CHAPTER THREE

The Data Encryption Standard

Generally, all ciphers are substitution ciphers since in a cipher a set of plaintext messages is always uniquely transformed into a set of ciphertext messages for a given key, and in effect this constitutes substitution of ciphertext for plaintext. However, three classes of ciphers are ordinarily distinguished: *transposition ciphers, substitution ciphers,* and a combination of both called *product ciphers.*

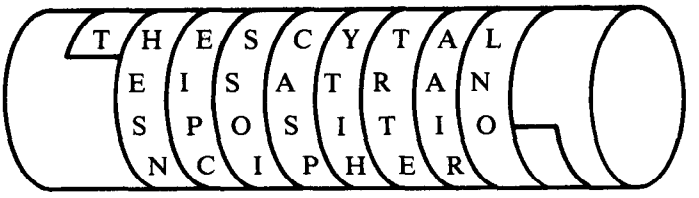
The chapter begins with a short discussion of these three classes of ciphers. Next, ciphers using a linear feedback approach are shown to be breakable by expressing the relationship between keys, ciphertext, and plaintext as a set of mathematical equations, and then using analytical techniques to solve them. Although the attacks are elementary in nature, they do provide good background information to discuss broad principles followed in cipher design.

A detailed discussion of the Data Encryption Standard (DES) is presented in the second half of the chapter, including the generation procedures for DES's internal keys. Also, there is a simple numerical example that allows the DES's operation to be followed using only paper and pencil. Finally, there is a demonstration, both mathematically and pictorially, of how complexity in the ciphering process builds up with the number of iterations—a process involving substitution and permutation.

CLASSES OF CIPHERS

Transposition ciphers, consisting of rearrangements of plaintext letters, were used by the Greeks as early as 400 B.C. They represent the oldest known method of encryption. To encipher messages, the ancient Greeks used a device called a scytale (Figure 3-1). It consisted of a long narrow strip of papyrus wrapped around a cylinder (the diameter determined the key), with the plaintext written horizontally (row-by-row). When the strip was unwound, the letters were rearranged to conceal the message. To read the message, the recipient merely rewrapped the strip around a cylinder whose diameter was the same as the original.

A substitution cipher replaces plaintext letters, without changing their sequence, with one or more letters, figures, or symbols. An example of an early substitution cipher is the Julius Caesar cipher illustrated in Figure 3-2.



Plaintext: The Scytale is a transposition cipher.

Figure 3-1. The Scytale

For encipherment, each plaintext letter was replaced by the letter obtained when the alphabet was shifted three positions to the left (i.e., A would be replaced by D, B by E, and so forth). In decipherment, the process was simply reversed (i.e., D would be replaced by A, E by B, and so forth).

A product cipher involves the steps of both substitution and transposition. The ADFGVX product cipher illustrated in Figure 3-3 was used by the German Army during World War I. It employed a table having six rows and six columns labeled with the letters A, D, F, G, V, X. The table contained 26 letters (A, B, . . . , Z) and 10 digits (0, 1, . . . , 9) inserted in a random order. Encipherment was accomplished by replacing each plaintext letter with the pair of letters that described its position (row and column) within the ADFGVX table. This intermediate text was then written (row-by-row) into a transposition rectangle, and the columns read according to a numerical key. For example, the sorted alphabetical order for the key DEUTSCH became CDEHSTU, since letter C is ranked lowest or first in alphabetical sequence, letter D is ranked next, and so forth. Thus, to obtain the ciphertext, the column under C is read first, the column under D is read second, and so forth.

Decipherment is accomplished by writing the ciphertext back into the transposition rectangle (column-by-column) according to the same numerical key, reading the transposition rectangle (row-by-row) to obtain the intermediate text, and then finding each plaintext letter by using each pair of letters in the intermediate text as coordinates in the ADFGVX table.

Key:

Plaintext Letter:	ABCDEFGHIKLMNOPQRSTVXYZ
Ciphertext Letter:	DEFGHIKLMNOPQRSTVXYZABC

(In Latin the letters J, U, and W were not used.)

