A key controlled cryptographic system requires a mechanism for the safe and secure generation, distribution, and installation of its cryptographic keys. This paper discusses possible key generation, distribution, and installation procedures for the key management scheme presented in the preceding paper.

Generation, distribution, and installation of cryptographic keys

by S. M. Matyas and C. H. Meyer

Key generation is the process that provides the cryptographic keys required by a cryptosystem. *Key distribution* is the transporting or routing of cryptographic keys through the cryptosystem for subsequent installation. *Key installation* is the entering of cryptographic keys into designated cryptographic devices. The protocols presented here for the generation, distribution, and installation of cryptographic keys are based on the key management scheme presented by Ehrsam et al. in the preceding paper.[1]

The key management scheme discussed by Ehrsam et al. distinguishes between key-encrypting keys and data-encrypting keys. The former are used to encipher other keys and are defined ahead of time as part of the process of initializing the cryptographic system, or cryptosystem. They remain constant for relatively long periods—they may be changed perhaps once a year. Data-encrypting keys, on the other hand, are generated dynamically during regular system operations, hence special precautions must be taken to protect them. Data-encrypting keys remain in existence as long as the data exists that they protect. That period, for communication security, is determined by the length of time the user is signed on to the system. Usually it is relatively short. For file security, when data is stored in enciphered form, data-encrypting keys may exist for a relatively long time.

One special key-encrypting key, the *host master key*, is generated by some random process such as tossing a coin or throwing dice. All other key-encrypting keys are produced by using the Data Encryption Standard (DES)[2] as a pseudorandom-number generator, a procedure that can be performed under secure conditions on the computer. Data-encrypting keys are generated dynamically, as needed, at the host processor by exploiting the randomness associated with the many different users and processes that normally are active on the system at any given time.

With the DES, each 64-bit cryptographic key consists of 56 independent key bits and eight bits that can be used for parity checking. There are 2^{56} different possible keys. Since the DES is a publicly known algorithm, cryptographic strength must be based on the secrecy of its cryptographic keys. For that reason, great care must be exercised in selecting keys. They should be as nearly random as possible, so that an organized search for them would not be likely to meet with early success. If there were a known bias in the selection of keys, for example, an opponent could search the more likely candidates first.

The distribution of cryptographic keys can be accomplished in a variety of ways, as by courier, registered mail, or telephone, as long as the likelihood of compromising the key is within acceptable limits. Key installation can be accomplished by activating hardware switches or dials or by reading the key into main memory and then exercising an appropriate operation to set the key into the cryptographic facility. A number of techniques can be used to minimize the risk of entering an incorrect key into the system. They include multiple entry, validation patterns, and exercising the cryptographic system. In the present discussion, it is assumed that host and terminal master keys are entered in the form of 16 parity-adjusted hexadecimal digits.

Generation of the host master key

Regardless of the selection procedure used for the specification of keys, organized and predictable methods should be avoided for choosing key bits. Any selection procedure based on one's telephone number, name and address, date of birth, or the like, is so frail that no real protection is provided. Also, the pseudorandom-number generator programs available on many computer systems are far too predictable to be used for this purpose and should be avoided.

Since the master key, either directly or through one of its derived variants, provides protection (through encipherment) for all other keys stored in the system, and since the master key will in all probability remain unchanged within the system for long

periods, great care must be taken to select this key in a random manner. The method recommended here is to use a random process performed by the user of the system.

Assume that a 64-bit parity-adjusted key is required for the selection process, and that odd parity is used; that is, every eighth bit is adjusted so that the number of bits in the eight-bit group is odd. With a parity-adjusted key in the system, a consistency check can be made to ensure that the proper key has been entered.

tossing coins

Let the bit values 0 and 1 in the cryptographic key be determined by the occurrence of heads and tails, respectively. Then toss 56 coins in eight groups of seven coins each, and record the results. Each group is then converted to its corresponding parity-adjusted hexadecimal digits as shown in Table 1.

