user is a human being (Figure 4-8). It is assumed that only one end user can be active at any given time. Terminal control units (also called cluster control units) that can support several terminals and their respective end users concurrently are, for simplicity, not specifically identified in this discussion. However, the developed key management scheme would support these devices, provided that the control unit's cryptographic operations were designed differently to accommodate several enciphered session keys.

At the host processor, end users are application programs. Because of multiprogramming (concurrent execution of two or more programs by a host processor), many application programs can contend for the use of the host's cryptographic service at the same time (Figure 4-9).

For the encryption capability in a node to function, it must have a copy of the cryptographic algorithm (DES is assumed) and the specific cryptographic key to be used for ciphering data. The key that is in use at any given time is defined as the *working key*. To prevent the working key and intermediate rounds of encipherment from being probed by an opponent (since knowledge of intermediate rounds reduces the work factor associated with the algorithm), the cryptographic algorithm and working key are maintained or stored in a

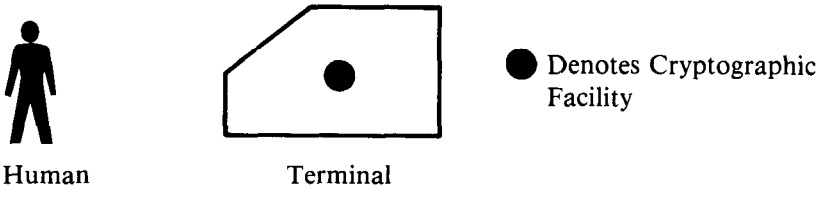


Figure 4-8. Terminal End User

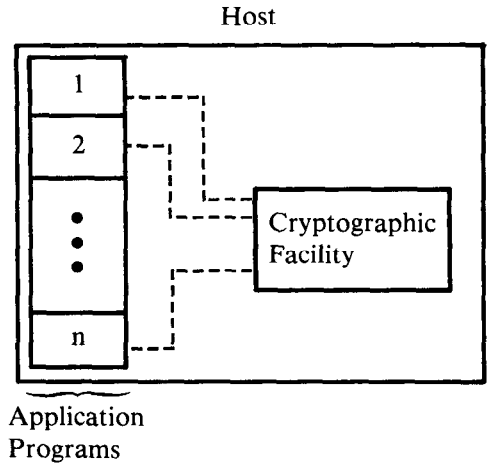


Figure 4-9. Host End Users

protected area defined as the *cryptographic facility* (Figure 4-10; see also The Cryptographic Facility).

It may be helpful for the reader to think of the cryptographic facility as a physical object, for example, a standalone hardware device or security module with its own protective covers [8]. Or one can think of the cryptographic facility as a special component integrated into another device, such as a terminal or computer. In either case, the security of the cryptographic facility is a function of the protection features associated with or provided by its physical embodiment and/or the device within which it is integrated; for example, probe resistant packaging, automatic tamper-proof detection features, covers, and the like, and the access control measures (administrative controls, locks and keys, badges, guards, fences, and so forth) that are present to limit access to the cryptographic facility or its embodying cryptographic device. The cryptographic facilities shown in Figures 4-8 and 4-9 are integrated into their respective devices.

When implemented in software, the boundaries of the cryptographic facility are not well-defined. In such cases, the physical protection achieved in hardware must now be achieved logically through programming. Ultimately, the degree of protection will depend on a processor's hardware protection features as utilized by the resident operating system (e.g., store and fetch protection, privileged operations, program execution modes, and the like). Thus the security of software implementations is no better than that of the underlying operating system.

To be initialized, at least one cipher key must be inserted into the cryptographic facility in clear form. This initial key must be transmitted to the facility in a secure manner, without compromising its secrecy. Subsequently, other keys can be introduced into the cryptographic facility by the system automatically. A key can be transmitted to a cryptographic facility over a nonsecure path as long as it is first enciphered under a key already present in the receiving cryptographic facility.

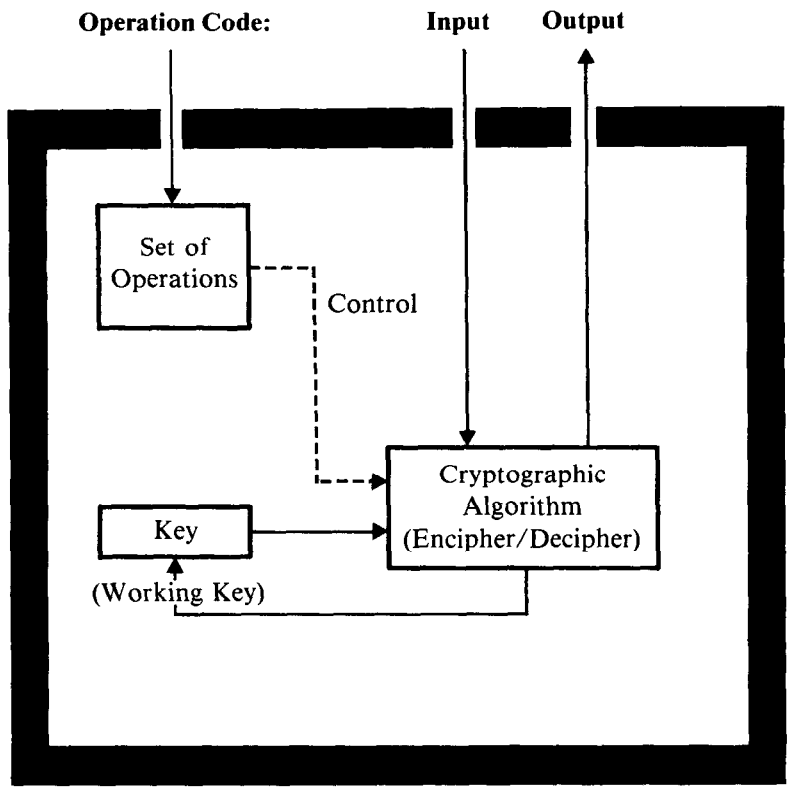


Figure 4-10. Cryptographic Facility—General Concept

The cryptographic facility provides the means for inserting data or keys to be ciphered. A control line is provided to activate the desired operation. The results of the operation are provided as output, except when an enciphered key is deciphered to produce a working key. In this case, the result is retained within the cryptographic facility.

The initial key may be a personal key associated with a particular user, or a system-supplied key associated with a particular node. In either case, the key is manually entered into the node's cryptographic facility. A personal key is normally entered by the user at the time ciphering is required. Conversely, a key supplied by the system is entered by installation personnel at the time the system is first initialized.

The described method of key management allows keys to be managed entirely by the system and its operating personnel, thus achieving cryptographic transparency. As indicated earlier, transparency is a highly desirable feature. However, the design does allow for user-supplied keys, and users have the option to manage keys and invoke the system's cryptographic operations themselves.

Nodes at which data encryption is desired must be equipped with identical encryption algorithms, and each node must have a copy of the same cipher

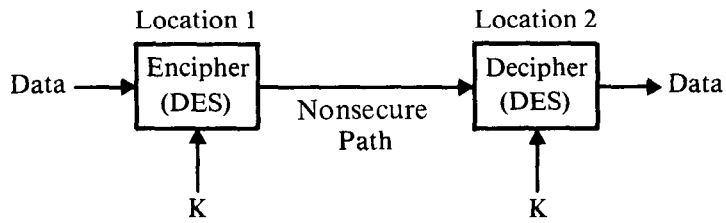


Figure 4-11. Transmission of Enciphered Data from Location 1 to Location 2

key K (Figure 4-11). Though two nodes must always share a common data-encrypting (or data-decrypting)[9] key to permit enciphered communications, greater security is achieved if different data-encrypting keys are used by each pair of communicating nodes. Such a design minimizes the resulting damage if a key should become compromised.

Using different data-encrypting keys for enciphered communications between each pair of communicating nodes has other advantages. Information intended for one node cannot be surreptitiously decrypted at another node, and misdirected messages cannot be accidentally deciphered by unintended recipients.[10]

The cryptographic system being described uses a different key for enciphering the data transmitted or transferred between each pair of end users. These data may be transmitted electronically or they may be magnetically encoded on a removable storage medium and transported from one location to another. Subsequent diagrams will use the following notation to differentiate between nodes and end users:

Circle: Denotes a network node (terminal or host)
Square: Denotes an end user (human being or application program)

In this discussion, a key used to protect (encipher and decipher) data is called a *primary key* (**K**). A primary key is also called a *data-encrypting* (or *data-decrypting*) *key*. When a primary key is used directly to provide communication security, it is called a primary communication key (**KC**), or simply, a communication key. The primary communication key used to protect data during a communications session is called a *session key* (**KS**). When a primary key is used directly to provide file security, it is called a primary file key (**KF**), or simply, a *file key*. A primary key (K) that protects data between end users 1 and 2 can be represented by the following diagram (Figure 4-12).

Primary keys are automatically generated by the system at the request of an end user, although primary keys can be supplied by end users as well. During periods of storage outside the cryptographic facility, these primary

[9] A conventional algorithm is assumed so that the enciphering and deciphering keys are equal.

[10] This is also true for a public-key algorithm.

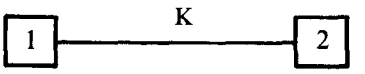


Figure 4-12. Shorthand Notation Representing Data Protection Between Users 1 and 2 Using Primary Key K

keys are protected by encipherment under another key, a *key-encrypting* (or *key-decrypting*) *key*. Keys must be kept secret for the period of their existence or until the data they protect are deemed no longer of value.

A *secondary key* (**KN**), where N stands for node, is one type of key-encrypting key used to protect primary keys (Figure 4-13). A second type of key-encrypting key, defined *master key*, is described below. When a secondary key is used to provide key protection in a communications environment, it is called a *secondary communication key* (**KNC**). When a secondary key is used to provide key protection in a data base environment, it is called a *secondary file key* (**KNF**).

Secondary keys are ordinarily introduced into the system at the request of installation personnel via the key generator, although secondary keys can be specified by installation personnel as well. Secondary keys, as a rule, remain unchanged for relatively long periods of time—months or perhaps years. With the benefit of physical protection, the risk of compromise is reduced.

Once the secondary keys have been manually set in their respective network nodes, data-encrypting keys can be sent from one node to another by encipherment under the secondary key of the receiving node. Similarly, data communicated between end users can be protected by encipherment under an established data-encrypting key.

Both COMSEC and FILESEC subscribe to the same concept of data protection: cryptography is used to protect data in an uncontrolled and presumably hostile environment by encipherment under a data-encrypting key. There are, however, differences in the expected life of the keys.

In a communications environment, a primary key exists only for the time required for two end users to exchange data. Ordinarily, the key would exist for a matter of minutes, perhaps an hour, but rarely for more than a day. In cases where encrypted files are transported between data processing locations, the key would exist for a matter of days or weeks. In contrast, a primary key used to protect archival data could exist for a period of years, or as long as the file is retained.

Figure 4-13. Shorthand Notation Representing Key Protection Between Nodes A and B Using Secondary Key KN

During the initiation of a communications session, a session key (either dynamically generated by the cryptographic system or supplied by the end user) is assigned to the session. At the completion of the session, the session key is erased or overwritten. Thus each session carries on cryptographic communications with a different key, thereby reducing the amount of data encrypted under a single key.

When used in a data base environment, the cryptographic system dynamically provides, or accepts from the end user, a key analogous to the session key (a file key) which it then assigns to a file. The file keys are protected via a secondary file key, which is analogous to the secondary communication key mentioned above. The net result is that different files are encrypted with different file keys, just as the data for different sessions are encrypted with different session keys. When a personal key is used as a file key, access to encrypted data also depends upon the user's ability to supply the correct key.

The sequence of events involved in gaining cryptographic protection for data communications between two end users can now be summarized. In order for the two nodes to establish a common primary key (data-encrypting key) on behalf of their respective end users, they must share, or have access to, a common secondary key. For good security, a different secondary key should be used between each pair of nodes. During the initiation phase (Figure 4-14), a dynamically generated primary key (K) is sent from node A to node B enciphered under a secondary key (KN) previously installed at each node. During the communication phase (Figure 4-15), end users 1 and 2 located at nodes A and B, respectively, communicate using primary key K.

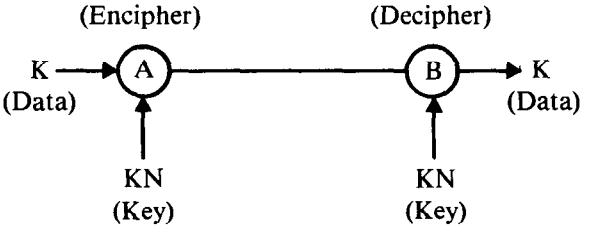


Figure 4-14. Initiation Phase

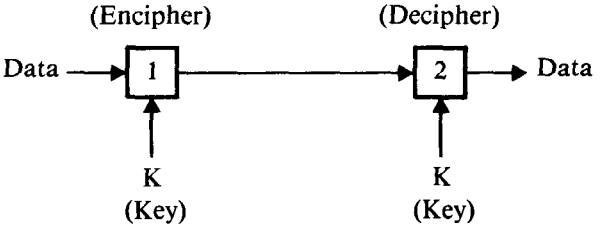


Figure 4-15. Communication Phase

In a communications environment, key distribution occurs at the time a session is established. Where nodes A and B are hosts and the data being transmitted are in the form of an encrypted file, key distribution can precede, be part of, or follow the transmission of data.

Let us now expand on this concept. It is evident that in a system with n network nodes, all of which require and support cryptography, there are

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

ways in which nodes may be selected two at a time (i.e., in pairs). A total of $n(n-1)/2$ different secondary keys is needed within the system, where each node is required to store $n-1$ different secondary keys, one for each of the other nodes in the system. This is illustrated by a simple four node system (Figure 4-16).

In a system with a realistic number of nodes, say 100, each node would have to store 99 different secondary keys. A total of 4950 different secondary keys would have to be stored in the system. Clearly, the task of installing and managing such a large number of cipher keys would be difficult. However, recognizing that data processing systems have different processing and storage capabilities, one can reduce the number of secondary keys to a more manageable number by concentrating keys at host nodes and installing only one key in each terminal node.

Each terminal, as a rule, is, or can be, associated with a single managing host. In effect, a terminal has a logical owner (the host). Therefore, the collection of terminals and other nodes managed by a single host is defined here as a *domain* or *single domain*. When two or more hosts are logically connected, the resultant network is said to have *multiple domains*. This concept of ownership allows keys to be assigned more simply.

The path between each terminal and its owning host (a single domain) is protected with a unique secondary communication key, installed in the terminal and stored in the host's key table. The path between each pair of hosts

Node	Stored Keys
A	KN1, KN2, KN3
B	KN2, KN4, KN5
C	KN1, KN4, KN6
D	KN3, KN5, KN6

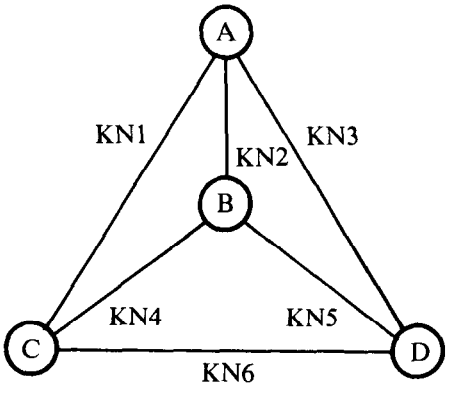


Figure 4-16. Allocation of Secondary Keys in a Four Node System

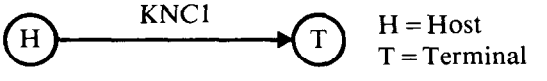
is protected with a pair of unique secondary communication keys, stored in the key table at each host. These key-encrypting keys are *unidirectional*: they are used for data movement in only one direction. The property of unidirectionality is actually enforced via the system's cryptographic operations, which must be appropriately designed (see Chapter 5). In effect, the cryptographic operations permit decryption of encrypted information only at the intended destination node.

Assume that a session key is generated at a host (for matters of economy and practicality). It is enciphered under the secondary communication key (obtained from the host's key table) assigned to each of the receiving nodes. In this form, the session key can be safely transmitted to each receiving node, while enjoying the protection provided by the secondary communication key. The operations defined to the host's cryptographic facility are such that the session key, once enciphered, cannot be recovered at the generating host (under the assumption that the generating host is not a receiving node), preserving the unidirectional property of the secondary communication key.

In an environment with multiple domains, each host must be able to send as well as receive session keys enciphered under a unidirectional secondary communication key. The two different cases (single and multiple domains) are shown below (Figure 4-17), where the unidirectional nature of the secondary communication key is illustrated through the use of a right or left-directed arrow.

In summary, each host shares two different secondary communication keys with each of the other hosts in the network, and a single secondary communication key with each of the terminals within its own domain. Thus in a network with 3 hosts and 2 terminals per host (Figure 4-18), each host must store 4 different secondary keys, each terminal must store 1 secondary key, and a total of 12 different secondary keys are required in the system. The

Single Domain COMSEC:



Multiple Domain COMSEC:

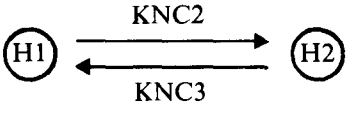


Figure 4-17. Allocation of Secondary Communication Keys

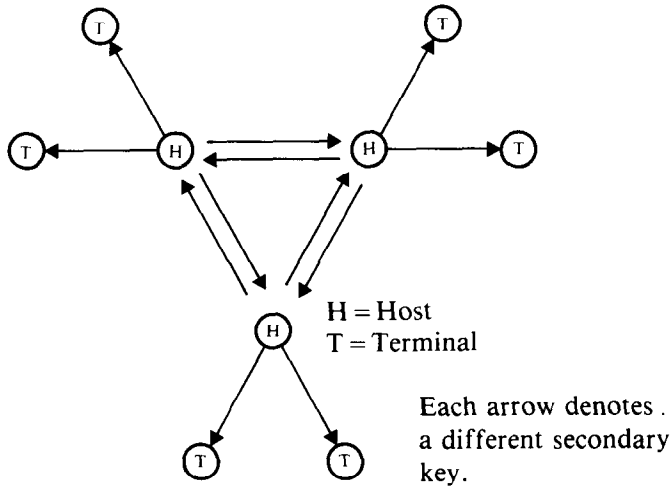


Figure 4-18. Allocation of Secondary Communication Keys in a Communications Network with 3 Hosts and 2 Terminals per Host

allocation of these secondary communication keys can be illustrated by a directed graph, where each arrow denotes a different secondary communication key, KNC (Figure 4-18).

A system with 4 hosts and 24 terminals per host yields a network with 100 nodes. Each host would have to store 30 different secondary communication keys, each terminal would be required to store 1 secondary communication key, and a total of 108 secondary communication keys would be needed by the system. This compares favorably with the prior system of 100 nodes, where the path connecting each pair of nodes was protected with a different secondary communication key. Recall that in such an approach 99 different secondary communication keys were stored at each host and each terminal, resulting in a total of 4950 different keys in the system.

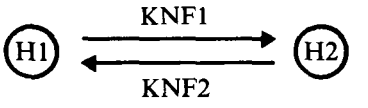
The allocation of secondary keys for FILESEC (Figure 4-19) is similar to that described for COMSEC. However, encrypted files can be created and recovered (decrypted) only at host nodes.[11] Therefore, secondary file keys are needed to protect only the paths connecting two hosts, but not the paths leading to or from terminals. Encrypted files can also be created and recovered at the same host, and so a secondary file key is used to protect the path leading from a host back to itself.[12]

In a network with three hosts (Figure 4-20), each host must store five different secondary file keys, and a total of nine different file keys are required within the network.

[11] This is a convention, not a requirement. The developed key management scheme could be extended to handle file encryption and decryption at terminals, but is not shown.

[12] The corresponding case in COMSEC would involve data communications among application programs within a single host. Data protection for such an environment is a function of hardware protection features and access control procedures, *not cryptography*.