Message: E P L V R I B V S V N V M

Cryptogram: H S O Z V M E Z X Z Q Z P

Figure 3-2. Substitution Cipher Used by Julius Caesar

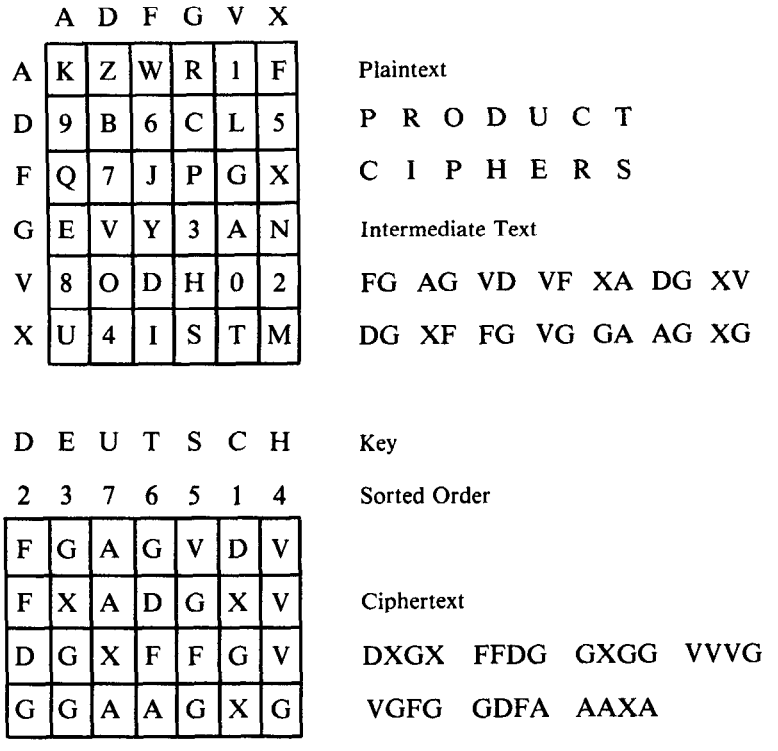


Figure 3-3. The ADFGVX Cipher—an Example of a Product Cipher

Cryptographic research conducted during World War II showed that strong encryption algorithms could be obtained using alternate steps of substitution and transposition, resulting in a product cipher. In his classic paper on secrecy systems, Shannon [1] pointed out that mixing functions—functions obtained as the product of two simple noncommutative operations—could be used to achieve cryptographic strength. Although based on the principle of mixing functions, the ADFGVX cipher is actually a weak algorithm, since substitution and transposition are used only once and the substitution is not under control of a key.

By the late 1960s, threats to computer data began to be viewed as real problems. The need for a strong method of encryption in the private sector was at last apparent. At the same time, large scale integration (LSI) technology permitted a highly complex cryptographic algorithm to be implemented on a single chip, thus achieving the high-speed encryption essential to data processing.

Research into the development of strong product ciphers was undertaken by private industry between 1968 and 1975. A block product cipher designed by Feistel [2] was implemented in a cryptographic system known as LUCIFER [3]. A new cryptographic algorithm based on the LUCIFER design was developed shortly thereafter at IBM under the leadership of Dr. W. L. Tuchman [4]. The new algorithm consisted of 16 alternate steps (or rounds) of key-

controlled substitution and fixed permutation. This algorithm, approved by the NBS, was embodied in a federal standard that became effective on July 15, 1977.

Known as the Data Encryption Standard (DES) [5] the algorithm enciphers a 64-bit block of plaintext into a 64-bit block of ciphertext under the control of a 56-bit cryptographic key. The process of encryption consists of 16 separate rounds of encipherment, each round using a product cipher approach, or *cipher function*. The interaction of data, cryptographic key K, and cipher function g is illustrated in Figure 3-4. The externally supplied key K consists of 64-bits: 56 bits are used by the algorithm and eight bits may be used for parity checking. A different subset of 48 key bits from the 56-bit key is used in each round. The subsets of key bits used for encipherment are denoted K(1), K(2), . . . , K(16). During decipherment, the keys are used in reverse order (K(16) in round one, K(15) in round two, and so forth). The initial and inverse initial permutations allow the algorithm to be implemented more easily on a single chip, provided that the data and key are serially loaded.

DES can be thought of as a huge key-controlled substitution box (S-box) with a 64-bit input and output. With such an S-box, a total of $(2^{64})!$ different transformations or functions from plaintext to ciphertext are possible. The 56-bit key used with DES thus selects only a small subset (2^{56}) of the total set's possible functions.