Each group of seven bits in the 56-bit key is expanded to eight bits by appending an additional parity bit (odd parity is maintained). This process can be performed with the aid of a table, if desired. Using Table 2, for example, the first four bits index the table row and the last three bits the table column. Since every entry in the table has correct parity, a parity-adjusted key will be formed even if there should be an error in indexing. The cryptographic key (the value entered into the system and saved in a secure repository for backup purposes) is defined as that string of hexadecimal digits produced by the table lookup process. The values used in indexing the table are ignored (discarded).

throwing dice

The method described above can also be used with dice. Instead of tossing seven coins, the user rolls seven dice. The binary digits can be obtained by considering an even roll (2, 4, or 6) to represent a 0 bit and an odd roll (1, 3, or 5) to represent a 1 bit.

Generation of key-encrypting keys

The potentially large number of key-encrypting keys that might be used in a cryptosystem increases the likelihood that at least one of them may become known to an opponent. Therefore the key generating procedure must, as an absolute minimum, be designed so that if one or more keys become compromised, the work factor will remain high enough to provide sufficient protection for the remaining keys.

It is recommended that the key generating procedure involve or make use of the host master key, or one of its variants, by executing one or more of the cryptographic key management operations. Not only will an opponent be forced to carry out part of his attack on the same host system, but because the operations themselves must be executed as part of the attack, the opponent is

Table 1 Results of coin tossing converted to binary and hexadecimal digits (heads H = binary 0, tails T = 1)

Trial	*Result*	*Binary*	*Hex*
1	HHHTHTH	0001 010	15
2	THTHHHT	1010 001	A2
3	TTHTHHH	1101 000	D0
4	THHHTTT	1000 111	8F
5	HHTTTTH	0011 110	3D
6	TTHHHHH	1100 000	C1
7	HHHTTHT	0001 101	1A
8	HHTHTHH	0010 100	29

Parity adjusted key = hex 15A2D08F3DC11A29

Table 2 Parity adjusted hexadecimal digits (odd parity)

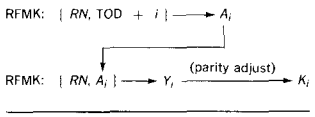
DECIMAL		0	1	2	3	4	5	6	7
	BINARY	000	001	010	011	100	101	110	111
0	0000	01	02	04	07	08	0B	0D	0E
1	0001	10	13	15	16	19	1A	1C	1F
2	0010	20	23	25	26	29	2A	2C	2F
3	0011	31	32	34	37	38	3B	3D	3E
4	0100	40	43	45	46	49	4A	4C	4F
5	0101	51	52	54	57	58	5B	5D	5E
6	0110	61	62	64	67	68	6B	6D	6E
7	0111	70	73	75	76	79	7A	7C	7F
8	1000	80	83	85	86	89	8A	8C	8F
9	1001	91	92	94	97	98	9B	9D	9E
10	1010	A1	A2	A4	A7	A8	AB	AD	AE
11	1011	B0	B3	B5	B6	B9	BA	BC	BF
12	1100	C1	C2	C4	C7	C8	CB	CD	CE
13	1101	D0	D3	D5	D6	D9	DA	DC	DF
14	1110	E0	E3	E5	E6	E9	EA	EC	EF
15	1111	F1	F2	F4	F7	F8	FB	FD	FE

constrained by the particular operational characteristics of the host machine itself. Since the time it takes to encipher and decipher is known for a given system, the minimum required computation time can be determined.

the key generating procedure

The present implementation[3, 4] uses a method employing the DES as a pseudorandom-number generator. The basic idea is that a 64-bit random number *RN* can be used in conjunction with the DES algorithm to produce the entire set of key-encrypting keys (except the host master key). *RN*, in this case, is generated externally by a random process like that used in generating the host master key, such as coin tossing or dice throwing. Let Y_i equal the ith pseudo-random number generated, and let K_i equal the ith cryptographic key ($i = 1, 2 \cdots n$), which is obtained from Y_i by adjusting each byte for odd parity.

Figure 1 Using the *reencipher from master key* (RFMK) operation to make Y_i a function of both the host master key *KM0* and a 64-bit pseudorandom number *RN*



Note that:
TOD = time of day
i = the number of pseudorandom numbers generated
A = a 64-bit intermediate result
Y_i = $E_\alpha (D_{KM0} (E_a (D_{KM0} (i))))$
α = $D_{KM1} (RN)$
KM0 = host master key
KM1 = first variant of KM0
E = *encipherment of*
D = *decipherment of*

The approach described here is to use one of the host processor's cryptographic operations so that the Y_i values are functions of the host master key as well as functions of *RN*. This approach makes use of the *reencipher from master key* (RFMK) operation[1] as shown in Figure 1. Since *RN* is a random number, so is the intermediate quantity $D_{KM1}(RN)$ in Figure 1 (provided, as assumed here, that the DES does not introduce any bias). Therefore Y_i is a function of two secret, independent cryptographic keys: one (*RN*) supplied by the user, the other (*KM0*) supplied by the system.