Multiple Domain FILESEC:



Single Domain FILESEC:

Figure 4-19. Allocation of Secondary File Keys

A procedure in which files are encrypted at one host and decrypted at another is analogous to a communication session in which data are transmitted only in one direction. As one might expect, the similarity between COMSEC and FILESEC is reflected in their respective key management schemes. Thus in FILESEC, the secondary key plays the same role that it does in COMSEC; namely, it protects the primary key until it is loaded into the cryptographic facility and used to decipher enciphered data.

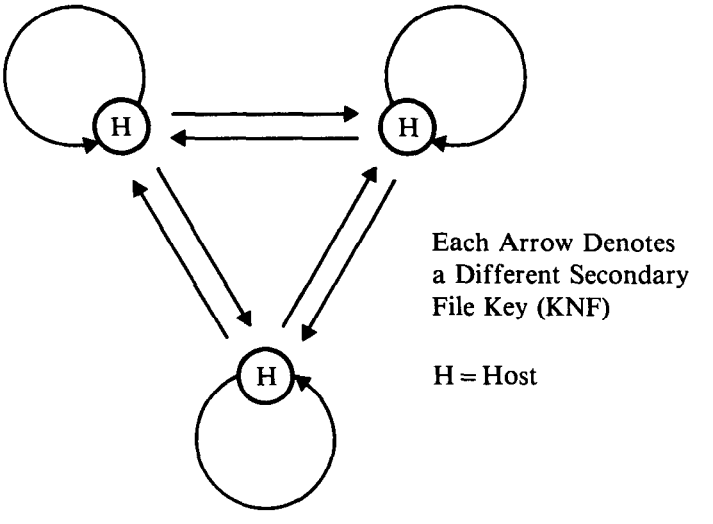


Figure 4-20. Allocation of Secondary File Keys in a Network with 3 Hosts

Let domain i and domain j be two domains within a multiple domain network. The following instances of data encryption are treated within the protocols for COMSEC and FILESEC:

Single Domain COMSEC
Terminal user (domain i) ⟷ terminal user (domain i)
Terminal user (domain i) ⟷ application program (domain i)
Multiple Domain COMSEC
Terminal user (domain i) ⟷ terminal user (domain j)
Terminal user (domain i) ⟷ application program (domain j)
Application program (domain i) ⟷ application program (domain j)
Single Domain FILESEC
Application program (domain i) ⟷ application program (domain j)
Multiple Domain FILESEC
Application program (domain i) ⟷ application program (domain j)

An Example of the Encryption of Transmitted Data

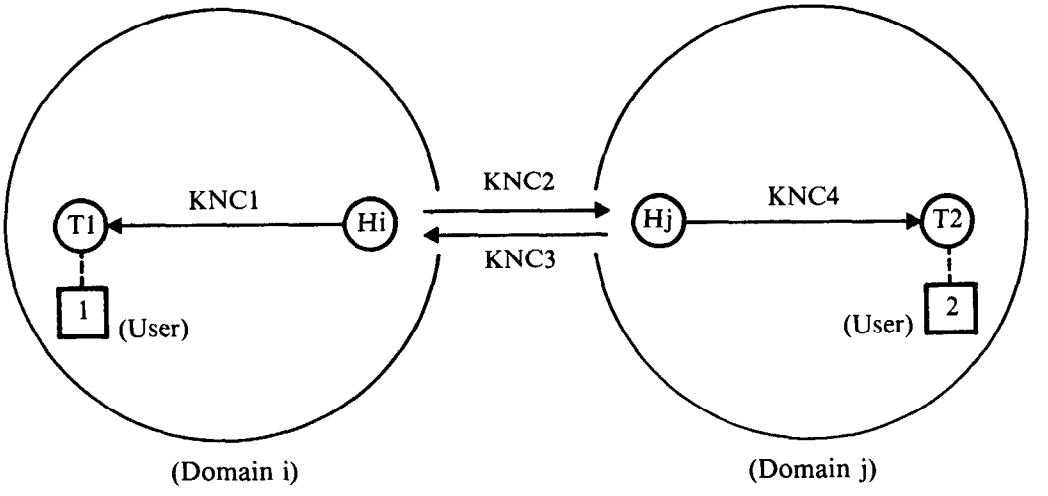
An example is now given, using the concept of multiple domains, to show how a user located at a terminal in one domain can initiate a communications session with a user located at a terminal in a different domain. Cryptography is transparent in this example. In the following diagrams, for simplicity, only the relevant key transformation functions are depicted. Although the complexities of an actual network implementation are purposely omitted, bear in mind that the described operations require a host system which has a full complement of programming support, including an operating system to control the execution of application programs and telecommunication access methods to manage data transmission between the host and other network nodes.

Suppose that user 1 is located at terminal 1 in domain i and that user 2 is located at terminal 2 in domain j. The network configuration and allocation of secondary keys are shown in Figure 4-21.

Each host generates the secondary communication keys for its respective domain, and these keys are securely distributed and inserted into designated nodes by authorized personnel. The means for protecting the keys while they are stored at each node, and the means for using the keys to perform the required key transformation functions, are topics to be covered in the remainder of this chapter.

User 1 initiates the session by invoking the appropriate *logon* procedure, specifying a designated application program in host i. The logon message would, in this case, identify the sending terminal (T1) and the requested destination terminal (T2). As a result, at host i, a pseudo-random number (session key) is generated, which then is enciphered under both KNC1, for secure transmission to T1, and KNC2, for secure transmission to host j (Figure 4-22).

The copy of the session key enciphered under KNC1 is transmitted to



Note: It is assumed in this and subsequent Figures that KNC1 ≠ KNC2 ≠ KNC3 ≠ KNC4.

Figure 4-21. Initial Configuration

terminal 1 and the copy enciphered under KNC2 is transmitted to host j. At host j, a transformation is performed to reencipher the session key from encipherment under KNC2 to encipherment under KNC4 (Figure 4-23). The copy of the session key reenciphered from KNC2 to KNC4, is now transmitted to terminal 2.

Terminals 1 and 2 each recover the session key by deciphering the enciphered value of KS using the stored values of KNC1 and KNC4, respectively,

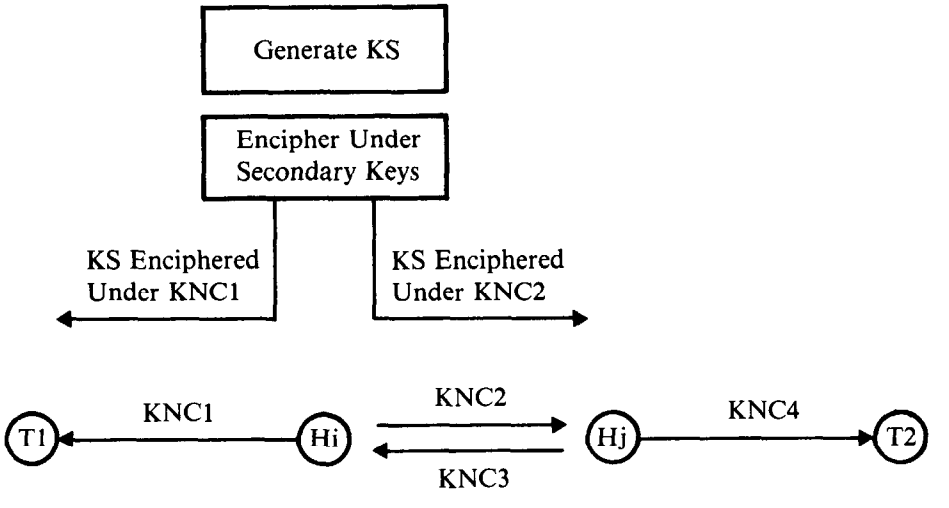


Figure 4-22. Session Key Generation/Encryption—Host i

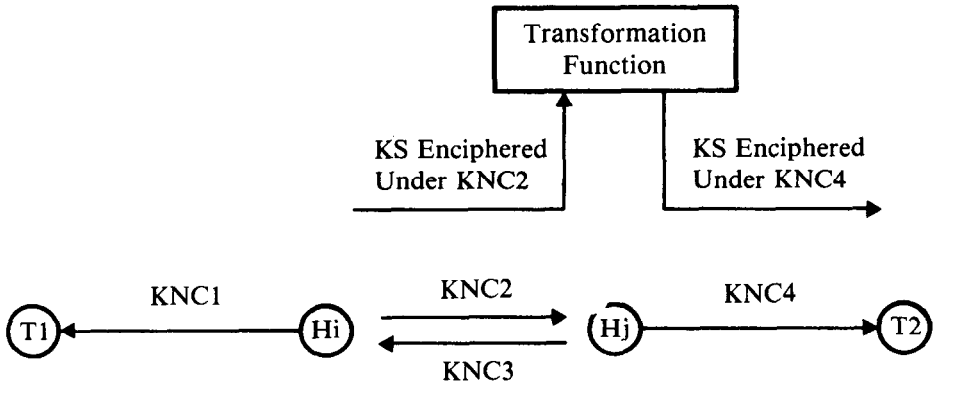


Figure 4-23. Session Key Transformation—Host j

and the recovered session key is then stored in a secure area in the respective cryptographic facilities (Figure 4-24).

At the end of the session initialization process, the communicants, 1 and 2, have a common session key (KS). Encrypted data can now be sent and received, thus completing the communication protocol.

Using a similar approach, an application program executing in one host can encipher data that are to be written to a portable storage medium, such as a reel of tape. The same data, after having been transported to another host, can be read and deciphered by a second application program.

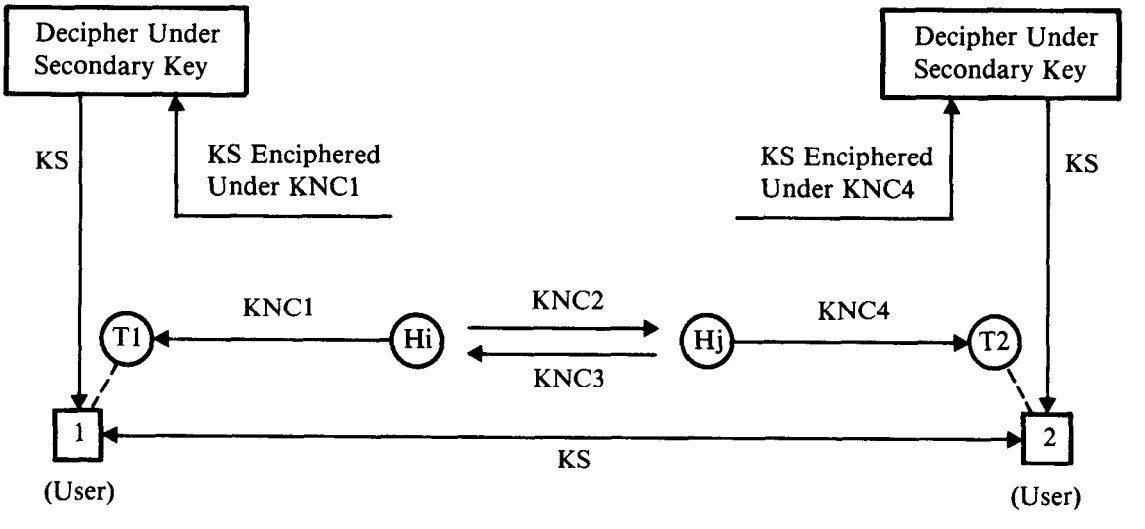
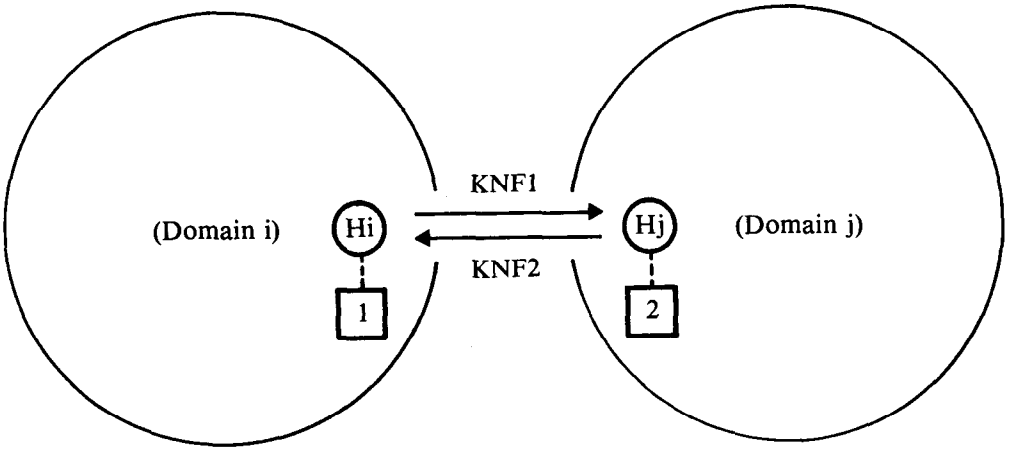


Figure 4-24. Session Key Recovery at Terminal 1 and Terminal 2



Note: It is assumed in this and subsequent Figures that KNF1 ≠ KNF2.

Figure 4-25. Initial Configuration

An Example of the Encryption of a Data File

Suppose that application program 1, executing in host i, creates an enciphered file for transmission to host j, and that at host j, the file is deciphered by application program 2. The network configuration and allocation of secondary keys are as shown in Figure 4-25.

Application 1 initiates the procedure by requesting a file key from host i. In response, host i generates a random file key, KF, and provides this value in a form that can be used by host i's cryptographic facility. In addition, a second copy of the file key is enciphered under a secondary file key, KNF1, and this value is written to the file's header record. This latter step allows the enciphered file to be recovered at its intended destination. Once the file has been enciphered, it is transported to host j. The entire process is illustrated in Figure 4-26.

At host j, the enciphered file key is read from the file header, and KF is recovered by deciphering under KNF1. The resulting value of KF is stored in a secure area of host j's cryptographic facility, where it is then used to decipher the enciphered file, as illustrated in Figure 4-27.

THE CRYPTOGRAPHIC FACILITY

The basic cryptographic operations of key transformation and data ciphering are performed by a cryptographic facility (Figure 4-28). The cryptographic facility is a secure implementation containing a conventional cryptographic algorithm (DES is assumed) and storage for a small number of key and data parameters. It can be accessed only through inviolate interfaces (secure

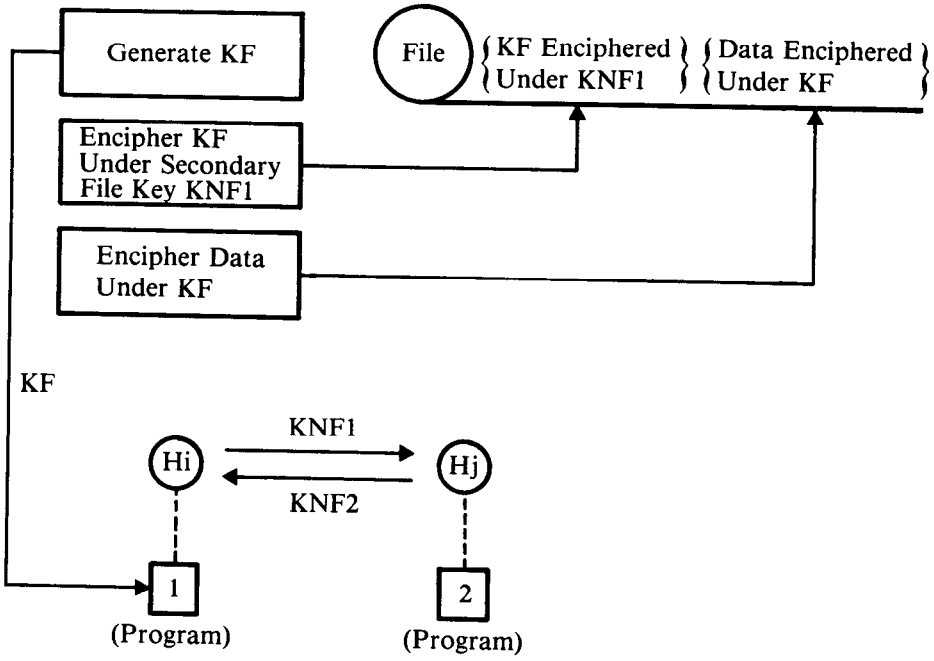


Figure 4-26. File Key Generation and Encipher Data Operations—Host i

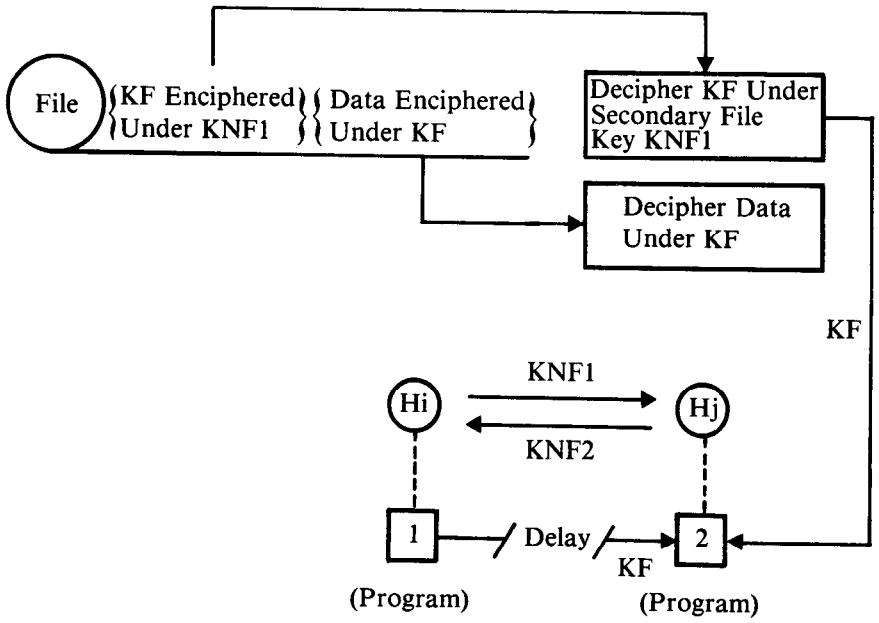


Figure 4-27. File Key Recovery and Decipher Data Operations—Host j

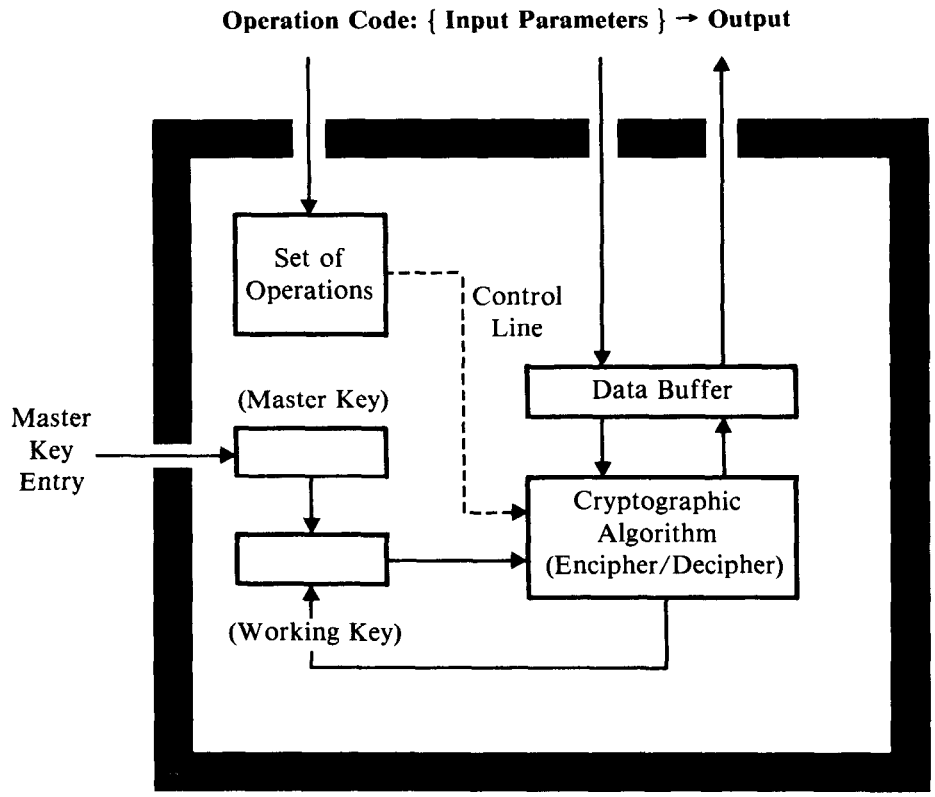


Figure 4-28. Cryptographic Facility

against intrusion, circumvention, and deception) which allow processing requests (via the indicated control line), key, and data parameters to be presented, and transformed output to be received. This strategy ensures that *clear keys and results of intermediate rounds of encipherment and decipherment are kept secret*. Access to the cryptographic facility can be controlled by a combination of physical and logical means that vary depending on whether the cryptographic facility is implemented in hardware or software.[13] Cryptographic operations are described using the following notation:

Operation Code: {Input Parameters} ⟶ Output

A cryptographic facility is assumed present at every host and terminal using cryptography. (It is assumed that useful processing of encrypted data is not possible.) However, since a terminal receives, but never produces enciphered keys, the cryptographic operations required at a terminal are different from those required at a host (see Basic Cryptographic Operations).