A single huge S-box is impossible to construct. Therefore, DES is implemented by using several smaller S-boxes (6-bit input and 4-bit output) and permuting their concatenated outputs. By repeating the substitution and permutation process several times, cryptographic strength increases.

When referring to the cryptographic transformations of encipherment and decipherment, E denotes encipherment and D denotes decipherment. The notation used to express these operations is

$$E_K(X) = Y$$

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

$$D_K(Y) = X$$

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

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

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

where the bars represent complementation, or bit inversion. (This property can be used advantageously for testing purposes, as demonstrated in Chapter 6.)

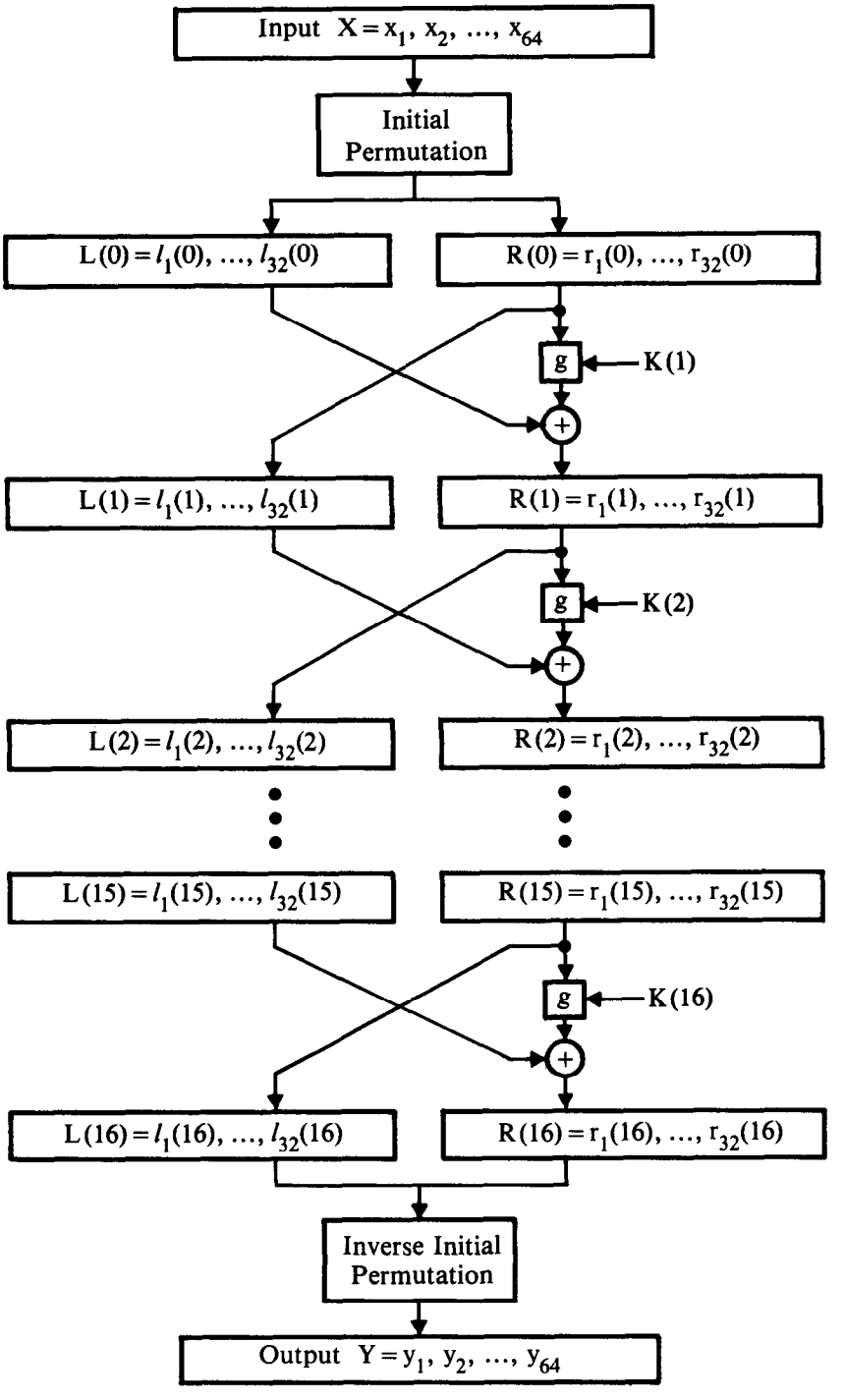


Figure 3-4. Enciphering Computation

Because of the complementary property of DES, if an analyst could obtain $E_K(X)$ and $E_K(\overline{X})$ for an arbitrary X, he could reduce the size of the key space he must search from 2^{56} to 2^{55}. Therefore, the key space could be exhausted in 2^{55} trials instead of 2^{56} trials. However, depending on the implementation, it may not be possible for an opponent to obtain plaintext X and its complement $\overline{X}$ enciphered under the unknown cipher key. Moreover, it has been suggested that in special cases, when more security is desired, multiple encryption methods can be used as a means of increasing the work factor associated with breaking the system (see also Appendix D). At this writing, the authors are unaware of any demonstrated method of using DES-ciphered data to solve for a single key bit, other than by key exhaustion (trying each possible key).

DESIGN CRITERIA

Generally, the steps of substitution and permutation in the DES algorithm have the following relationship to the algorithm's strength. Deterministic attacks (purely mathematical and nonstatistical in nature) are deterred mainly by the use of nonlinear functions in the substitution process. Statistical attacks are deterred mainly by the permutation of bits after each step of substitution (smoothing out the statistics). In the strict sense, both deterministic and statistical attacks are deterred by a combination of substitution and permutation.