Because the DES algorithm is irreversible, knowledge of several clear keys $K_{i_1}, K_{i_2} \cdots K_{i_j}$ or, in fact, even knowledge of the corresponding values $Y_{i_1}, Y_{i_2} \cdots Y_{i_j}$ would not permit *RN* or *KM0* to be deduced. Therefore, knowledge of one or more of the generated keys would not allow any of the remaining keys to be deduced.

It should be pointed out that the procedure described above does not depend on the randomness provided by the time-of-day clock. A clock value is introduced, in this case, to reduce the likelihood that the user will inadvertently generate a duplicate list of keys by accidentally entering the same *RN* value into the procedure. A duplicate *RN* could be entered, for example, if a new *RN* were punched into a data card which was accidentally replaced with a card containing an old value.

Generation of data-encrypting keys

A data-encrypting key is produced by generating a 64-bit number *RN*, and defining it to be the desired key already enciphered under a key-encrypting key known to the system. For example, in communication security, *RN* is defined as the session key, *KS*, enciphered under the host master key, as follows:

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

In file security, on the other hand, *RN* is defined as the file key, *KF*, enciphered under a secondary file key *KNF*, as follows:

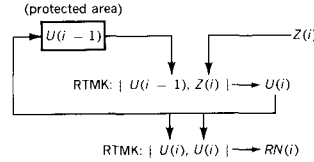
$$RN \equiv E_{KNF}(KF)$$

With this strategy, it is not necessary to generate the data-encrypting key in clear form and then encipher it under the appropriate key-encrypting key—that is, the data-encrypting key is never exposed in clear form.

Basically, a pseudorandom number *RN* is generated within the host processor as a result of the dynamically changing and unpredictable nature of the resource demands placed upon the system by its users.

One approach to generating pseudorandom numbers makes use of the *reencipher to master key* (RTMK) operation in conjunction with two seed values, U and Z, which are derived internally within the host processor. The seed values are used as input parameters to the RTMK operation, which, in turn, is used in deriving RN. Two independent seed values are used to give the procedure added strength, since both values must be compromised before a successful attack is possible. Figure 2 illustrates the basic idea behind this pseudorandom-number generating procedure.

Figure 2 DES-based pseudorandom-number generator for data-encrypting keys



$U(0)$ = arbitrary value
$Z(i)$ = function of two or more time-of-day clock readings
$RN(i)$ = ith generated random number

The specific relations are:

$$U(i) = E_{KM0}(D_{D_{KM2}(U(i-1))}(Z(i)))$$
$$RN(i) = E_{KM0}(D_{D_{KM2}(U(i))}(U(i)))$$

Consecutive seed values—$Z(1)$, $Z(2) \cdots Z(i)$, for example—could be generated by combining two or more independent time-of-day clock readings. Independence, in this case, can be achieved by interleaving an input-output operation of unpredictable length between successive clock readings. The seed value $U(0)$ could be derived from a combination of user-dependent and process-dependent information stored in the volatile memory of the host processor. Each value $U(i)$, for values of i greater than zero, is defined as the output of an RTMK operation whose input consists of $U(i-1)$ and $Z(i)$. Hence it follows that $U(i)$ is a function of $U(0)$ and $Z(1)$, $Z(2) \cdots Z(i)$.

It should be noted that $RN(i)$ is a function of two independent seed values, $U(i-1)$ and $Z(i)$, each with enough combinations to thwart exhaustive analysis. The U-values, $U(1)$, $U(2) \cdots U(i)$ are generated internally by feeding back the result from the first RTMK operation. These values are protected by using a second RTMK operation. The RTMK operation ensures that it is not possible to deduce $U(i)$ from $RN(i)$. Hence knowledge of one or more of the generated values of RN will not permit other values of RN, past or future, to be deduced.