The cryptographic facility contains storage for both a master key and a working key. The working key, which is the key actively used by the cryp-

[13] Hardware is emphasized because it is more secure than software and it is required, per FIPS Publication 46 [10] for equipment procured for U.S. Government use.

tographic algorithm at any given time, normally changes from operation to operation. The master key, on the other hand, is a permanently stored quantity, and is maintained in a nonvolatile memory (i.e., a memory under battery power to prevent its loss when power to the device is shut off). A volatile memory (erased when power to the device is interrupted) or data buffer is also provided for the storage of input key and data parameters, intermediate results, and transformed output.

For simplicity, certain nonessential or unimportant elements (e.g., set of operations, control line, key and data storage elements) are omitted in subsequent figures depicting the cryptographic facility. Rather, emphasis is placed on the sequence of encipher and decipher operations required to perform a cryptographic operation and the key and data parameters used therein.

The cryptographic facility performs only two primitive operations: encipherment and decipherment. That is, the contents of the data buffer are either enciphered or deciphered under the cryptographic key contained in the working key storage, as illustrated in Figure 4-29.

Encipherment is denoted by the letter **E** and decipherment by the letter **D**. Each operation has two inputs (the data to be enciphered or deciphered and the working key) and a single output (the transformed data). The cryptographic algorithm is represented by a box, and by convention, the key enters from the left, data enter at the top, and output exits at the bottom. The notation used to express these operations is

$$\mathbf{E}_K(\mathbf{X}) = \mathbf{Y}$$

which means that ciphertext Y is produced by the encipherment of plaintext X under key K, and

$$\mathbf{D}_K(\mathbf{Y}) = \mathbf{X}$$

which means that plaintext X is produced by the decipherment of ciphertext Y under key K.

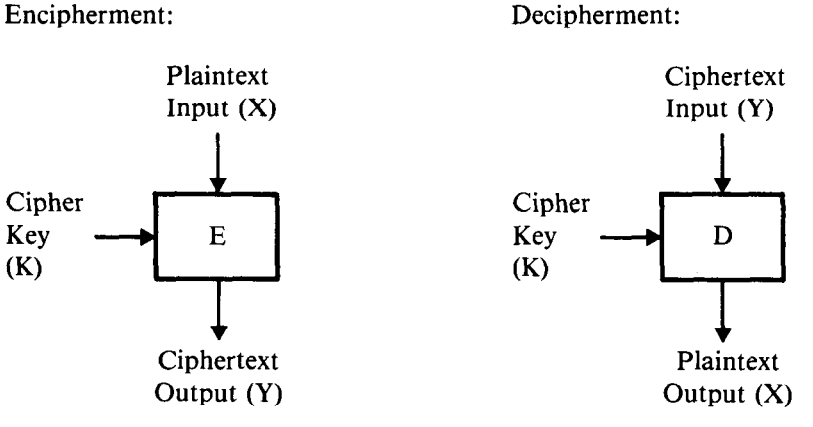


Figure 4-29. "Primitive" Operations of Encipherment and Decipherment

The basic cryptographic operations invoked through the programmed interface are performed by repeated executions of the primitive operations of encipherment and/or decipherment. During this process, the master key may be used as the working key if it is first transferred to the working key storage. In the same manner, an output from one operation may be used as an input (data or working key) to the next operation (described in more detail in the next section, Cipher Key Protection). Accordingly, when a series of encipher/decipher executions is required to satisfy a request, the contents of the working key storage will typically change as each encipher and decipher operation is performed.

Insertion of the master key into the cryptographic facility may be accomplished by a direct manual process, in which case a key is entered from mechanical switches, dials, or from a hand-held device, and is read directly into the master key storage. Alternatively, insertion of the master key may be accomplished by an indirect entry process, in which case a key entered from a nonvolatile medium, such as a card with a magnetic stripe, or entered through a keyboard-entry device, is first read into main memory and is then transferred to the master key storage. In this case, the residual copy of the master key in main memory must be erased (overwritten) after transfer to the master key storage is complete.

The master key is only used by the cryptographic facility to encipher and decipher other keys, and therefore is classified as a key-encrypting key. Being stored in nonvolatile memory, it can remain in use for long periods of time without having to be reentered.

CIPHER KEY PROTECTION

Because the cryptographic algorithm is assumed to be nonsecret, the degree of protection provided by the cryptographic system depends on how well the secrecy of the cryptographic keys is maintained. Therefore, the objective of sound key management is to ensure that *cryptographic keys never occur in clear form outside the cryptographic facility*, except under secure conditions during the period when keys are first generated, distributed, and installed, or when they are stored for backup or recovery in a safe or vault.

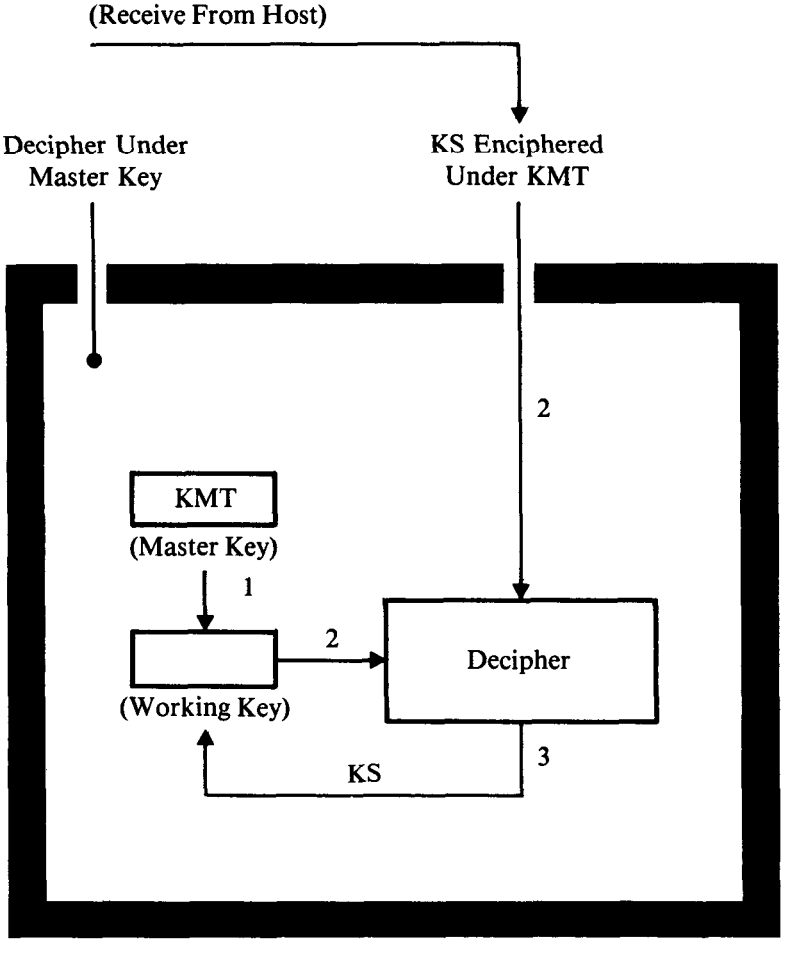
Protection of Terminal Keys

Since only one secondary communication key needs to be stored at a terminal (the one shared with its owning host), this key can be stored directly in the master key storage of the cryptographic facility. (Accordingly, the secondary communication key (KNC) used by a terminal is referred to as the *terminal master key*, **KMT**.) In effect this means that the terminal's cryptographic operations are less complicated than those required by the host, since the terminal must manage only a single secondary communication key. On the other hand, the host must manage the activities of the entire cryptographic system and therefore must cope with greater numbers of secondary and primary keys. In such a system, the terminal plays a passive role by responding

to requests made by the host, while the host plays a more active role by managing and initiating requests.

As previously stated, a terminal master key provides a means for the host to transmit securely to a terminal a primary communication key (KC), referred to as a session key (KS). The process of recovering a session key at a terminal is illustrated in Figure 4-30.

First the terminal master key is copied from master key storage to working key storage, and the enciphered session key is accepted as input to the cryptographic algorithm (step 1). The cryptographic algorithm deciphers the input



KMT = Secondary Communication Key (Terminal Master Key)
KS = Primary Communication Key (Session Key)

Note: To improve readability, some nonessential elements of the cryptographic facility are omitted in this and subsequent figures.

Figure 4-30. Session Key Recovery at the Terminal's Cryptographic Facility

data, under control of the terminal master key located in the working key storage, to obtain the clear session key (step 2). The session key is then transferred to the working key storage, replacing the terminal master key (step 3). The terminal can now initiate requests for data to be ciphered using the session key.

Protection of Host Keys

Because the host system has a greater role than a terminal in managing and controlling the operation of the cryptographic system, proportionately more demands are placed upon it to store and protect many different cipher keys (see Cipher Key Allocation).[14] Likewise, it becomes a more desirable target to an opponent.

Since several primary keys may be in use at any given time (typically, one per application program using cryptography), the host's cryptographic facility must be shared among the various applications. And when the working key storage will accommodate only one key, it is impractical to retain a primary key until its associated application program terminates. Instead, governed by the priority scheduling rules imposed by the host, the working key storage is regularly reinitialized with the primary key corresponding to the active application program. Thus primary keys must be protected when they are not being used by the cryptographic facility.

The Master Key Concept

One way to keep primary keys secret is to store them (in clear form) in a memory that can be read only by the cryptographic facility (i.e., a protected memory). However, an equally acceptable alternative is to encrypt keys and control their use via the host system. In the approach suggested here, all primary keys stored outside the cryptographic facility are enciphered under a single *host master key* (**KM0**) stored within the cryptographic facility. Therefore, before it can be used, an enciphered primary key, $E_{KM0}(K)$, must first be deciphered under KM0.

The concept of using one key to protect many other keys is defined as the *master key concept*. Basically, the problem of providing secrecy for a large number of cipher keys is reduced to that of providing secrecy for only a single key. More generally, cryptography reduces the problem of protecting large amounts of information (the plaintext) to that of a small amount of information (the cryptographic keys).

Encrypted vs. Unencrypted Primary Keys

When encrypted primary keys are used, an opponent must either compromise the security of the cryptographic facility containing KM0 or gain access to the system and invoke the decipher data operation, specifying $E_{KM0}(K)$ and data enciphered under K as input parameters. (It is assumed that a cryp-

[14]©1979 IEEE. Reprinted from NTC 79 Conference Record, November 27–29, 1979, Washington D.C. [9]

tographic operation designed to produce a clear key does not exist.[15]) Unencrypted primary keys, on the other hand, offer no protection if they become known.

In a network with multiple hosts, good security (i.e., independence among hosts) dictates that the master key at each host (KM0) should be different from that at any other host, or equal only by pure chance. Thus the points at which an encrypted primary key can be attacked are further reduced.

Not only must the secrecy of keys be protected, either by storing them in a protected memory or by enciphering them, but key usage must also be controlled, that is users must be able to access and use only those keys for which they are authorized.[16] Cryptography alone does not provide a solution to this problem, even though it provides effective measures to augment system services (see Authentication Techniques Using Cryptography, Chapter 8). Instead, security features provided by the host processor hardware and the host operating system must be used in conjunction with cryptography to ensure the security of data when resident in the host's main memory. Examples of hardware security features include store and fetch protection and special operations that may be used only by a supervisory program (or when the host processor is operating in the supervisory state).

Multiple Master Keys

Secondary keys stored at the host system can also be protected through encryption. These enciphered keys are stored in a data set accessible only to the cryptographic system (see The Host Cryptographic System). However, enciphering the keys using the host master key is not sufficient for protection since it allows primary keys to be recovered in clear form via selected cryptographic operations (see The Host System Cryptographic Operations, Chapter 5). To prevent this, a second host master key, KM1, is used. A third host master key, KM2, is defined to guarantee the unidirectional property of secondary communication keys (used to encipher primary keys while they are routed through a communications network). Thus three master keys are required: KM0, KM1, and KM2.

Note that multiple master keys are broadly applicable to cryptography since they allow the cryptographic operations for one application or purpose to be separated from those used for another (see Partitioning of Cipher Keys). The discussion of multiple master keys presented in this section has been kept brief deliberately and is intended only to give the reader enough information to allow the hierarchical structure of keys and the system's cryptographic operations to be discussed. The full justification for having three separate master keys is presented in Chapter 5.

Within the cryptographic facility, storage could be provided for three

[15] To create such an operation, an opponent would have to modify or replace the cryptographic facility. It is assumed that adequate physical security measures are present to deny an opponent such an opportunity.

[16] In a cryptographic system which operates transparent to the user, the system provides the keys, not the user. In contrast, if private or personal keys are allowed, the user provides the key, not the system.

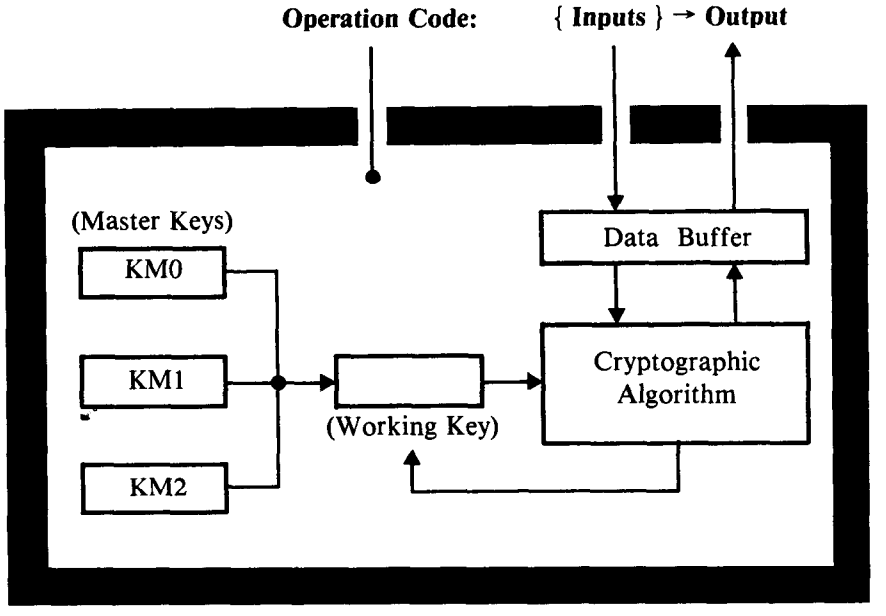


Figure 4-31. An Implementation Using Three Independent Host Master Keys

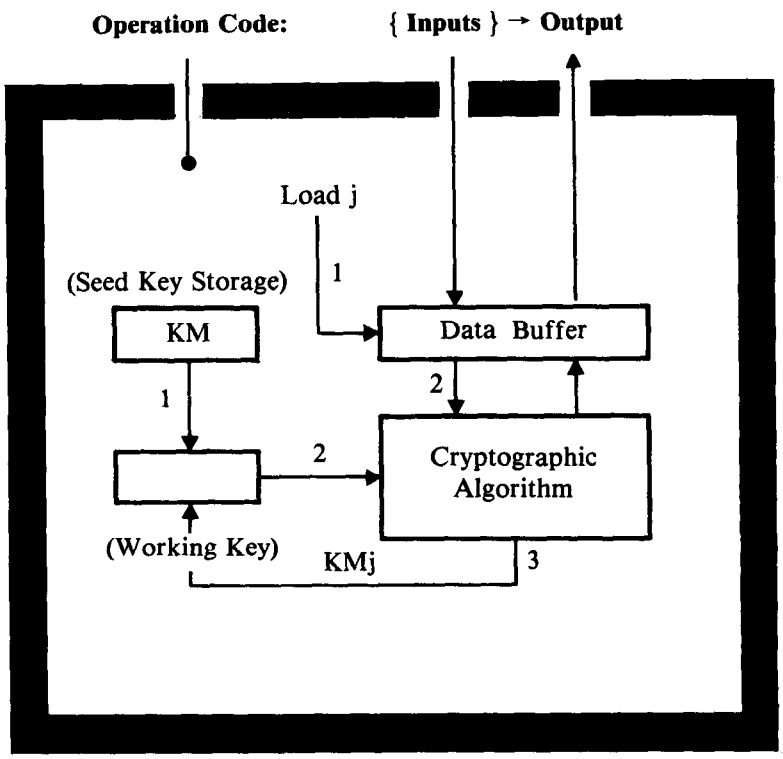
independent master keys (Figure 4-31). One or more of these keys (KM0, KM1, and KM2) would be selected as the working key and used to perform a sequence of encipher and decipher operations, depending upon the particular operation code presented to the cryptographic facility.

In a different approach, the three master keys could be derived from a seed key, KM, stored in the cryptographic facility. This requires an additional computation, but less storage. The keys could be obtained, for example, by defining KM0 as the encipherment of 0 under the key KM, KM1 as the encipherment of 1 under key KM, and KM2 as the encipherment of 2 under key KM (Figure 4-32).

When a master key is needed, KM is copied to the working key storage, a value is then selected for j (0, 1, or 2 depending upon the particular operation specified), and j is placed in the data buffer (step 1). The contents (j) of the data buffer are enciphered under the key (KM) located in the working key storage to produce the desired master key, KMj (step 2). The computed master key is then returned to the working key storage, overwriting the value of KM (step 3).

Master Key Variants

Another procedure for implementing multiple master keys is to store KM0 in the cryptographic facility and then derive KM1 and KM2 from KM0, as needed, by simply inverting selected bits in KM0. In such an approach, KM1 and KM2 are called *variants* of the host master key.



j = 0, 1 or 2

Figure 4-32. Implementation in which Multiple Master Keys are Derived from a Seed Key

KM0: Host master key
KM1: First variant of KM0
KM2: Second variant of KM0

Different variants are derived by inverting different bits. Such an operation can be easily performed within a cryptographic facility at the time KM0 is transferred from master key storage to the working key storage (Figure 4-33). (Note that the particular bits inverted to produce KM1 and KM2 are not important to this discussion.)

Obviously, a knowledge of any one of the keys (KM0, KM1, or KM2) is equivalent to a knowledge of all the keys, and therefore, using variants of KM0 must, by definition, provide less overall security than using either three independent master keys or three dependent master keys derived from one seed key using an irreversible operation. However, the trade-off between key storage, key computation time, and key independence is a good one. The resulting key management scheme is strong in any case.