Why nonlinear substitution functions are essential to strong product ciphers is illustrated by a cryptanalysis of three different encryption algorithms that make use of linear functions. Such an analysis is useful in arriving at design criteria for strong cryptographic algorithms. For example, a common pitfall in the design of a stream cipher is the assumption that a strong algorithm can be achieved merely by ensuring that the bit stream, which is added to the plaintext using modulo 2 addition, is a long sequence of pseudo-random bits (perhaps on the order of billions of bits). Even so, if there is a linear relationship among the bits in the pseudo-random bit stream, then the algorithm can easily be broken.

Breaking a System with Two Key-Tapes

In 1926, Vernam [6] suggested an approach for enciphering large amounts of plaintext using two relatively short key-tapes. His idea was to use the two original key-tapes to produce a very long bit stream. This key, or bit stream, could then be added to the plaintext, using modulo 2 addition, to produce the ciphertext.

However, Vernam's method of producing the bit stream from two key-tapes is weak and is easily broken. Let the bits on the two key-tapes be denoted by

$$U = u_1, u_2, \ldots, u_{p_1}, u_1, u_2, \ldots, u_{p_1}, \ldots$$

$$V = v_1, v_2, \ldots, v_{p_2}, v_1, v_2, \ldots, v_{p_2}, \ldots$$

The periods (intervals between equal recurring values) of U and V are p_1 and p_2, respectively. If p_1 and p_2 are chosen to be relatively prime, the bit stream resulting from the modulo 2 addition of U and V will have a period $p = p_1p_2$. Let

$$R = r_1, r_2, \ldots, r_p, r_1, r_2, \ldots$$

denote the resulting bit stream. Assume that the number of enciphered plaintext bits is less than or equal to p, thus avoiding repetition of bits in the cryptographic bit-stream R. The operations of encipherment and decipherment are represented, as in the case of the key auto-key cipher (Chapter 2), by the following.

$$\begin{aligned} Y(i) &= X(i) \oplus R(i) \\ X(i) &= Y(i) \oplus R(i) \end{aligned} \qquad (3\text{-}1)$$

where $\oplus$ represents modulo 2 addition, and

$X(i) = x_1(i), x_2(i), \ldots, x_n(i)$	denotes an n-bit plaintext
$Y(i) = y_1(i), y_2(i), \ldots, y_n(i)$	denotes the resulting n-bit ciphertext
$R(i) = r_1(i), r_2(i), \ldots, r_n(i)$	denotes the ith subset of bits from R that is used to encipher the ith block of plaintext.

At first glance, this scheme may appear to employ a one-time tape, thus satisfying the conditions for an unbreakable cipher. In an unbreakable cipher (see Chapter 2), the key (or the bit stream R in the present case) must be randomly selected and used only once. But the bits denoted by R in the present example are not random—at best, they are only pseudo-random. And because the bits in R can be represented by a set of linear equations, the scheme can be broken by solving for the unknown key tapes U and V. While p equations can be written to describe the bits in R, the analyst must cope with no more than $p_1 + p_2 - 1$ equations to break the system, as shown below.