To subvert this pseudorandom-number generating process, an opponent must cope with the changing and unpredictable values of $Z(i)$ and the secret quantity $U(i-1)$, which itself is a function of $U(0)$ and $Z(1)$, $Z(2) \cdots Z(i-1)$. Even if one of the seed values should become compromised, the other provides enough cryptographic strength so that an exhaustive attack intended to recover a set of eligible $RN(i)$ would be computationally infeasible.

Master key entry at the host processor

For reasons of security, the master key cannot be read once it has been set into a cryptographic facility. The following procedure, however, will allow a system administrator to determine, within certain limits, whether the master key stored in the cryptographic facility is the one that was intended.

Some function, $\phi(KM0)$, of the master key can be computed externally to the system and compared with a similar quantity generated within the system. For example, with the aid of a programmed version of the DES, $KM0$ could be used as a key to encrypt a 64-bit random number RN:

$$\phi(KM0) = E_{KM0}(RN).$$

Once the master key has been set into the cryptographic facility, the *encipher under master key* (EMK) operation could be used to generate the same quantity, as follows:

$$\text{EMK: } \{RN\} \rightarrow E_{KM0}(RN).$$

Comparison of these two values can establish whether the keys used in the two routines are identical. The comparison is only relative, however, because the wrong, but identical, key could have been entered in both routines. The challenge, therefore, is to reduce the risk of this event as far as possible.

hard-wired entry

The reading of temporarily stored keys into the main storage of a system can be avoided by providing a direct wire connection between the key entry point and the nonvolatile key storage area of the cryptographic facility. The key can then be entered by means of toggle switches, dials, or the like. The direct wire connection should be so constructed, as by shielding, that probing or the tapping of transmitted information (the keys) is not possible.

Even with hard-wired key entry, there is still some chance that the key entered into the cryptographic facility will be different from the key that was intended. The following analysis provides an estimate of the probability of an undetected error, p(UE), in the entered key—that is, the probability that a wrong key has been installed in the nonvolatile storage of the cryptographic facility.

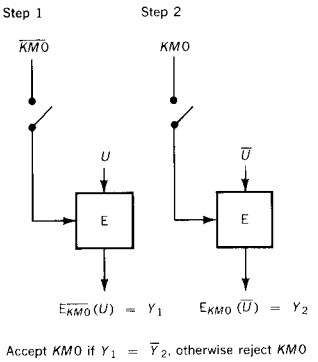
To simplify the calculation, let it be assumed that only one of the 16 hexadecimal digits entered into the cryptographic facility might be in error (that is, that multiple errors, whose probability of occurrence is small anyhow, will be ignored), and that the probability of error, p, during key entry is much higher than the probability of error between the key entry point and the nonvolatile storage area of the cryptographic facility. These assumptions are reasonable in view of the fact that machine error is much less likely than is human error.

In the situation described above, an error in key entry will not be detected if the parity of the incorrectly entered hexadecimal digit is correct. Since there are 16 possible hexadecimal digits, of which only eight have odd parity, eliminating the correct digit leaves seven combinations that have correct parity out of 15 possible combinations. Therefore the probability of an undetected error, p(UE), is given by

$$
\begin{aligned}
p(UE) &= p(KM0 \text{ entry in error, correct parity}) \\
&= p(\text{parity correct} \mid KM0 \text{ entry in error}) \cdot p(KM0 \text{ entry in error}) \\
&= (7/15)p \\
&= 0.4667p
\end{aligned}
$$

To improve the situation, two quantities ($KM0$ and a function ϕ of $KM0$) are specified in such a way that errors associated with the entry of $KM0$ and $\phi(KM0)$ are statistically independent. The choice of $\phi(KM0) = KM0$ would not be a candidate, since it would amount to entering $KM0$ twice, and an error in the first entry might well be repeated in the second. The choice of $\phi(KM0) = \overline{KM0}$ (the complement of $KM0$) appears to be satisfactory. In this case, the complementary property of the DES algorithm[1] can be used to advantage by first installing the complement of $KM0$ in the cryptographic facility and enciphering the arbitrary value U, then installing $KM0$ in the facility and enciphering the complement of U. The output values are defined as Y_1 and Y_2, respectively. By the complementary property of the DES, $KM0$ can be assumed to be installed properly in the cryptographic facility whenever Y_1 equals the complement of Y_2. Figure 3 illustrates this entry procedure.