Note that using three closely related keys is worthwhile only if a high correlation between keys does not result in exploitable correlations in the

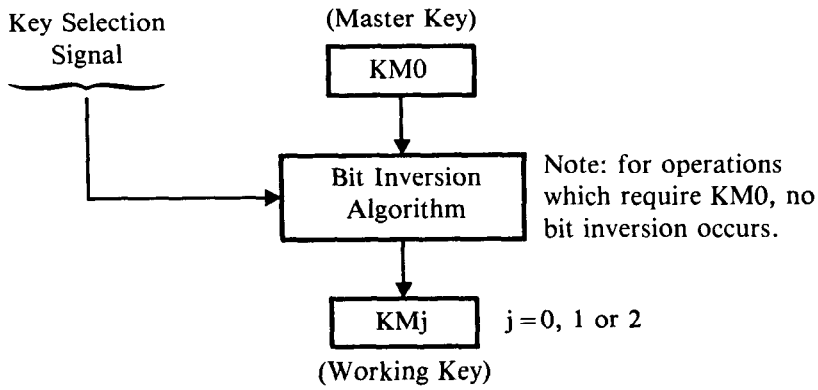


Figure 4-33. Derivation of Variants within the Cryptographic Facility Using Selected Bit Inversion

ciphertext which is produced by enciphering the same plaintext. Since, for DES, a single bit change in the key has a drastic effect on the ciphertext (see Analysis of Intersymbol Dependencies for the Data Encryption Standard, Chapter 3), there is no computationally feasible way to deduce one enciphered value from another. For example, given $E_{KMx}(K)$, for x = 1, 2, or 3, where KMx and K are unknowns, it is not presently possible to compute, deduce, or otherwise infer $E_{KMy}(K)$, for y = 1, 2, or 3 and $y \neq x$.

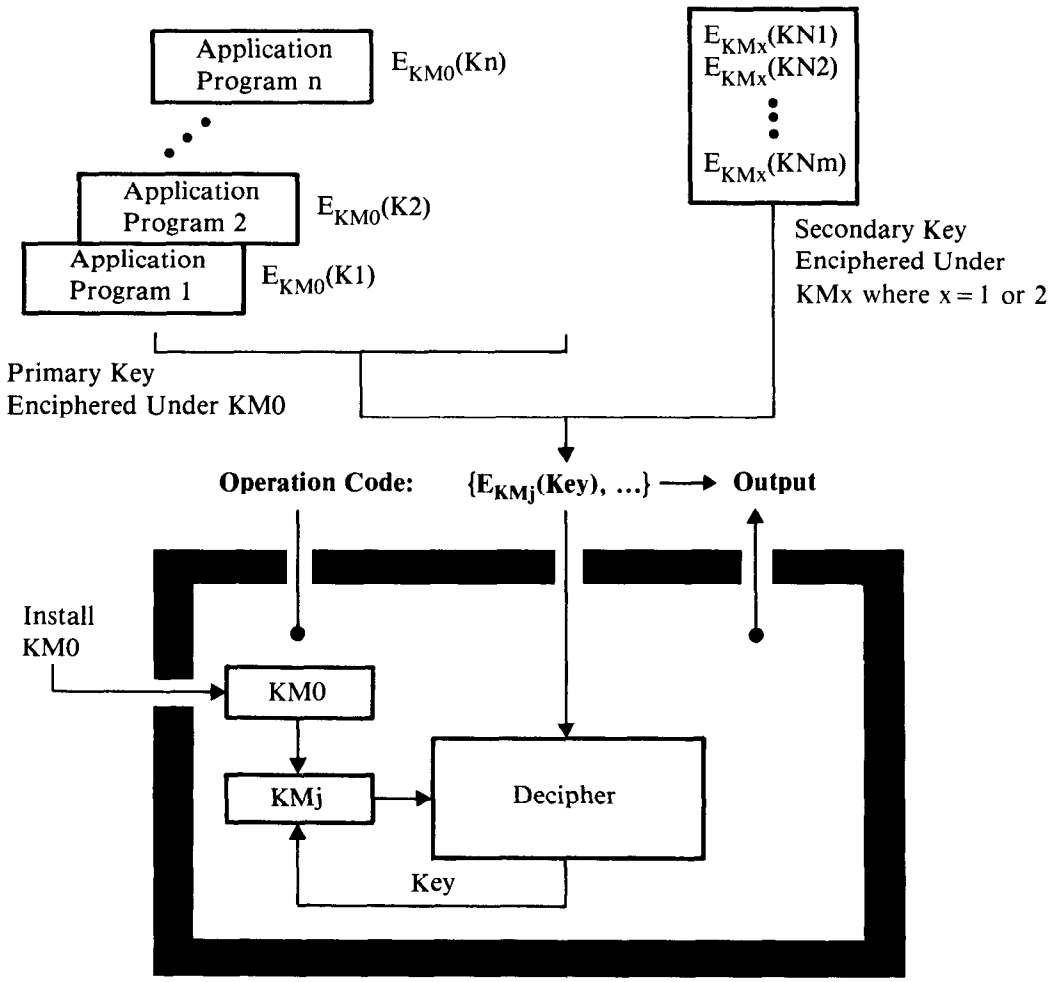
Summary

Figure 4-34 illustrates the overall scheme for protecting cipher keys at the host system.

Hierarchy of Cipher Keys

The discussion of cipher key allocation and protection implicitly defined a hierarchy of cipher keys and of key protection. This hierarchy is illustrated in Figure 4-35.

Large amounts of data are protected through the use of a smaller number of dynamically generated data-encrypting keys (primary keys), and the data-encrypting keys are protected through a still smaller number of relatively constant key-encrypting keys (secondary keys) or with the host master key (KM0). The key-encrypting keys are, in turn, protected with the variants of the host master key (KM1 and KM2). As a consequence, only a small number of keys need to be stored in clear form within the cryptographic facility, whereas the remainder of the keys can be stored outside the cryptographic facility in enciphered form. The various cipher keys defined to the cryptographic system are summarized in Figure 4-36, by name, category, use, and item protected (key or data).



KM0 = Host Master Key
KM1 = First Variant of the Host Master Key
KM2 = Second Variant of the Host Master Key
j = 0, 1, or 2 Depending on the Requested Operation Code

Figure 4-34. Host Cipher Key Protection—Summary

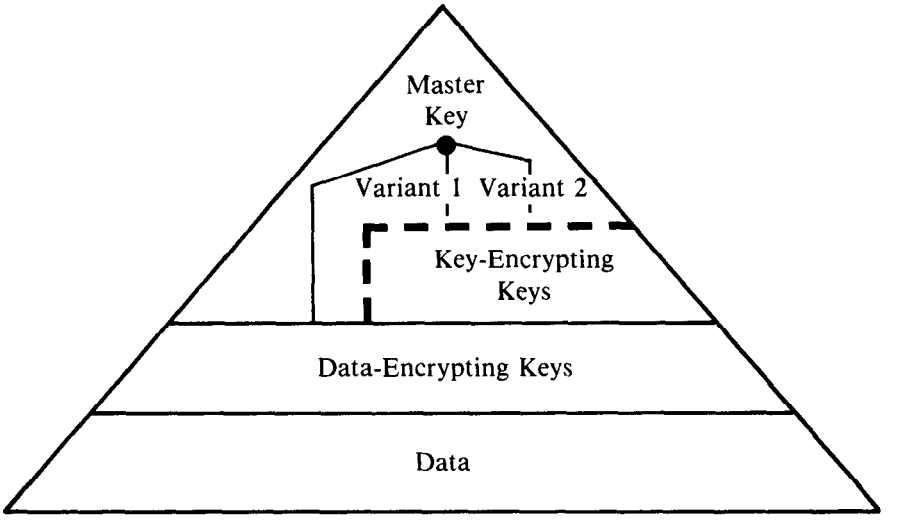


Figure 4-35. Hierarchy of Key Protection

THE HOST CRYPTOGRAPHIC SYSTEM[17]

Even for a modest-sized cryptographic system, the number of data-encrypting keys required over a period of time may indeed be quite large. Therefore, manual techniques for key generation and key management would soon prove to be inadequate. Automated procedures which are fast and eliminate security exposures resulting from human errors are required.

Fortunately, in this case, the computer itself provides the answer. Key generation and key management can both be effectively handled as computer applications. The net effect is that the computer augments cryptography by generating and managing the system's cipher keys, and in turn, cryptography protects the computer's data.

The host cryptographic system is comprised of three basic elements:

1. **Cryptographic Facility** The cryptographic facility, as previously described (Figure 4-28), is a secure implementation (hardware and/or software) containing the cryptographic algorithm (DES is assumed) and storage for a small number of key and data parameters. It can be accessed only through inviolate interfaces which allow processing requests (basic cryptographic operations), key, and data parameters to be presented, and transformed output to be received.
2. **Key Generator** The key generator is a computer program that

[17] The material in this and subsequent sections represents the details of one particular implementation [2]. This implementation illustrates the operation and interrelationships between the various components of an actual cryptographic system. Other designs are possible.

Category	Key Name	Use	Item Protected
Key-Encrypting Keys	Host Master Key (KM0)	Encipher Keys Actively Used or Stored at Host	Primary Key
	First Variant of Host Master Key (KM1)	"	Secondary Key
	Second Variant of Host Master Key (KM2)	"	Secondary Key
	Secondary Communication Key	Encipher Keys External to Host	Primary Communication Key
	Secondary File Key	"	Primary File Key
Data-Encrypting Keys	Primary Communication Key	Encipher or Decipher Data	Data in Motion
	Primary File Key	"	Data in Storage

KMT = Terminal Master Key (Secondary Communication Key Stored at Terminal)

KS = Session Key (Primary Communication Key that is Unique for Each Session)

Figure 4-36. Summary of Cipher Keys

creates the key-encrypting keys required by a host. Key-encrypting keys can also be specified by installation personnel. These keys are enciphered under a variant of the host master key, either KM1 or KM2, and then written in a file called the Cryptographic Key Data Set (CKDS). The CKDS resides on secondary storage (either disk or drum) and is assumed to be accessible by the cryptographic system during normal operations. A discussion of which secondary keys are enciphered under KM1 and which are enciphered under KM2 is given in the section called Key Management Macro Instructions. A label, or symbolic name, is associated with each enciphered key stored in the CKDS. The key

manager accesses a key in the CKDS by using the label of the key (i.e., the key's name). The label could, for example, be the name of a resource protected by the referenced key. The generated keys (in clear form) and their respective identifying labels are recorded on a second output medium, such as a printer listing or punched cards. This allows the keys to be distributed to other locations and installed within their nodes. It also permits a backup record of the keys to be kept in a secure repository (safe, vault, or the like). Whenever the existing host master key is changed, the key generator is also used to decipher the secondary keys on the CKDS and reencipher them under the appropriate variant of the new host master key.

3. **Key Manager** The key manager is a program that creates primary keys, accesses the CKDS using a key's name supplied as input, and exercises the cryptographic facility in response to requests for key translation operations. In short, the key manager is the resource manager for the host's cryptographic keys. It has programming interfaces through which processing requests, keys, and data are presented, and generated or transformed outputs are received.

Figure 4-37 provides an overview of the basic elements needed at a host to implement a cryptographic system; namely, cryptographic facility, key generator program, and key manager program. GENKEY and RETKEY, which denote programming interfaces to the key manager, are used by programs to request key translation functions. CIPHER, another programming interface to the cryptographic facility, is used by programs for ciphering data. GENKEY, RETKEY, and CIPHER are implemented as programming *macro instructions*.

A macro instruction is commonly used with programs written in the basic assembly language of a particular computer. It approximates a higher level language in that with one uniquely named statement a programmer can incorporate a predefined sequence of instructions needed to satisfy a particular (and normally repetitive) function. The substitution of assembler language instructions for the macro instruction is performed during the compilation of the source program. For additional flexibility, macro instructions can be customized through the use of input parameters to vary the sequence of, and values associated with, the substituted assembler language instructions.

The macro instructions GENKEY, RETKEY, and CIPHER, when placed on the system macro library, can be used by system programs (where transparent cryptography is desired) or by user application programs (where private cryptography is desired).[18]

In the particular implementation discussed here, GENKEY, RETKEY, and CIPHER are implemented through the use of four basic cryptographic operations defined to the host's cryptographic facility; namely, encipher data (ECPH), decipher data (DCPH), reencipher from master key (RFMK), and reencipher to master key (RTMK). Two additional operations, set master

[18] Restricted access to a macro is achieved by placing it on a private library, addressable only by authorized users.

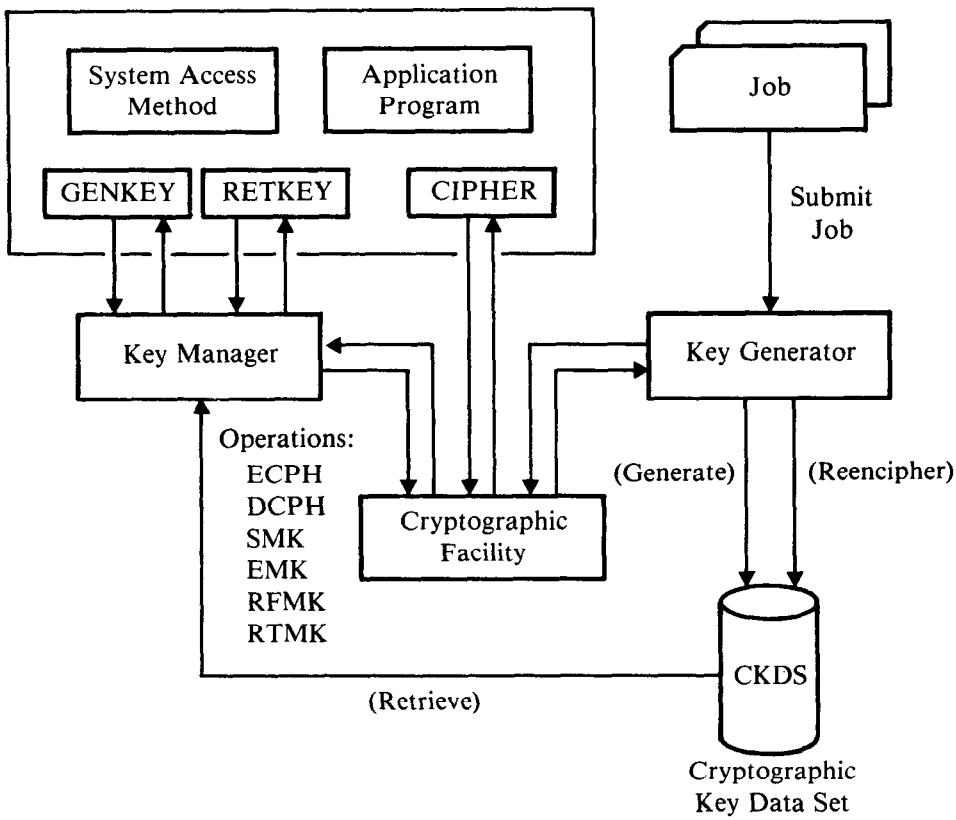


Figure 4-37. Host Cryptographic System

key (SMK) and encipher under master key (EMK), permit, respectively, installation of the host master key and encipherment of user-supplied data-encrypting keys (private or personal keys) under the host master key. (See The Host System Cryptographic Operations, Chapter 5.)

BASIC CRYPTOGRAPHIC OPERATIONS

There are two categories of cryptographic operations within the cryptographic system: those that transform data (data-ciphering operations), and those that initialize and transform cipher keys (key-ciphering operations). Since the host has an active role in managing the system's cipher keys (as compared to the terminal's less active role), the key-ciphering operations at the host system are more complex than those at the terminal.

In any case the cryptographic operations performed by a cryptographic facility are defined as *basic* since they are the most elementary operations that can be requested by the system. Recall that a cryptographic operation is described by its operation code, one or more input parameters (enclosed

in braces), and output (pointed to by a right arrow) (see also The Cryptographic Facility):

Operation: {Input Parameters} ⟶ Output

An input, as well as the operation's output, can be either a key or data parameter. Keys and data parameters may be in either clear or enciphered form. However, operations involving clear keys will ordinarily be exercised only when the system is sterile, that is, no other users or application programs are active.[19]

The basic cryptographic operations provide a means for implementing high level key management functions in the cryptographic system.[20] But, if not properly designed, they could allow an opponent to perform complex cryptographic transformations leading to the recovery of keys and data. Therefore, the cryptographic system must be designed so that if one or more of the operations are used together with any enciphered keys or data parameters that are routinely generated, routed, or stored within the system, it is not possible to recover:

1. Clear keys outside the cryptographic facility, regardless of the inherent security of the supporting host operating system.
2. Plaintext from ciphertext outside the cryptographic facility, except in the specific manner intended (using a decipher data operation), and under the specific conditions anticipated.

The same method used to validate the strength of a cryptographic algorithm (Chapter 1) is used to certify that cryptographic operations are dependable and secure. The process of subjecting cryptographic operations to a series of hypothetical attacks is called *threat analysis*. A favorable result (validation by threat analysis) leads to the conclusion that exposure of keys and data, although not provably impossible, is at least demonstrably difficult or unlikely.

As an aid to the reader in interpreting the meaning of the cryptographic operations described below, a summary of common abbreviations is given.

X: Plaintext block
Y: Ciphertext block
K: Primary key (data-encrypting key)
KN: Secondary key (key-encrypting key)
KM0: Host master key
KM1: First variant of host master key
KM2: Second variant of host master key

[19] For example, clear keys would exist at a host when the key generator program is executed or when a key is installed in the cryptographic facility. The key generator program might even be executed on a separate system, if the host's system programs cannot be trusted.

[20] They represent one way in which the high level key management functions can be achieved, but not the only way.

Recall that the notation $E_K(X)$ denotes the encipherment of plaintext X under key K, and the notation $D_K(Y)$ denotes the decipherment of ciphertext Y under key K.

Cryptographic Operations at a Terminal

The following basic cryptographic operations are used at a terminal:

Load Key Direct (LKD)
Write Master Key (WMK)
Decipher Key (DECK)
Encipher (ENC)
Decipher (DEC)

The LKD, WMK, and DECK operations are used for initializing and transforming keys, while ENC and DEC are used to transform data.

Since a terminal participates in only one communication session at a time, it is possible for the session key to be placed into the working key storage of the terminal's cryptographic facility at the beginning of a session, and remain there unchanged for the session's duration. The terminal's cryptographic operations are designed to avoid the overhead of reinitializing the working key storage whenever possible.[21]

LOAD KEY DIRECT

LKD: {K} ⟶ Load Cipher Key K into Working Key Storage

The LKD operation (Figure 4-38) is used to load a clear key into the working key storage of the terminal's cryptographic facility. No inverse operation is available for reading the working key. The LKD operation allows the terminal user to enter a private key which can be used for ciphering data. (This operation is not needed if only system-initiated session keys are used.)

WRITE MASTER KEY

WMK: {KN} ⟶ Write Cipher Key KN in Master Key Storage

The WMK operation (Figure 4-39) is used to write a key into the master key storage of the terminal's cryptographic facility. No inverse operation is available for reading the master key. The WMK operation may be exercised only in an authorized state. Such a state may be created by a physical key-operated switch which enables or disables the operation. (This is an example of an authorized state that is not automated, that is, the machine is placed in an authorized state manually by a designated user or security administrator. See also the Set Master Key operation.)

[21] The cryptographic operations for a terminal control unit (or cluster controller), which can service several terminals and terminal users concurrently, would need to be designed to accommodate several enciphered session keys, associating each with the corresponding session. The details of such a design are not provided in this discussion.