Let the period of the first tape be 3 and the period of the second tape be 2, and let one bit be enciphered at a time. In that case, R(i) = r(i), and hence

$$r(1) = u_1 \oplus v_1 \qquad (3\text{-}2a)$$

$$r(2) = u_2 \oplus v_2 \qquad (3\text{-}2b)$$

$$r(3) = u_3 \oplus v_1 \qquad (3\text{-}2c)$$

$$r(4) = u_1 \oplus v_2 \qquad (3\text{-}2d)$$

$$r(5) = u_2 \oplus v_1 \qquad (3\text{-}2e)$$

$$r(6) = u_3 \oplus v_2 \qquad (3\text{-}2f)$$

where it is assumed, without loss of generality, that r(1) coincides with the start of the tape. The period of R is $2 \cdot 3 = 6$.

If the opponent has a fragment of plaintext and corresponding ciphertext available for analysis (see Chapter 2), he can recover a fragment of the bit stream R by adding the plaintext to the ciphertext using modulo 2 addition. The problem then is to calculate the $p_1 + p_2$ bits comprising tapes U and V by knowing only a portion of the $p_1 p_2$ bits from R.

Since $u_i \oplus v_j = \bar{u}_i \oplus \bar{v}_j$, it follows that the unknown key tapes U and V always have two solutions:

$$u_1, u_2, \ldots, u_{p_1}; v_1, v_2, \ldots, v_{p_2}$$

and

$$\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_{p_1}; \bar{v}_1, \bar{v}_2, \ldots, \bar{v}_{p_2}$$

Thus a single bit in one of the two key-tapes can be assigned an arbitrary value (either 0 or 1). The remaining unknown bits in U and V are expressed in terms of a subset of the bits in R and the arbitrarily assigned bit in U or V. Let u_1 be the choice for the independent bit.

In the example ($p_1 = 3$, $p_2 = 2$), the dependent variables v_1, v_2, u_2, and u_3 can be represented as follows.

$$v_1 = r(1) \oplus u_1 \qquad \text{from 3-2a}$$

$$v_2 = r(4) \oplus u_1 \qquad \text{from 3-2d}$$

$$u_2 = r(2) \oplus v_2 = r(2) \oplus r(4) \oplus u_1 \qquad \text{from 3-2b}$$

$$u_3 = r(3) \oplus v_1 = r(3) \oplus r(1) \oplus u_1 \qquad \text{from 3-2c}$$

This shows that the arbitrary assignment of $u_1 = 0$ or $u_1 = 1$ and knowledge of four bits in R, namely r(1), r(2), r(3), and r(4), are enough to permit the unknown bits in U and V to be calculated. Thus to solve a pair of key-tapes, U and V, the analyst must have knowledge of as many bits in R as there are unknown bits in U and V. And the number of unknown bits in U and V is $p_1 + p_2 - 1$, since one bit can be arbitrarily assigned to either 0 or 1. U and V can then be used to calculate the remaining unknown bits in R.

With matrix notation, the set of linear equations (3-2a through 3-2f) is expressed as follows.

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} r(1) \\ r(2) \\ r(3) \\ r(4) \\ r(5) \\ r(6) \end{bmatrix} \qquad (3\text{-}3)$$

In the present example, the *rank* [7] of the (0, 1) matrix, as well as the augmented matrix [7], is four. Therefore, there are only four independent variables and only four bits in R are needed to solve for the unknown bits in U and V. In general, it can be shown that the rank of these matrices is equal to $p_1 + p_2 - 1$, which means that $p_1 + p_2 - 1$ bits in R are needed to solve for U and V.

Breaking a Key Auto-Key Cipher Using Linear Shift Registers

A linear shift register is a hardware circuit that can produce a stream of pseudo-random bits. It consists of a sequence of flip-flops[1] (Figure 3-5), denoted by FF_1, FF_2, . . . , FF_n, and n initial switch settings which are the secret key, denoted by $k_1, k_2, \ldots, k_n$, where

$k_j = 0$ if the jth switch is open

$k_j = 1$ if the jth switch is closed

At each clock pulse of the circuit, a single bit in the pseudo-random bit stream is generated, so that r(1), r(2), . . . , r(t) denotes the bits produced by the linear shift register at clock times 1, 2, . . . , t. The mathematical equations defining the operation of a 3-stage shift register and used for calculating r(1) through r(t) are given below.