Figure 3 Master-key entry procedure at the host processor



An undetected error can now occur only if the entered quantities are complements of each other and the parity of each quantity is correct. It can be shown that

$$
\begin{aligned}
p(UE) &= (7/15)^2(1/16)(1/7)p^2 \\
&= 0.00194p^2
\end{aligned}
$$

Any corruption of the key in transit between the entry point and the cryptographic facility also will be detected by this checking procedure. There is very little chance that such errors will occur.

indirect entry

To enter the master key indirectly, it is read into main memory and a *set master key* (SMK) operation is exercised to load it into the nonvolatile storage area of the cryptographic facility. The process of entering the master key into the system through an external interface, however, introduces a potential source of human error. Moreover, a hardware or software error could occur at any time while the key is in transit from its point of entry to the cryptographic facility. Any of these potential sources of error could cause an incorrect key to be set.

It is recommended that the master key be entered from a nonvolatile medium such as punched cards or magnetic tape. Such a medium could be stored in a secure location, as in a safe or vault. Thus it would be possible, as part of the standard protocol, to define the master key as that value which is recorded on the medium—provided, of course, that it has correct parity. Any human error committed in recording the key on the medium, then, would be of no real consequence. Likewise, any corrup-

tion of the master key between its entry point and the cryptographic facility can be detected by using the procedure described previously, in which both the host master key and its complement are entered into the cryptographic facility.

Master key entry at a terminal

Again, because the terminal master key (*KMT*) cannot be read once it has been set, a checking procedure should be used to ensure that it has been installed properly in the cryptographic facility. The key can be set by means of switches, dials, or the like, or it can be entered at a keyboard.

on-line checking

One way of determining whether the proper master key has been set into a terminal's cryptographic facility is to establish a communications session with the host processor. If the installed terminal master key differs from that stored at the host, the session key at each node will be different and it will not be possible to send and recover an agreed upon message.

For example, a simple hand shaking protocol could be adopted as part of the session initiation process. The terminal could transmit a value N, under KS encipherment, to the host processor, where a function $\phi(N)$ would be computed and sent back to the terminal under KS encipherment. At the terminal, a check would be made to ensure that the returned value of $\phi(N)$ agreed with a similar value computed at the terminal. If the values agreed, the terminal master key would be accepted.

off-line checking

It is often desirable to check *KMT* directly at a terminal without involving the host processor at the same time. To do so, a validation pattern V can be used. This pattern is a nonsecret function of *KMT*, created as part of the key generation process. By using the *encipher data* (ECPH) operation,[1] the validation pattern V_T is derived at a terminal as follows:

ECPH: $\{TID, TID\} \rightarrow E_{D_{KMT(TID)}}(TID) = V_T$

where *TID* is a terminal identification number unique to each terminal. At key generation time, the *encipher under master key* (EMK) and *decipher data* (DCPH) operations are used, along with the *encipher data* (ECPH) operation,[1] to produce a similar validation pattern, V_H, at the host system, as follows:

EMK: $\{KMT\} \rightarrow E_{KM0}(KMT)$

DCPH: $\{E_{KM0}(KMT), TID\} \rightarrow D_{KMT}(TID)$

EMK: $\{D_{KMT}(TID)\} \rightarrow E_{KM0}(D_{KMT}(TID))$

ECPH: $\{E_{KM0}(D_{KMT}(TID)), TID\} \rightarrow E_{D_{KMT(TID)}}(TID) = V_H$.

The person who is authorized to enter *KMT* at the terminal is given the quantity V_H. As part of the entry procedure, the terminal will cause the value V_T to be generated and displayed. The user can determine whether *KMT* has been entered correctly by comparing V_H with V_T.

If V_H is stored in nonvolatile storage at the terminal, frequent checks can be made on the correctness of the master key. Alternately, V_H could be written down and posted in a conspicuous location at the terminal. A keyboard entry command causing V_T to be generated and displayed thus would allow V_T and V_H to be compared periodically by the terminal operator.

Distribution of cryptographic keys

Whenever data-encrypting keys (session keys and file keys) occur outside the cryptographic facility, they are maintained under the encipherment of some key-encrypting key. This protocol allows data-encrypting keys to be routed through the system over paths that are not secure. Recovery of the data-encrypting key in a usable form is possible only if the recipient possesses the appropriate key-encrypting key.