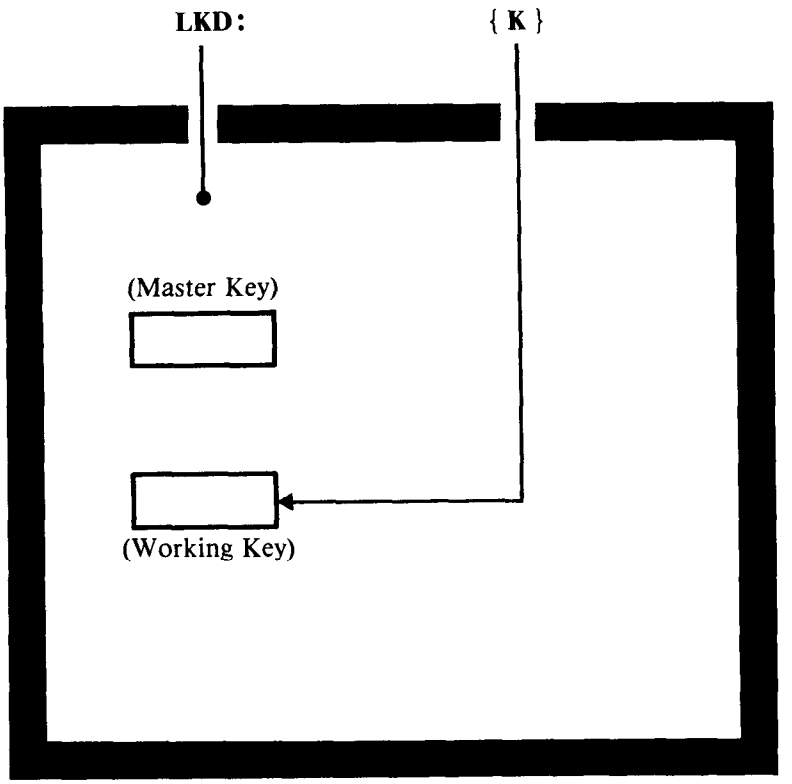


Figure 4-38. Load Key Direct Operation at Terminal

DECIPHER KEY

DECK: $\{E_{KN}(K)\} \longrightarrow$ Load Cipher Key K into Working Key Storage

The DECK operation (Figure 4-40) is used to decipher a key under the cipher key stored in the master key storage of the terminal's cryptographic facility, and place the result in working key storage. By definition, the input key parameter is a primary key enciphered under the terminal master key, and hence the value placed in the working key storage (as a result of the DECK operation) will be the intended data-encrypting key.

ENCIPHER

ENC: $\{X\} \longrightarrow E_K(X)$

The ENC operation (Figure 4-41) is used to encipher data. A 64-bit block of plaintext (X) is enciphered under the data-encrypting key (K) stored in the working key storage of the terminal's cryptographic facility. A 64-bit block of ciphertext, denoted by $E_K(X)$, is produced.[22] As long as the cryptographic

[22] The described implementation of DES is called the Electronic Code Book Mode (ECB). As a rule, block encryption (see also the DEC, ECPH, and DCPH operations) is used to protect keys. A different method, called chained block encryption (see CIPHER Macro Instruction), is used to protect data. (See also Block Ciphers with Chaining, Chapter 2.)

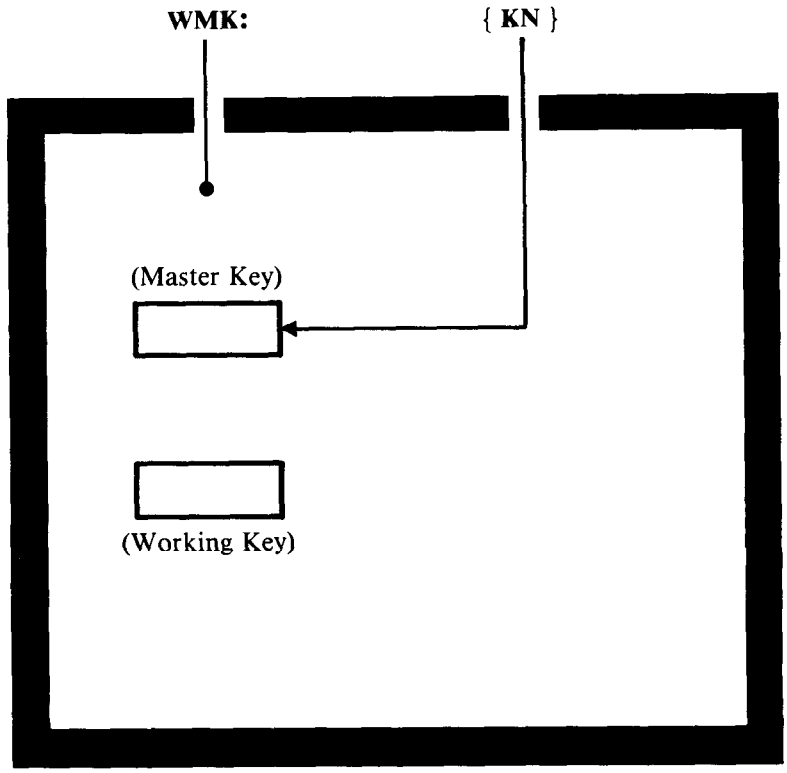


Figure 4-39. Write Master Key Operation at Terminal

facility's controls remain set to the encipher mode of operation, a message of multiple 8-byte blocks of plaintext can be enciphered as a series of steps in which each 8-byte block of plaintext is presented to the cryptographic facility in succession. Thus message encipherment can be expressed by the notation

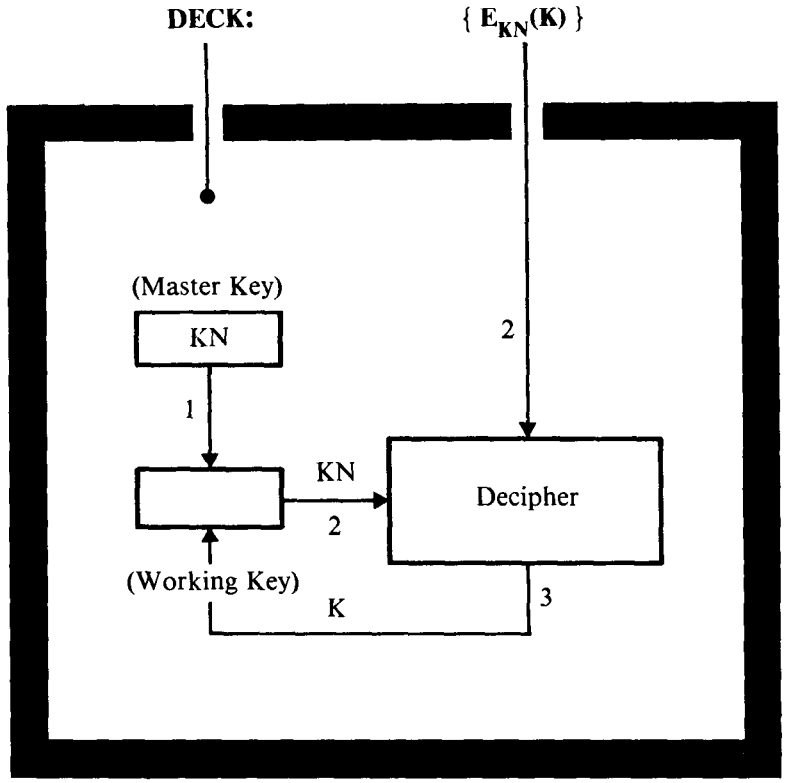
$$\text{ENC: } \{X(1), X(2), \ldots, X(n)\} \longrightarrow Y(1), Y(2), \ldots, Y(n)$$

where X(1), X(2), . . . , X(n) denotes a message of n 8-byte blocks of plaintext, Y(1), Y(2), . . . , Y(n) denotes the resulting ciphertext, and $Y(1) = E_K(X(1))$, $Y(2) = E_K(X(2))$, and so on.

DECIPHER

$$\text{DEC: } \{E_K(X)\} \longrightarrow X$$

The DEC operation (Figure 4-42) is used to decipher data. A 64-bit block of ciphertext (denoted by $E_K(X)$) is deciphered under the data-encrypting key (K) stored in the working key storage of the terminal's cryptographic facility. A 64-bit block of plaintext (X) is recovered. As long as the cryptographic facility's controls remain set to the decipher mode of operation, a message of multiple 8-byte blocks of ciphertext can be deciphered as a series of steps in which each 8-byte block of ciphertext is presented to the cryptographic



Note: The numbers indicate the order in which the individual steps are performed.

Figure 4-40. Decipher Key Operation at Terminal

facility in succession. Thus message decipherment can be expressed by the notation

$$\text{DEC: } \{Y(1), Y(2), \ldots, Y(n)\} \longrightarrow X(1), X(2), \ldots, X(n)$$

where Y(1), Y(2), . . . , Y(n) denotes a message of n 8-byte blocks of ciphertext, X(1), X(2), . . . , X(n) denotes the recovered plaintext, and $X(1) = D_K(Y(1))$, $X(2) = D_K(Y(2))$, and so on.

Once the session key has been placed into the working key storage using the DECK operation, there is no need to repeat the step of loading and deciphering the working key as a precondition of requesting subsequent encipher (ENC) and decipher (DEC) operations, since the session key is still present in working key storage. Hence all data ciphering operations can be performed using an implicit cipher key (the cipher key present in the working key storage), rather than an explicit cipher key (a cipher key supplied as an input parameter to the requested cryptographic operation).

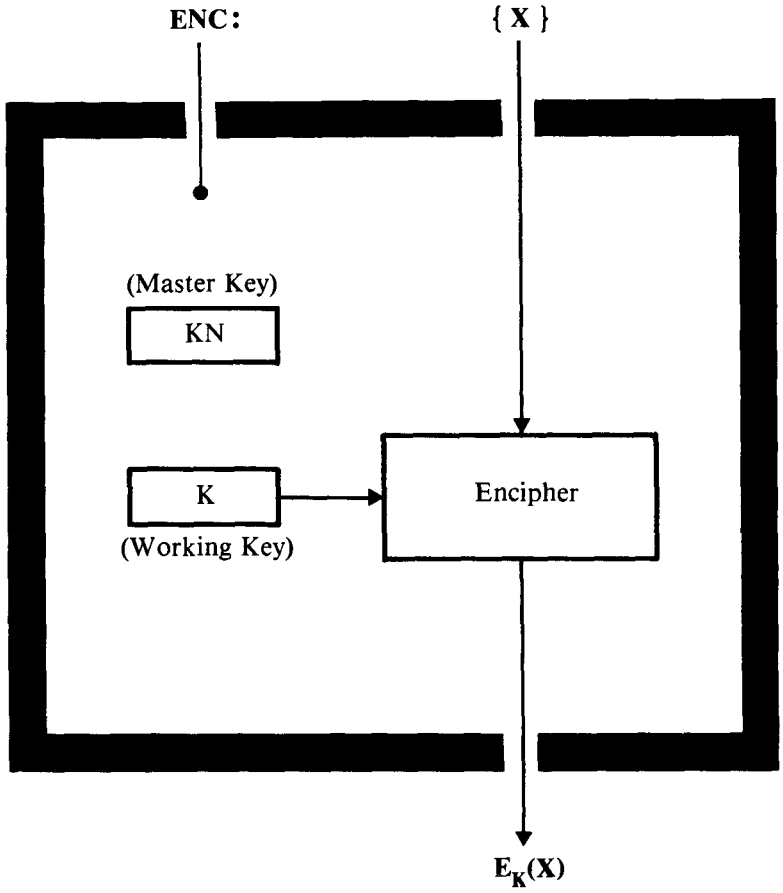


Figure 4-41. Encipher Operation at Terminal

Cryptographic Operations at a Host

There are two categories of basic cryptographic operations defined to the host's cryptographic facility, those that transform data:

Encipher Data (ECPH)
Decipher Data (DCPH)

and those that are used to initialize and transform keys:[23]

Set Master Key (SMK)
Encipher Under Master Key (EMK)

[23] In an actual implementation, the steps describing each cryptographic operation vary depending upon the number of storage elements within the cryptographic facility available for storing intermediate results produced by the operation.

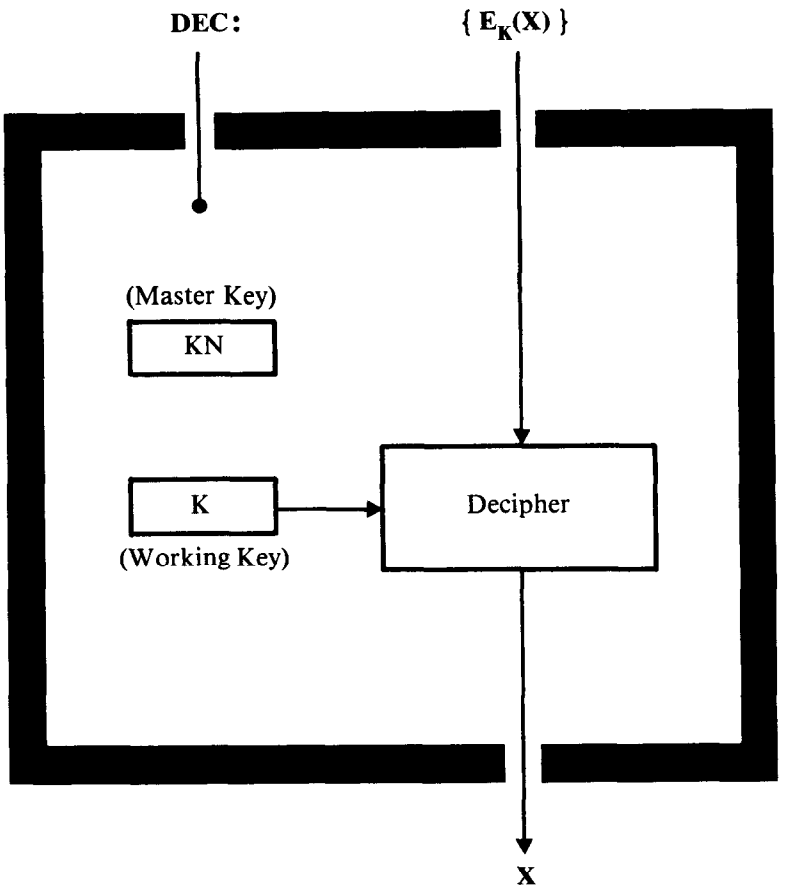


Figure 4-42. Decipher Operation at Terminal

Reencipher from Master Key (RFMK)
Reencipher to Master Key (RTMK)[24]

Data Ciphering Operations

Since several application programs may be involved in communication sessions at one time, the cryptographic facility must be shared among these several different users. In the approach described here, the primary key is provided as an input parameter to all requests for enciphering and deciphering data.

ENCIPHER DATA

$$\text{ECPH: } \{E_{KM0}(K), X\} \longrightarrow E_K(X)$$

The ECPH operation (Figure 4-43) is used to encipher data. A 64-bit block of plaintext (X) is enciphered under the data-encrypting key (K) to produce

[24] In an actual implementation, effective security could require that the RTMK operation be restricted to privileged programs (see Key Management Macro Instructions).

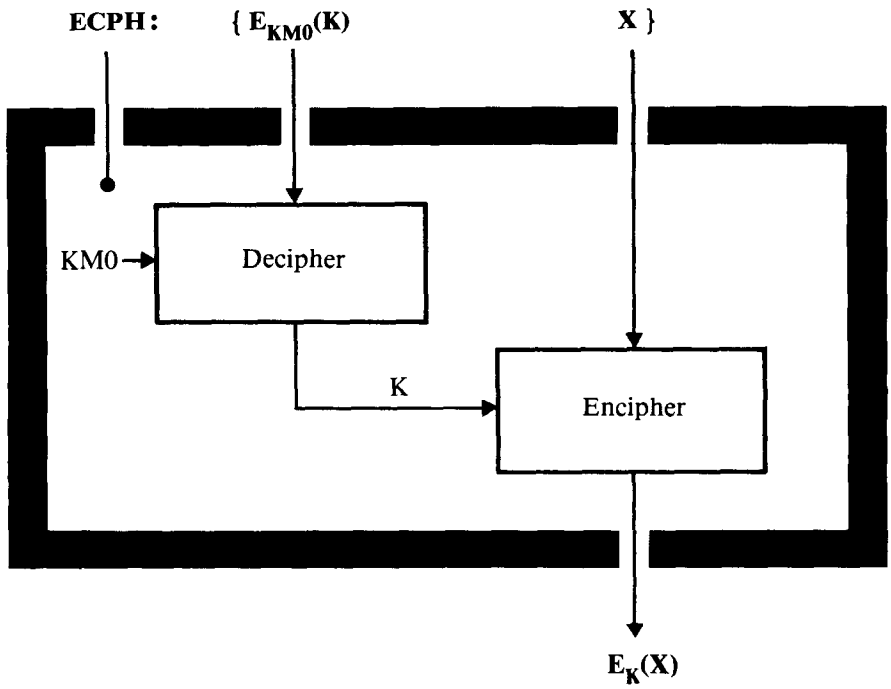


Figure 4-43. Encipher Data Operation at Host System

a 64-bit block of ciphertext, denoted by $E_K(X)$. As long as the cryptographic facility's controls remain set to the encipher data mode of operation, a message of multiple 8-byte blocks of plaintext can be enciphered as a series of steps in which each 8-byte block of plaintext is presented to the cryptographic facility in succession. Message encipherment can be expressed by the notation

$$\text{ECPH: } \{E_{KM0}(K), X(1), X(2), \ldots, X(n)\} \longrightarrow Y(1), Y(2), \ldots, Y(n)$$

where X(1), X(2), . . . , X(n) denotes a message of n 8-byte blocks of plaintext, Y(1), Y(2), . . . , Y(n) denotes the resulting ciphertext, and $Y(1) = E_K(X(1))$, $Y(2) = E_K(X(2))$, and so on.

DECIPHER DATA

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

The DCPH operation (Figure 4-44) is used to decipher data. A 64-bit block of ciphertext, denoted by $E_K(X)$, is deciphered under the data-encrypting key (K) to recover a 64-bit block of plaintext (X). As long as the cryptographic facility's controls remain set to the decipher data mode of operation, a message of multiple 8-byte blocks of ciphertext can be deciphered as a series of steps in which each 8-byte block of ciphertext is presented to the cryptographic facility in succession. Message decipherment can be expressed by the notation

$$\text{DCPH: } \{E_{KM0}(K), Y(1), Y(2), \ldots, Y(n)\} \longrightarrow X(1), X(2), \ldots, X(n)$$

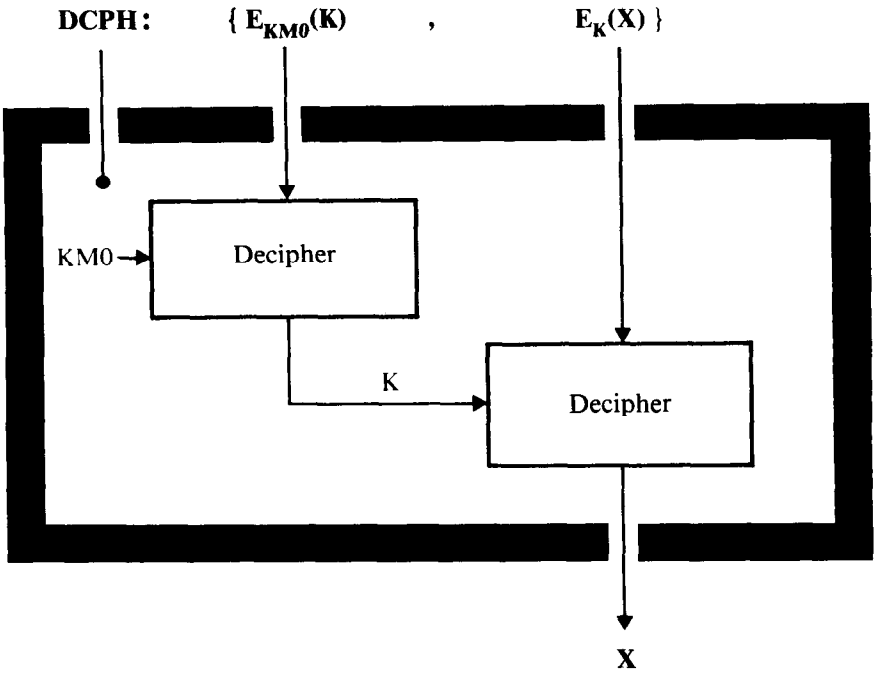


Figure 4-44. Decipher Data Operation at Host System

where Y(1), Y(2), . . . , Y(n) denotes a message of n 8-byte blocks of ciphertext, X(1), X(2), . . . , X(n) denotes the recovered plaintext, and X(1) = $D_K(Y(1))$, X(2) = $D_K(Y(2))$, and so on.