Key-encrypting keys are distributed through the system in an altogether different way. One cannot always rely on encryption as a means of protecting the secrecy of these keys, since each node must have at least one key that is installed initially in clear form. That key must be sent to the node over a path whose risk of compromise is within acceptable limits; that is, the probability of interception must be very low. One such method is to use a courier, normally the safest and most secure means of transporting keys. Security in this case, of course, depends on the reliability of the courier.

Although not necessarily recommended, other means of transmitting keys are by registered mail and by private telephone conversation. These methods are less secure than using a courier because there is a greater chance that an opponent could intercept the key during transmission. The probability of compromise could be reduced, however, by transmitting two or more bit patterns over independent paths and combining them at the final destination, for example by using an EXCLUSIVE-OR operation.

The same method could be applied to the key entry procedure itself. For example, several different bit patterns could be entered into the cryptographic facility by each of several persons. These bits could then be combined within the cryptographic facility to produce the desired key. For the key to be compromised, this protocol would require the collusion of all persons involved in the key entry process.

Summary

In connection with the key management scheme discussed by Ehrsam et al. in the preceding paper,[1] we have described two kinds of keys:

- Data-encrypting keys, which protect either data in transit (primary communication keys, or session keys) or stored data (primary file keys, or file keys);
- Key-encrypting keys, which encipher other keys—for example, host master keys, secondary communication keys (of which the terminal master key is a special case), and secondary file keys.

Generally speaking, the method of key generation that is best for a given class of cryptographic keys is determined by the expected number of each type of key that will be needed and the time when the keys will be required by the cryptosystem. In many cases, the keys can be created dynamically on demand, but sometimes they are required ahead of time to initialize the system.

The host master key is generated by a random process such as tossing coins or throwing dice. Human involvement to that extent is reasonable in the key generation process because only one master key is required for each host processor and the master key is likely to remain unchanged for a relatively long time. Since the master key protects all other keys stored at the host processor, special care should be taken to ensure that it is generated and installed in the cryptographic facility in a secure manner.

It is reasonable to anticipate that the total number of key-encrypting keys (excluding the host master key) may be large enough so that they should be generated using mechanical (nonhuman) procedures. The desired keys can be produced using the DES algorithm as a pseudorandom-number generator. The seed values used in this procedure are generated by the user employing a random process similar to that used in generating the host master key. Since the key-encrypting keys are used in initializing the cryptosystem, they must be generated ahead of time. This can be accomplished under secure conditions on the computer.

Data-encrypting keys also are required in large numbers (one for each session and file using encryption), but they need not be generated until specifically requested—that is, until they are needed to protect a communications session or to protect stored data. Hence data-encrypting keys either could be generated ahead of time and stored in table form until needed, or they could be generated dynamically on demand. Disadvantages in generating them ahead of time are that the keys would be exposed longer to possible compromise by an opponent, and they would require addi-

tional storage. One approach for dynamically generating data-encrypting keys is to make use of the randomness associated with the many users and processes normally active on the system at any one time.

Among the more important principles to be followed in key generation is that the compromise of one or more keys should not make it possible for the remaining keys to be deduced easily. With regard to key distribution, we have shown that security can be increased whenever two or more bit patterns of 64 bits are transmitted over different paths and combined at the final destination. To enhance the security of key installation, we suggested that two different related values, the key and a function of the key, be entered into the cryptographic facility. Any errors that occur in both values will be statistically independent, so the likelihood of an undetected error—that is, the probability that a wrong key will be installed—will be greatly reduced.

CITED REFERENCES

1. W. F. Ehrsam, S. M. Matyas, C. H. Meyer, and W. L. Tuchman. "A cryptographic key management scheme for implementing the Data Encryption Standard," *IBM Systems Journal* **17**, No. 2, 106–125 (1978, this issue).
2. *Data Encryption Standard*, Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington, DC (January 1977).
3. *IBM Cryptographic Subsystem Concepts and Facilities*, IBM Systems Library order number GC22-9063, IBM Corporation, Department 63T, Neighborhood Road, Kingston, New York 12401 (1977).
4. *Programmed Cryptographic Facility Program Product General Information Manual*, IBM Systems Library order number GC28-0942, IBM Corporation, Department D58, South Road, Poughkeepsie, New York 12603 (1977).

Reprint Order No. G321-5067