Key Management Operations

SET MASTER KEY

SMK: {KM0}⟶Write Cipher Key KM0 in Master Key Storage

The SMK operation (Figure 4-45) is used to write a key into the master key storage of the host's cryptographic facility. No inverse operation is available for reading the master key. The SMK operation can be invoked only in an authorized state. Such a state may be controlled by a physical key-operated switch which enables or disables the operation.

ENCIPHER UNDER MASTER KEY

EMK: {K} ⟶ $E_{KM0}(K)$

The EMK operation (Figure 4-46) is used to encipher a data-encrypting key under the host's master key (KM0). No inverse operation is available which allows decipherment under KM0. (A description of how keys can be enciphered under the variants of the host master key is given in Chapter 6.)

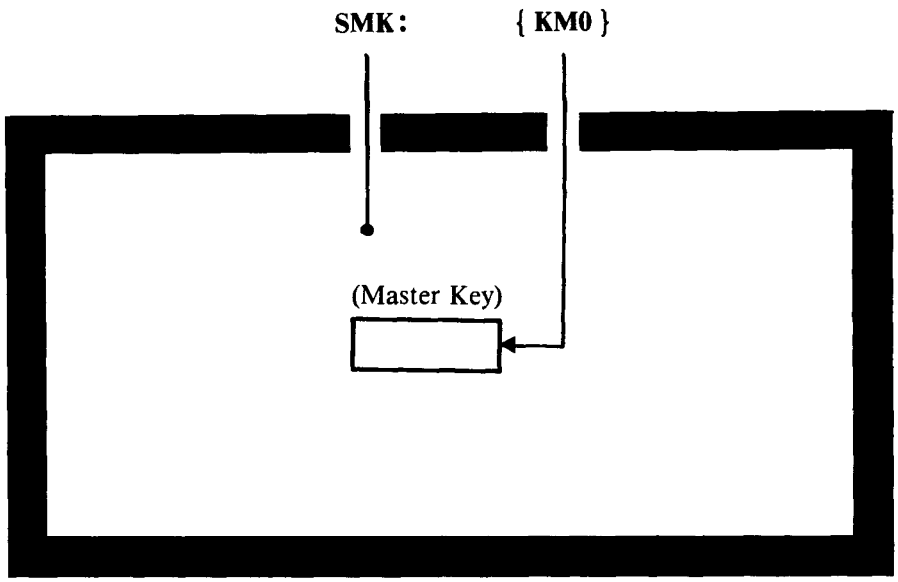


Figure 4-45. Set Master Key Operation at Host System

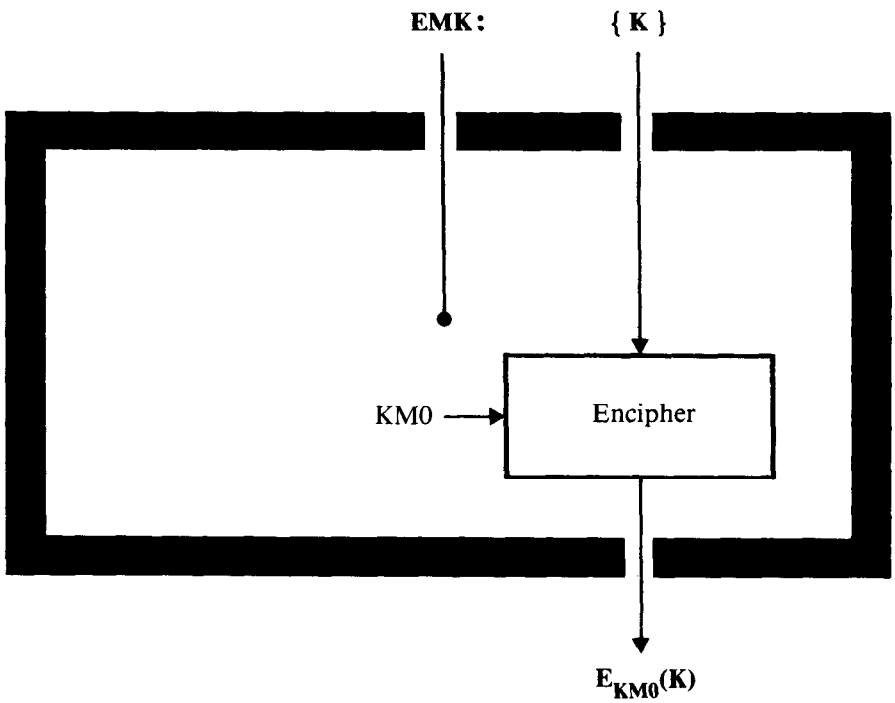


Figure 4-46. Encipher Under Master Key Operation at Host System

An EMK operation is provided at the host to support the use of personal (user-supplied) keys. Without EMK it would be impossible to encipher a personal key under the host master key so that it could be used to encipher and decipher data (see CIPHER Macro Instruction).

REENCIPHER FROM MASTER KEY

RFMK: $\{E_{KM1}(KN), E_{KM0}(K)\} \longrightarrow E_{KN}(K)$

The RFMK operation (Figure 4-47) is used by the key manager to transform a primary key (K) from encipherment under the host master key (KM0) to encipherment under a secondary key (KN).

REENCIPHER TO MASTER KEY

RTMK: $\{E_{KM2}(KN), E_{KN}(K)\} \longrightarrow E_{KM0}(K)$

The RTMK operation (Figure 4-48) is used by the key manager to transform a primary key (K) from encipherment under a secondary key (KN) to encipherment under the host master key (KM0).

Although it may appear that the RFMK and RTMK operations are the inverse of each other, this is true only if the same secondary key is enciphered under both KM1 and KM2. Two variants are introduced specifically so that the operations are not reversible (see Chapter 5). Translation of a key from

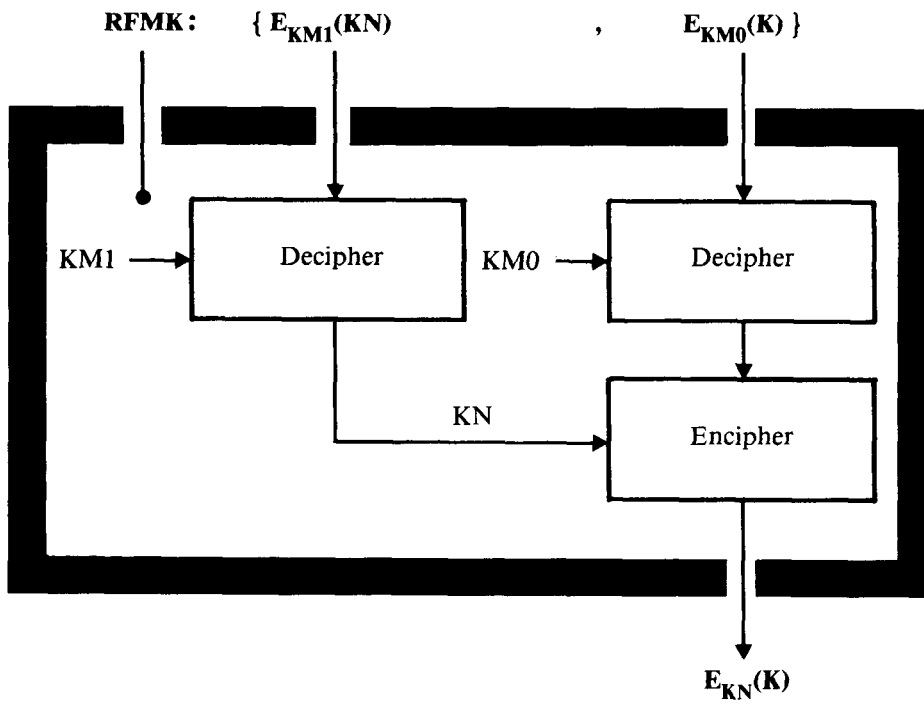


Figure 4-47. Reencipher From Master Key Operation at Host System

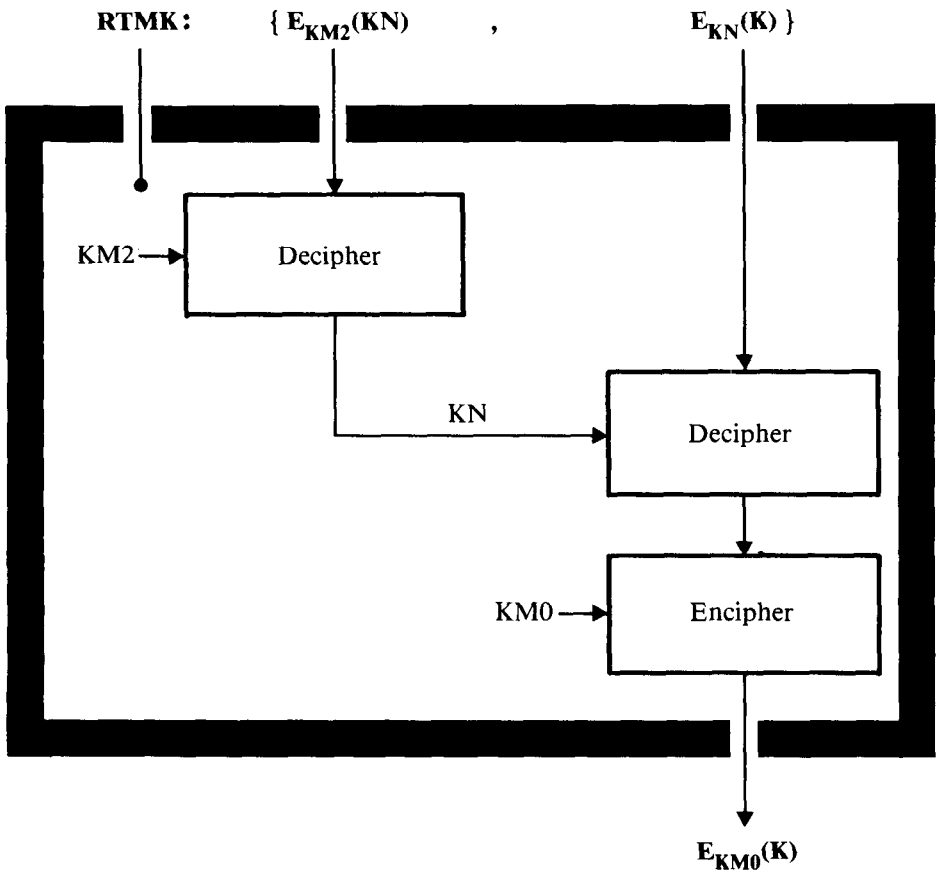


Figure 4-48. Reencipher To Master Key Operation at Host System

encipherment under one secondary key to encipherment under another secondary key would be accomplished using a combination of RTMK and RFMK.

Key Parity

A DES key consists of 64 bits of which 56 bits are used directly by the algorithm and 8 bits (the last bit of each 8-bit byte) can be used for error detection [9]. (See also The Data Encryption Standard, Chapter 3.) For example, the bits can be used to assure that each byte in the key has *odd parity* (i.e., that the number of 1 bits in each byte is odd). If used, keys are parity adjusted when they are created.

Since some of the concepts pertinent to key parity are applicable to this section, the topic is presented here. The topic of key generation is discussed in detail in Chapter 6.

Key-encrypting keys are generated in clear form. Prior to encipherment—in this case, with a variant of the host master key—each byte in the key can

easily be adjusted to have odd parity. When retrieved within the cryptographic facility, during execution of the various basic cryptographic operations, the clear key may be examined for correct parity. However, there is no firm requirement per FIPS Publication 46 [10] to test for parity, nor is there any recommended action that should be taken if parity is found to be incorrect. Therefore, parity checking remains an implementation option.

In the discussed implementation, data-encrypting keys are generated in an enciphered form. Thus, there is no way of knowing, short of deciphering the key and making a test, whether a generated key has correct parity or not. Furthermore, there is only a $1/2^8 = 1/256$ chance that correct parity will occur. Needless to say, ensuring correct parity for data-encrypting keys would be complex and time-consuming. Therefore, data-encrypting keys (as discussed here) are not parity adjusted.

Partitioning of Cipher Keys

The host master key (KM0) and its variants (KM1 and KM2) permit cipher keys to be separated or *partitioned* into functionally different groups.[25] Such partitioning ensures that the keys defined to one cryptographic operation, or set of cryptographic operations, cannot be used, misused, or manipulated meaningfully by another cryptographic operation, or set of cryptographic operations. This in turn is the basis for achieving isolation or independence among different cryptographic applications (e.g., communication security and file security).

Key partitioning may be achieved if the cipher keys used by a first operation are enciphered under the first variant of the host master key (KM1) and the cipher keys used by a second operation are enciphered under the second variant of the host master key (KM2). Assume that encipherment under the variants is restricted to personnel authorized to install keys, and that decipherment under the variants is possible only within the cryptographic facility where the variants are derived from the stored host master key (Figure 4-49).

Once a key has been enciphered under its appropriate variant (either KM1 or KM2) it can be specified as a parameter in a cryptographic operation. An enciphered key can be recovered only in the cryptographic facility. And only a key that has been specifically selected and enciphered in advance under one of the variants of the host master key can be recovered when specified as a parameter in a cryptographic operation. If a key not intended for use with a particular cryptographic operation is specified as a parameter of that operation, it will produce a final output that cannot be interpreted, understood, or used in a meaningful way.

This principle can be explained further through the use of a hypothetical cryptographic operation defined as OP1. OP1 has a single input parameter:

OP1: {**Key Parameter**} ⟶ **Output**

[25] ©1979 IEEE. Reprinted from *NTC 79 Conference Record*, November 27-29, 1979, Washington, D.C. [9].

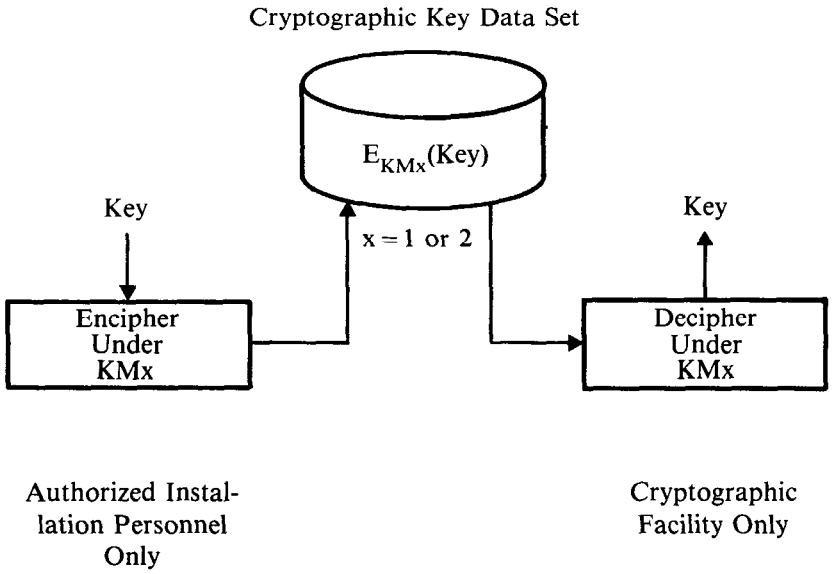


Figure 4-49. Encipherment and Decipherment Under the Variants of the Host Master key

and a set of secondary keys (KN1, KN2, . . . , KNn) that are to be used with it. Selected secondary keys are coupled to OP1 by enciphering them under the first variant of the host master key (KM1):

$$P1 = E_{KM1}(KN1)$$
$$P2 = E_{KM1}(KN2)$$
$$\vdots$$
$$Pn = E_{KM1}(KNn)$$

It is assumed that KM1 is used only with OP1. If E_{KM1}(KNi) is used as a key parameter in OP1, then KNi is recovered within the cryptographic facility. Afterwards, KNi participates in additional ciphering operations to produce a final output. (A precise specification of the additional ciphering operations is not important to the present discussion.)

If the input key parameter (P) is an element in the set E_{KM1}(KN1), E_{KM1}(KN2), . . . , E_{KM1}(KNn), then the recovered value of KN is a valid key known to the system (i.e., the recovered key-encrypting key is in the set KN1, KN2, . . . , KNn). However, if P is any other key parameter (e.g., a key enciphered under KM0 or KM2), or a key not in the set KN1, KN2, . . . , KNn, then the recovered value of KN is a key unknown to the system. The output of the cryptographic operation therefore involves a key whose value is both unpredictable and uncontrollable.

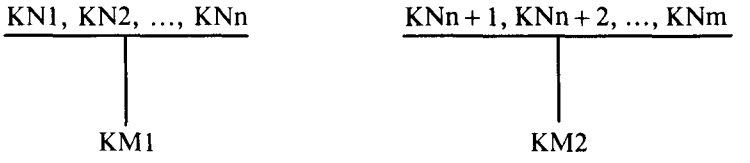


Figure 4-50. Hypothetical Scheme for the Protection of Keys

Suppose that a second hypothetical cryptographic operation is defined as OP2. OP2 also has a single input parameter:

OP2: {Key Parameter} ⟶ Output

and a set of cipher keys (KNn + 1, KNn + 2, . . . , KNm) that are to be used with it. However, in this case, the cipher keys to be used with OP2 are enciphered under the second variant of the host master key (KM2).

Encrypting KN1 through KNn with KM1, and encrypting KNn + 1 through KNm with KM2, cryptographically separates the secondary keys (Figure 4-50).

Since the results of OP1 or OP2 are meaningful only if the input key parameter is a key-encrypting key enciphered under KM1 or KM2, respectively, it is possible to eliminate shortcut methods of attack which either manipulate the cryptographic operations or use enciphered keys not intended for use with a specific operation. This point is illustrated below by showing the effect of using different key parameters with the example operations (Figure 4-51).

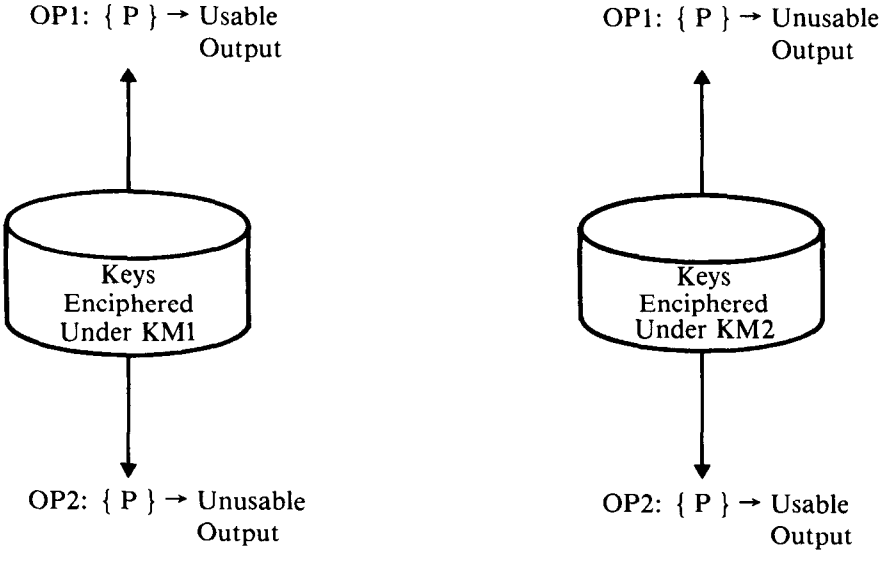


Figure 4-51. Correct and Incorrect Use of OP1 and OP2

An important advantage results from using multiple master keys; the cryptographic operations, higher-level functions, and applications provided by the cryptographic system can be isolated (logically separated) from one another. Cryptographic operations can also be made irreversible, thus limiting the function provided to the system and its users and thereby reducing the ways in which an opponent could attack the system. Whether implemented through variants of a single master key, or several different master keys, the concept of multiple master keys offers the cryptographic system designer flexibility in the development of provably secure cryptographic operations (see Extended Cryptographic Operations, Chapter 5).

CIPHER MACRO INSTRUCTION

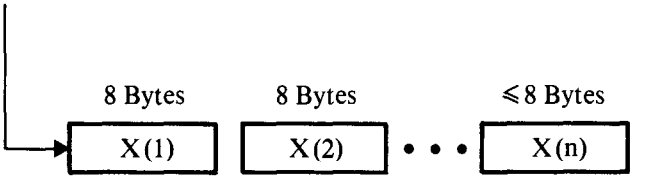
The reader is reminded that this and subsequent macro instructions represent only sample specifications for the purpose of illustration. A byte is defined as a sequence of 8 contiguous bits.

The CIPHER macro instruction allows users of the system to encipher and decipher data. The CIPHER macro is defined as follows:

```
name    CIPHER    PLNTXT  = address of plaintext
                  CHRTXT  = address of ciphertext
                  KEY     = address of enciphered primary key
                  LENGTH  = address of length parameter
                  FNC     = ENCPHR | DECPHR
                  ICV     = address of initial chaining value
                  CHAIN   = CHR | BLK | PLNCHR
                  OCV     = address of output chaining value
                  SHORT   = PAD | STREAM
```

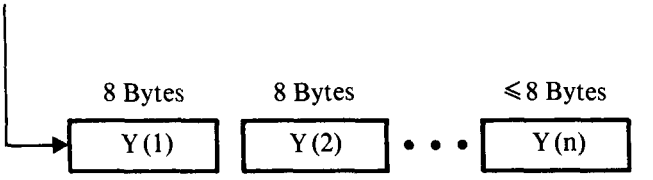
The symbol | denotes "or", whereas the underscore symbol indicates the defaulted parameter.

- PLNTXT = Address of Plaintext



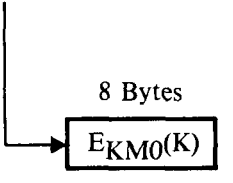
If the function specified is encipher (FNC = ENCPHR), then PLNTXT is the address of the data to be enciphered. If the function specified is decipher (FNC = DECPHR), then PLNTXT is the address of the storage location where the deciphered data are placed.

- CHRTXT = Address of Ciphertext



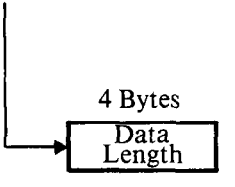
If the function specified is encipher (FNC = ENCPHR), the CHRTXT is the address of the storage location where the enciphered data will be placed. If the function specified is decipher (FNC = DECPHR), then CHRTXT is the address of the data to be deciphered.

- KEY = Address of Key Parameter (Primary Key Enciphered Under Host Master Key, KM0)



Data ciphering is performed using primary key K, where K is obtained by deciphering the input key parameter with KM0.

- LENGTH = Address of the Location Containing the Length of the Data in Bytes

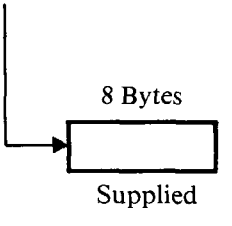


If FNC = ENCPHR, then LENGTH denotes length of plaintext. If FNC = DECPHR, then LENGTH denotes length of ciphertext.

- FNC = ENCPHR | DECPHR

ENCPHR specifies the function of encipherment and DECPHR specifies the function of decipherment.

- ICV = Address of Initial Chaining Value



When block chaining is used, a chaining value (CV) is required for each block of data to be ciphered. The CV used to cipher the initial block is called the initial chaining value (ICV). The ICV is a nonsecret, 8-byte, random or pseudo-random value which is supplied as input. It is also called the initialization vector Z (see Block Ciphers and Stream Ciphers, Chapter 2). Except for the ICV, all other chaining values are derived from information supplied or derived when ciphering takes place, and depend on the particular chaining scheme employed.

- CHAIN = CHR | BLK | PLNCHR

The function CHAIN = BLK specifies that ciphering is performed on a block-by-block basis, with no chaining. In this case, the LENGTH parameter must be an exact multiple of 8 bytes. Since the SHORT parameter is used only in situations where LENGTH is not a multiple of 8 bytes, SHORT is invalid when BLK is specified. The method of ciphering data using the BLK function is in Figure 4-52.

The function CHAIN = CHR (CHR being the default parameter for routine encipherment of data) specifies that block chaining with ciphertext feedback is used for data ciphering. The LENGTH parameter, in this case, does not have to be a multiple of 8 bytes. The method of ciphering data using the CHR function is in Figure 4-53.

Encipher		Decipher	
Input Plaintext	Output Ciphertext	Input Ciphertext	Output Plaintext
X(1)	$E_K(X(1))$	Y(1)	$D_K(Y(1))$
X(2)	$E_K(X(2))$	Y(2)	$D_K(Y(2))$
•	•	•	•
•	•	•	•
•	•	•	•
X(n)	$E_K(X(n))$	Y(n)	$D_K(Y(n))$

K is a Primary Key.
Input and Output Blocks are 64 Bits in Length

Encipherment:

ECPH: $\{ E_{KM0}(K), X(1), X(2), \ldots, K(n) \}$
$\rightarrow Y(1), Y(2), \ldots, Y(n)$

Decipherment:

DCPH: $\{ E_{KM0}(K), Y(1), Y(2), \ldots, Y(n) \}$
$\rightarrow X(1), X(2), \ldots, X(n)$

Figure 4-52. Ciphering Operation Using the BLK Function

Chaining Value	Encipher		Decipher	
	Input Plaintext	Output Ciphertext	Input Ciphertext	Output Plaintext
$CV(1) = ICV$	$X(1)$	$Y(1) = E_K(X(1) \oplus CV(1))$	$Y(1)$	$X(1) = D_K(Y(1)) \oplus CV(1)$
$CV(2) = Y(1)$	$X(2)$	$Y(2) = E_K(X(2) \oplus CV(2))$	$Y(2)$	$X(1) = D_K(Y(2)) \oplus CV(2)$
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•
$CV(n) = Y(n-1)$	$X(n)$	$Y(n) = E_K(X(n) \oplus CV(n))$	$Y(n)$	$X(n) = D_K(Y(n)) \oplus CV(n)$

Where: K is a Primary Key
Input and Output Blocks are 64-bits
$\oplus$ = Modulo 2 Addition (Exclusive-OR (XOR) Operation)

Encipherment:

ECPH: $\{ E_{KM0}(K), X(1) \oplus ICV, X(2) \oplus Y(1), \ldots, X(n) \oplus Y(n-1) \} \rightarrow Y(1), Y(2), \ldots, Y(n)$

Decipherment:

DCPH: $\{ E_{KM0}(K), Y(1), Y(2), \ldots, Y(n) \} \rightarrow D_K(Y(1)), D_K(Y(2)), \ldots, D_K(Y(n))$

Then XOR the Appropriate Chaining Value to Each Intermediate Block to Recover the Original Plaintext, Namely:

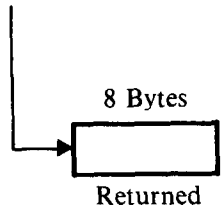
$D_K(Y(1)) \oplus ICV = X(1)$
$D_K(Y(2)) \oplus Y(1) = X(2)$
•
•
•
$D_K(Y(n)) \oplus Y(n-1) = X(n)$

Figure 4-53. Ciphering Operation Using the CHR Function

The function CHAIN = PLNCHR specifies that block chaining with plaintext-ciphertext feedback is used for data ciphering. The method of ciphering data using the PLNCHR function (Figure 4-54) is similar to that described in Figure 4-53, except that the chaining values are defined differently.

In the case of CHAIN = CHR, the chaining values are CV(1) = ICV, CV(2) = Y(1), . . . , CV(n) = Y(n − 1). In the case of CHAIN = PLNCHR, the chaining values are CV(1) = ICV, CV(2) = X(1) + Y(1) mod 2^{64}, . . . , CV(n) = X(n − 1) + Y(n − 1) mod 2^{64}.

- OCV = Address of Output Chaining Value



The OCV is a nonsecret, 8-byte, pseudo-random value which is computed from information supplied to or derived by the CIPHER macro at the time ciphering is performed. The method of computation depends upon the particular chaining scheme employed.

The OCV is defined here as the encipherment of the last block of ciphertext (Figure 4-55), although there are other ways in which an OCV could be derived.

- SHORT = PAD | STREAM

The SHORT parameter defines how short blocks (blocks of fewer than 8 bytes) are to be treated during the ciphering process. Basically, short blocks may be handled in two ways: they may be padded and then ciphered, or they may be ciphered directly with a stream cipher. The SHORT parameter is ignored for data that is a multiple of 8 bytes.

If SHORT = PAD is specified, then pad characters are added to a short block prior to its encipherment, and the pad characters are removed from the recovered plaintext block. Up to 7 pad characters can be added to a short block. The last pad character is a binary count of the total number of pad characters; the other pad characters are random or pseudo-random data (Figure 4-56).

If SHORT = STREAM is specified, then blocks 1, 2, . . . , n − 1 are ciphered using block chaining and block n (the short block) is ciphered using a stream cipher. The procedure for ciphering the short block consists of first enciphering the chaining value, CV(n), with the primary key K, and then

$$CV(1) = ICV$$

$$CV(2) = X(1) + Y(1)$$

$$\vdots$$

$$CV(n) = X(n-1) + Y(n-1)$$

Encipherment:

$$ECPH: \{ E_{KM0}(K), X(1) \oplus ICV, X(2) \oplus (X(1) + Y(1)), \ldots, X(n) \oplus (X(n-1) + Y(n-1))\} \rightarrow Y(1), Y(2), \ldots, Y(n)$$

Decipherment:

$$DCPH: \{ E_{KM0}(K), Y(1), Y(2), \ldots, Y(n) \} \rightarrow D_K(Y(1)), D_K(Y(2)), \ldots, D_K(Y(n))$$

Then XOR the Appropriate Chaining Value to Each Intermediate Block to Recover the Plaintext Values, Namely:

$$D_K(Y(1)) \oplus ICV = X(1)$$
$$D_K(Y(2)) \oplus (X(1) + Y(1)) = X(2)$$
$$\vdots$$
$$D_K(Y(n)) \oplus (X(n-1) + Y(n-1)) = X(n)$$

Note that addition is modulo 2^{64}.

Figure 4-54. Ciphering Operation Using the PLNCHR Function

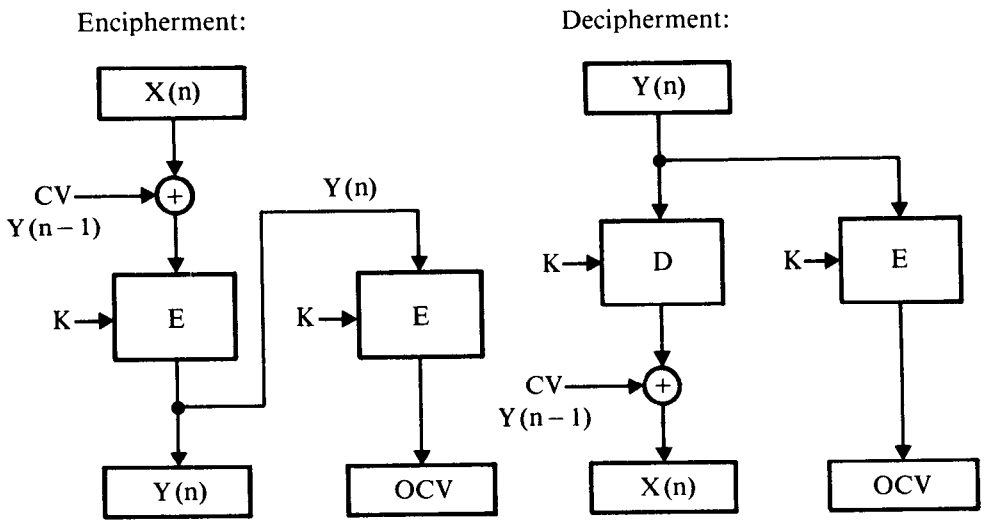


Figure 4-55. Procedure for Computing OCV

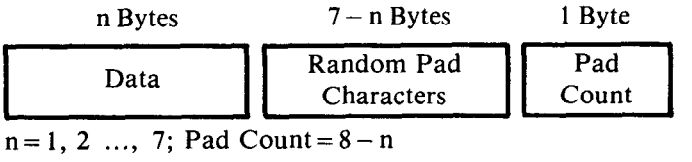


Figure 4-56. Padding of Short Blocks

Exclusive-ORing as many of the produced bits with the short block as necessary. Thus each bit in the short block is Exclusive-ORed with a corresponding bit in the enciphered chain value (Figure 4-57).

In the context of the present discussion, the SHORT parameter is invalid whenever CHAIN = BLK is specified. If CHAIN = CHR and SHORT = STREAM are specified, then the output chaining value is defined as

$$OCV = E_K(\text{RIGHT64}[CV(n) \,||\, Y(n)])$$

where RIGHT64 denotes a function that extracts the rightmost 64 bits from the bit string variable enclosed in brackets, and || denotes concatenation. By definition, $OCV = E_K(Y(n))$ whenever the length of X(n) is 64 bits. If CHAIN = PLNCHR and SHORT = STREAM are specified, then the output chaining value is defined as

$$OCV = E_K(\text{RIGHT64}[CV(n) \,||\, (Y(n) + X(n) \bmod 2^{64})])$$

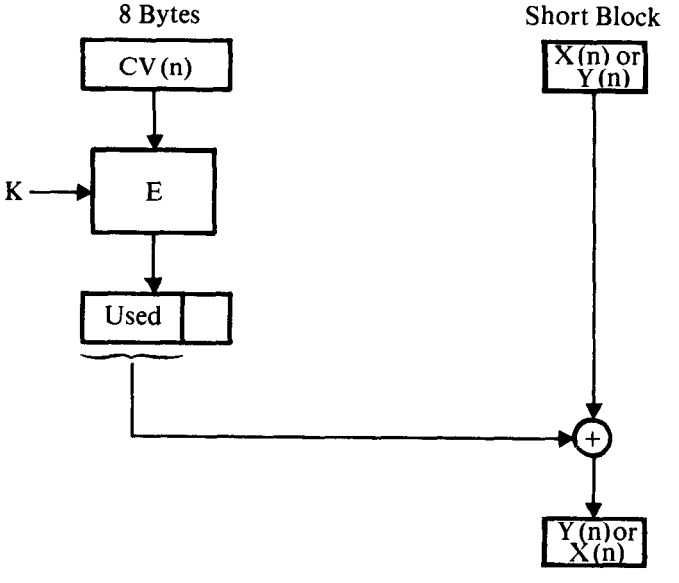


Figure 4-57. Ciphering a Short Block Using the STREAM Parameter

KEY MANAGEMENT MACRO INSTRUCTIONS

The key management macro instructions, GENKEY (generate key) and RETKEY (retrieve key), allow users of the cryptographic system to perform key transformations via the key manager program. In turn, the key manager program effects the desired key transformations by using one or more of the basic cryptographic operations. Used in the process are the key and data parameters specified by the macro, and other parameters stored in a key table that are available to the key manager program. The required transformations are performed by the key manager program using the cryptographic operations RFMK and RTMK, although in a different implementation they might be performed using a different set of cryptographic operations.

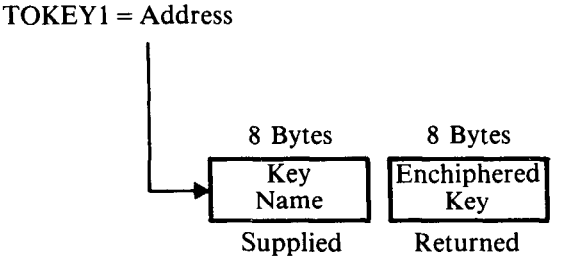
GENKEY and RETKEY Macros

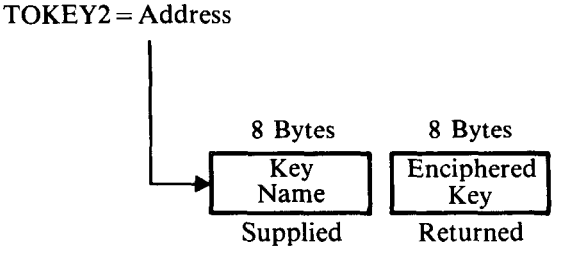
Possible specifications for the GENKEY and RETKEY macros are as follows:

GENKEY

name GENKEY TOKEY1 = address, TOKEY2 = address

The TOKEY1 and TOKEY2 parameters each specify the address of a 16-byte area that contains an 8-byte key name (the name of a cryptographic resource) and an 8-byte answer area for the generated key enciphered under the named key-encrypting key.





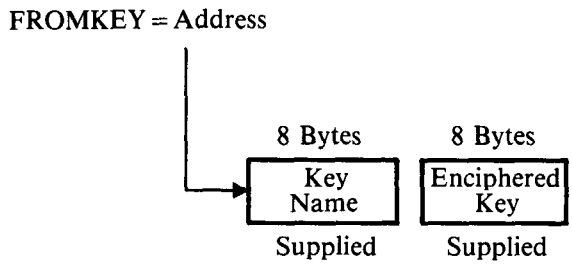
Execution of the GENKEY macro instruction causes the key manager to create a data-encrypting key (K) and to encipher this key under the key-

encrypting keys (KN1 and KN2) specified by the key names in the TOKEY1 and TOKEY2 parameters.[26] Each enciphered key is then placed in its respective 8-byte answer area.

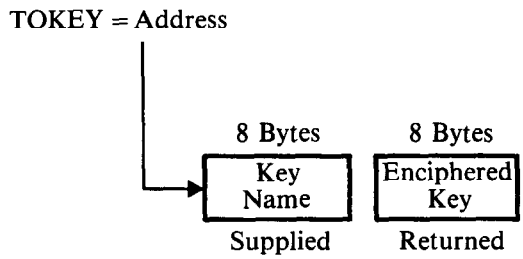
RETKEY

name RETKEY FROMKEY = address, TOKEY = address

The FROMKEY parameter specifies the address of a 16-byte area consisting of an 8-byte key name, and an 8-byte data-encrypting key which has been enciphered under the key-encrypting key specified.



The TOKEY parameter specifies the address of a 16-byte area that contains an 8-byte key name and an 8-byte answer area.



Execution of the RETKEY macro instruction causes the enciphered key supplied in the FROMKEY parameter to be deciphered under the key-encrypting key identified by the supplied key name also in the FROMKEY parameter. The result is then enciphered under the key-encrypting key identified by the supplied key name in the TOKEY parameter.

The qualifiers local and remote are used to distinguish the various cryptographic resources managed by a host processor. For a given host, a *local* resource is one located in the domain of that host; a *remote* resource is one located in or shared with another domain. Each host is a resource itself. There are five types of key-encrypting keys used by the cryptographic system: LOCAL TERMINAL, LOCAL FILE, LOCAL HOST, REMOTE FILE,

[26] Instead of producing K in clear form and then enciphering it under KN1 and KN2, the key manager generates a pseudo-random number, RN, which is defined as $E_{KN1}(K)$. $E_{KN2}(K)$ is then produced by deciphering RN with KN1 and reenciphering the resulting value (K) with KN2.

Local/Remote	Resource	Indicated Key	Key Location
LOCAL	TERMINAL	KMT	CKDS (Host i) & Terminal (Domain i)
	FILE	KNF	CKDS (Host i)
	HOST	KM0i	Cryptographic Facility (Host i)
REMOTE	FILE	KNFij; KNFji	CKDS (Host i & j)
	HOST	KNCij; KNCji	CKDS (Host i & j)

Notes:

LOCAL refers to domain i; REMOTE refers to domain j or something shared with domain j. Lower case letters (i, j) are used to indicate the referenced domain. Letter "i" is sometimes omitted when referring to a LOCAL resource, e.g. KM0 is used in place of KM0i. CKDS denotes the Cryptographic Key Data Set.

Table 4-1. Resources, Keys, and Key Storage Locations

and REMOTE HOST. The relationships among resources, keys, and key storage locations is shown in Table 4-1.

Every key-encrypting key stored at a host is assigned a unique key name. A key name is used by the key manager to locate a key stored in either the cryptographic facility or the CKDS. The key name could, for example, consist of a resource name (either the name of a resource protected by the key, or the location where it is stored) and an identifier:

<key name> = <resource name>, <identifier>

The name of the key installed in the first of a set of terminals might be TERM0001.

The parameters of the GENKEY and RETKEY macros provide a general framework for a user to request key translations. However, it must be realized that only certain combinations of parameter values (as specified by their supplied key names) will be accepted as valid, whereas others will not.[27] The reasons for this are:

1. Certain key translations may be inhibited because they cannot be performed (i.e., the basic cryptographic operations may not permit effective translation).
2. Certain key translations may be inhibited for reasons of security (i.e., the translation, if provided, would lead to an exposure of keys or data).

[27] Error reporting mechanisms are not presented here but can be assumed.

3. Certain key translations may be inhibited simply because the function is not needed by the cryptographic system. The user is only able to do those things that are absolutely required for cryptography.

For example, if the key name of a local terminal and the key name of a local host could be specified for the FROMKEY and TOKEY parameters, respectively, then it would be possible to translate a wiretapped session key (KS) from encipherment under a terminal master key (KMT) to encipherment under a host master key (KM0). Once $E_{KM0}(KS)$ is obtained, anyone with access to the host system could then decipher intercepted ciphertext (enciphered under KS) via the CIPHER macro instruction.

A truth table (matrix of 1s and 0s) could be used by the key manager to determine if certain parameter combinations are valid or invalid. The row and column headings could be identified by LOCAL TERMINAL, LOCAL FILE, LOCAL HOST, REMOTE FILE, and REMOTE HOST (Table 4-2). A

GENKEY

(TOKEY1) \ (TOKEY2)	LOCAL TERMINAL	LOCAL FILE	LOCAL HOST	REMOTE FILE	REMOTE HOST
LOCAL TERMINAL	1	1	1	1	1
LOCAL FILE	1	0	1	1	1
LOCAL HOST	1	1	X	1	1
REMOTE FILE	1	1	1	1	1
REMOTE HOST	1	1	1	1	1

"1" Denotes Requested Translation Allowed
"0" Denotes Requested Translation Denied
"X" Denotes a Null Translation

Note: The cryptographic operations will not permit the encipherment of a common data-encrypting key under two different LOCAL FILE keys.

Table 4-2. Valid and Invalid Parameter Combinations in the GENKEY Macro

similar truth table can be constructed for the parameters of the RETKEY macro (Table 4-3).

In each case, it can be shown that the defined key transformations (Tables 4-2 and 4-3) can be effected by using the RFMK and RTMK operations. For example, a GENKEY request where TOKEY1 designates a local host and TOKEY2 designates a local terminal could be handled as follows. A random number (RN) is produced and defined to be the required data-encrypting key (K) enciphered under the host master key (KM0). RN = $E_{KM0}(K)$ is returned in the 8-byte answer area of the TOKEY1 parameter. An RFMK operation is then used to transform K from encipherment under KM0 to

RETKEY

(FROM-KEY) \ (TOKEY)	LOCAL TERMINAL	LOCAL FILE	LOCAL HOST	REMOTE FILE	REMOTE HOST
LOCAL TERMINAL	0	0	0	0	0
LOCAL FILE	1	0	1	1	1
LOCAL HOST	1	0	X	1	1
REMOTE FILE	1	0	1	1	1
REMOTE HOST	1	0	1	1	1

"1" Denotes Requested Translation allowed
"0" Denotes Requested Translation Denied
"X" Denotes a Null Translation

Notes: ▽ No cryptographic operation(s) will permit a data-encrypting key to be transformed to encipherment under a LOCAL FILE key.

□ No cryptographic operation(s) will permit a data-encrypting key enciphered under a LOCAL TERMINAL key to be transformed to encipherment under any other key.

Table 4-3. Valid and Invalid Parameter Combinations in the RETKEY Macro

encipherment under the master key of the specified terminal (KMT). $E_{KMT}(K)$ is returned in the 8-byte answer area of the TOKEY2 parameter. In the described communications environment, the transformations provided by GENKEY would be effected with the RFMK operation. The logical inverse transformation provided by RETKEY is effected with the RTMK operation. As a general rule, key translations related to a file key involve the use of RTMK for both GENKEY and RETKEY. (See also The Host System Cryptographic Operations, Chapter 5.)

The translation functions of RETKEY are necessary to effect proper key management within the cryptographic system but also provide an opponent with a way to subvert the system's security. For example, interdomain communications require that there be a way for one domain to receive an enciphered session key from another domain, and translate that key (via the RETKEY macro instruction) into a form usable for deciphering data. In effect, this means that an opponent who has intercepted an enciphered session key and data enciphered under that key can perform the same translation by invoking the RETKEY macro, provided that access can be gained to the receiving host system.

It is absolutely essential that the cryptographic system have a way of *controlling* the use of the RETKEY macro. Control can be effected, for example, by requiring that the calling program have privilege equal to that of the host's operating system. In addition, the CKDS can be made a protected data set, thus making it the exclusive resource of the key generator and key manager. Recording denied RETKEY requests on a system log is also advisable as an extra precautionary measure.

Using GENKEY and RETKEY

In a previous example (see An Example of the Encryption of Transmitted Data), user 1 located at terminal 1 in domain i established a communications session, using cryptography in a transparent manner, with user 2 located at terminal 2 in domain j. The network configuration and allocation of secondary keys for this example is shown in Figure 4-58.

User 1 initiates the session by logging on via a terminal. This causes a message to be transmitted to host i requesting that a communications session be established between terminal 1 in domain i and terminal 2 in domain j. As a consequence of this action (Figure 4-59), the teleprocessing access method invokes the key manager via the GENKEY macro instruction, passing as parameters the names of KNC1 and KNC2. The key manager generates a session key (KS) and then causes KS to be enciphered under secondary key KNC1 so that it can be transmitted to terminal 1, and under secondary key KNC2 so that it can be transmitted to host j.

The quantity $E_{KNC1}(KS)$ is transmitted to terminal 1, and the quantity $E_{KNC2}(KS)$ is transmitted to host j. At host j, the teleprocessing access method invokes the key manager via the RETKEY macro and passes as parameters the names of KNC2 and KNC4. The key manager then causes the session key (KS) to be transformed from encipherment under KNC2 to encipherment

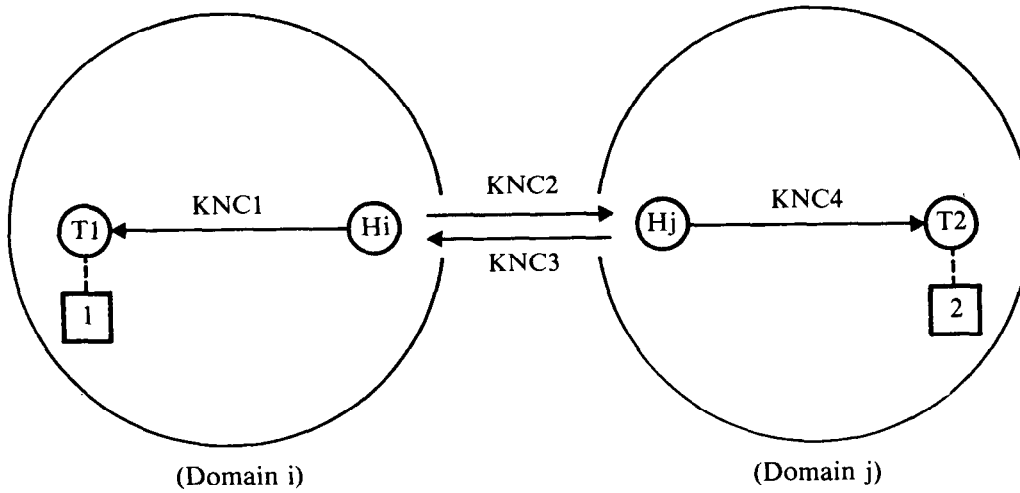


Figure 4-58. Initial Configuration

under KNC4 (Figure 4-60). The quantity $E_{KNC4}(KS)$ is transmitted to terminal 2.

At terminal 1, the session key is recovered by deciphering the quantity $E_{KNC1}(KS)$ with the cipher key KNC1. KNC1 is maintained within the master key storage of terminal 1's cryptographic facility. Similarly, at terminal 2, the session key is recovered by deciphering the quantity $E_{KNC4}(KS)$ with the cipher key KNC4. KNC4 is maintained within the master key storage of terminal 2's cryptographic facility. The cipher keys KNC2 and KNC4 are terminal master keys, although the notation KMT is not used in the present example. As a result of the process of initiating the session, end users 1 and 2 share a common session key (KS) that can be used for ciphering data.

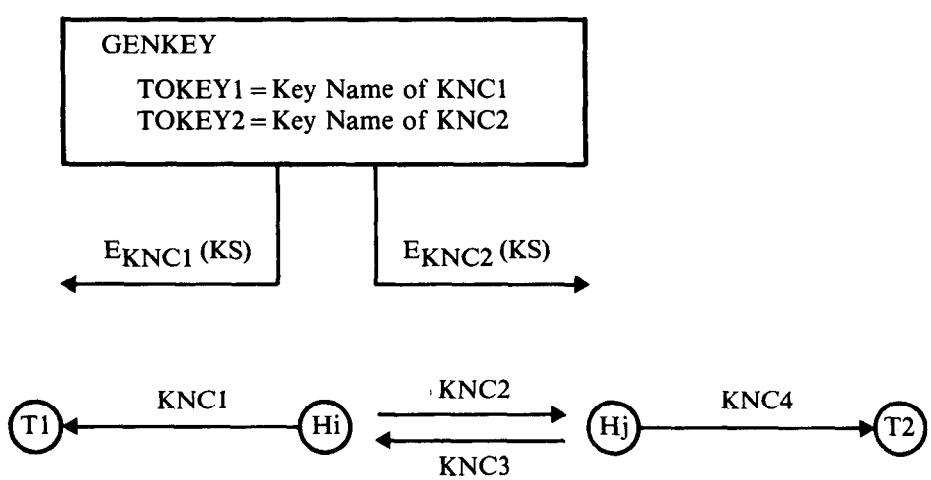


Figure 4-59. Session Key Generation at Host i

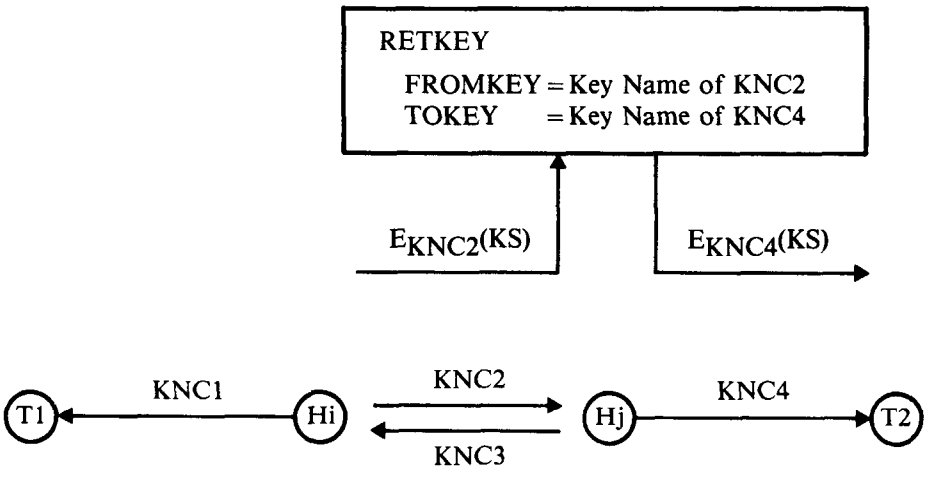


Figure 4-60. Session Key Translation at Host j

THE CRYPTOGRAPHIC KEY DATA SET

The *Cryptographic Key Data Set* (CKDS) is the repository for all secondary keys (other than the host master key) used by the key manager to perform key translation functions. These keys are protected (during their period of storage) by being enciphered under either the first (KM1) or second (KM2) variant of the host master key.

The CKDS is created and maintained by the key generator. A separate input statement is supplied to the key generator for each entry or record in the CKDS that is to be added, updated, or deleted. The input statement has the general format shown in Figure 4-61. The format of each entry (record) in the CKDS is shown in Table 4-4.

The key name is a parameter used to locate an entry in the CKDS. The key type indicates to the key generator whether the stored key is enciphered

Key Type:	Type 1, Type 2, or Type 3 (representing LOCAL TERMINAL, LOCAL FILE, and REMOTE HOST or REMOTE FILE, repectively)
Key Name:	Installation Specified Name
Key1:	Optional 64-Bit Key If a key is required and no key is specified, then a key will be generated automatically.
Key2:	Same as for Key1
Action:	Add, Update, or Delete

Figure 4-61. Input Statement to Add, Update, or Delete a CKDS Record

Field Position (Bytes)	Field Length (Bytes)	Field Description
0	8	Key Name
8	8	Key1 Enciphered Under KM1
16	8	Key2 Enciphered Under KM2
24	1	Resource Type (1, 2, or 3)
25	8	Test Pattern (Optional)

Table 4-4. CKDS Record Format

under KM1 (designated type 1) or KM2 (designated type 2), or whether two different keys are stored under KM1 and KM2 (designated type 3). The resource type indicates to the key manager what cryptographic operations must be performed by the cryptographic facility in order to satisfy requests for translation. The test pattern is an optional parameter that can be used to authenticate stored keys (see Authentication Techniques Using Cryptography, Chapter 8).

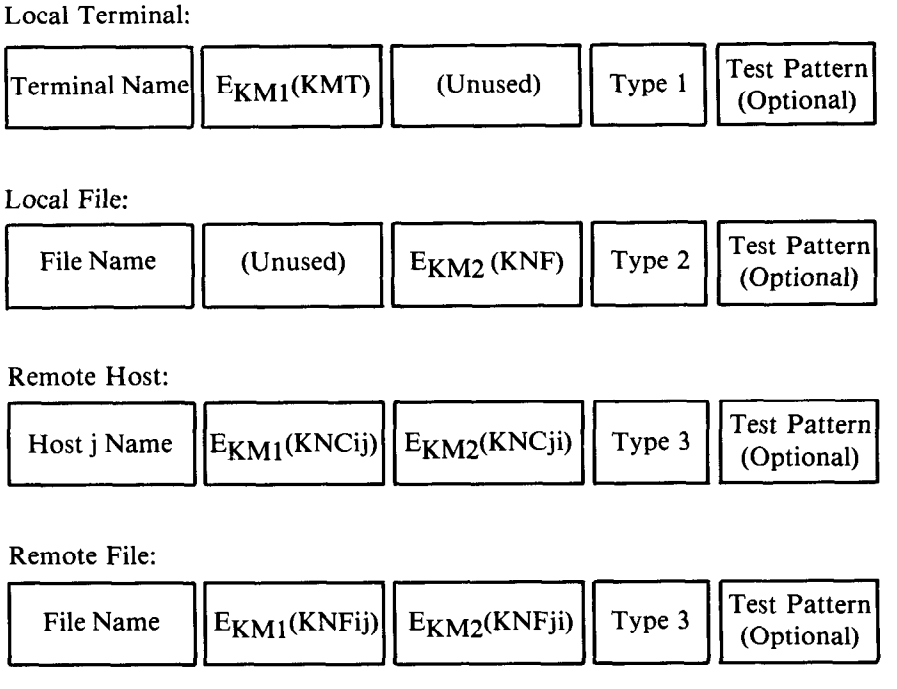


Figure 4-62. CKDS Entries

Secondary keys stored in the CKDS are of the types described in Figure 4-62. A secondary communication key associated with a local terminal is of type 1 (i.e., it is stored enciphered under KM1). A secondary file key associated with a local file is of type 2 (i.e., it is stored enciphered under KM2). A pair of secondary communication keys or secondary file keys associated with a remote host or remote file, respectively, are of type 3, (i.e., one of the keys is stored enciphered under KM1 and the other key is stored enciphered under KM2). A secondary key which allows a primary key to be sent to another host is stored enciphered under KM1, whereas a secondary key which allows a primary key to be received from another host is stored enciphered under KM2.

SUMMARY

The main thrust of this chapter has been to show how the DES algorithm can be implemented in a network of connected computers to provide end-to-end encryption for communication security and file security. (Link encryption was treated briefly for the sake of completeness.) Emphasis has been placed on describing a method for allocating keys (including a hierarchical structure of keys for both key and data protection) and defining a set of basic cryptographic operations that use these keys. The cryptographic operations are performed by a cryptographic facility, or secure implementation, that can be invoked through a programming interface (e.g., with program macro instructions). The intent was to provide the reader with an insight into the design of a cryptographic system and to give enough details of the implementation to allow its operation to be understood. A justification for the particular cryptographic operations and keys (including multiple master keys) defined herein is given in Chapter 5.

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," *IBM Systems Journal*, **17**, No. 2, 106–125 (1978).
2. *IBM Cryptographic Subsystem Concepts and Facilities,* IBM Systems Library Order Number GC22-9063, IBM Corporation, Data Processing Division, White Plains, NY (1977).
3. McFayden, J. H., "Systems Network Architecture: An Overview," *IBM Systems Journal,* **15**, No. 1, 4–23 (1976).
4. Green, P. E., "An Introduction to Network Architectures and Protocols," *IBM Systems Journal,* **18**, No. 2, 202–222 (1979).
5. Baran, P., "On Distributed Communications: IX. Security, Secrecy and Tamper-free Considerations," Memo. RRM-3765-PR, Rand Corporation, Santa Monica, CA (August 1964).
6. Chaum, D. L., "Untraceable Electronic Mail, Return Address, and Digital Pseudonyms," Memo. UCB/ERLM79/9, Electronic Research Laboratory, University of California, Berkeley (February 1979).

7. Cypser, R. J., *Communications Architecture for Distributed Systems,* Addison-Wesley, Reading, MA, 1978.
8. *IBM 3848 Cryptographic Unit Product Description and Operating Procedures,* IBM Systems Library Order Number GA22-7073, IBM Corporation, Data Processing Division, White Plains, NY (1979).
9. Lennon, R. E. and Matyas, S. M., "Unidirectional Cryptographic Functions Using Master Key Variants," *National Telecommunications Conference Record,* **3**, 43.4.1–43.4.5 (1979).
10. *Data Encryption Standard,* Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington, DC (January 1977).

Other Publications that Treat Key Management in Conventional and/or Public-Key Cryptographic Systems

11. Kent, S. T., "Encryption Based Protocols for Interactive User-Computer Communication," *Proceedings Fifth Data Communications Sysmposium,* 5-13 (September 1977). Available from ACM, New York.
12. Everton, J. K., "A Hierarchical Basis for Encryption Key Management in a Computer Communications Network," *Trends and Applications 1978: Distributed Processing,* IEEE Computer Society, Long Beach CA (1978).
13. Merkle, R., "Secure Communications Over Insecure Channels," *Communications of the ACM,* **21**, No. 4, 294–299 (April 1978).
14. Popek, G. J., and Kline, C. S., "Encryption Protocols, Public Key Algorithms, and Digital Signatures in Computer Networks," in *Foundations of Secure Computation,* edited by R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, Academic Press, New York, 1978, pp. 133–153.
15. Needham, R. M., and Schroder, M. D., "Using Encryption for Authentication in Large Networks of Computers," *Communications of the ACM,* **21**, No. 12, 993–999 (December 1978).
16. Kohnfelder, L. M., "Towards a Practical Public-Key Cryptosystem," B.S. Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge (May 1978).
17. Smid, M. E., *A Key Notarization System for Computer Networks,* NBS Special Publication 500–54, U.S. Department of Commerce, National Bureau of Standards, Washington, DC (October 1979).
18. Konheim, A. G., Mack, M. H., McNeill, R. K., Tuckerman, B., and Waldbaum, G., "The IPS Cryptographic Programs," *IBM Systems Journal,* **19**, No. 2, 253–283 (1980).
19. Blakley, G. R., "Safeguarding Cryptographic Keys," *AFIPS Conference Proceedings,* **48,** 313-317 (June 1979).
20. Shamir, A., "How to Share a Secret," *Communications of the ACM,* **22,** No. 11, 612-613 (1979).