The operations of encipherment and decipherment are represented, as in the case of the key auto-key cipher (Chapter 2), by the following..

$y(i) = x(i) \oplus r(i)$ encipherment

$x(i) = y(i) \oplus r(i)$ decipherment

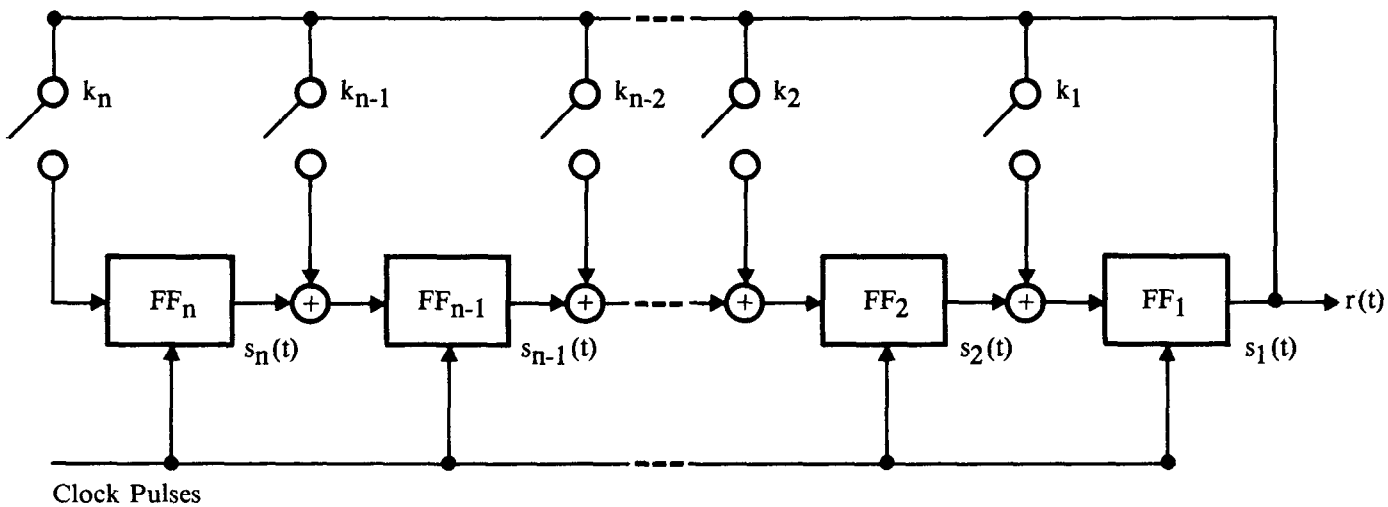
where x(i) and y(i) denote the ith plaintext and ciphertext bits, respectively. The output from the jth shift register stage (FF_j) at time t is denoted by $s_j(t)$. The initial conditions at clock time t = 1 are given by $z_j = s_j(1)$, for j from 1 to n.

It might seem that linear shift registers would give rise to strong cryptographic algorithms. A 61-stage linear shift register, for example, produces $2^{61} - 1$ different bit streams, whose period is $2^{61} - 1$ bits.[2] A communication terminal operating at a rate of 2400 bits per second would take over 30 million years to use up the entire bit system.

While the period is very long, the linear relationship among the generated bits represents a fundamental weakness. In an n-stage shift register, the bit stream is uniquely determined by the n initial conditions and n switch positions (see Figure 3-5). Thus to break the cipher, all that an opponent has to do is solve 2n independent equations involving the initial conditions and the

[1] A flip-flop is an electronic circuit having two stable states, 0 and 1, and the ability to change from one state to the other on application of a signal in a specified manner.

[2] The period of every bit stream is $2^{61} - 1$, because 61 is a Mersenne prime [8,9]. The maximum period of the bit stream is $2^n - 1$ when n is not a Mersenne prime.



The initial conditions at clock time $t = 1$ are represented by $z_j = s_j(1)$ for $j = 1, 2, \ldots, n$.

n = number of stages (flip-flops); $k_1, k_2, \ldots, k_n$ are switches where $k = 0$ if the jth switch is open and $k = 1$ if the jth switch is closed; $s_j(t)$ is the output from the jth flip-flop at time t.

Figure 3-5. Key Auto-Key Cipher Design Using an n-Stage Linear Shift Register

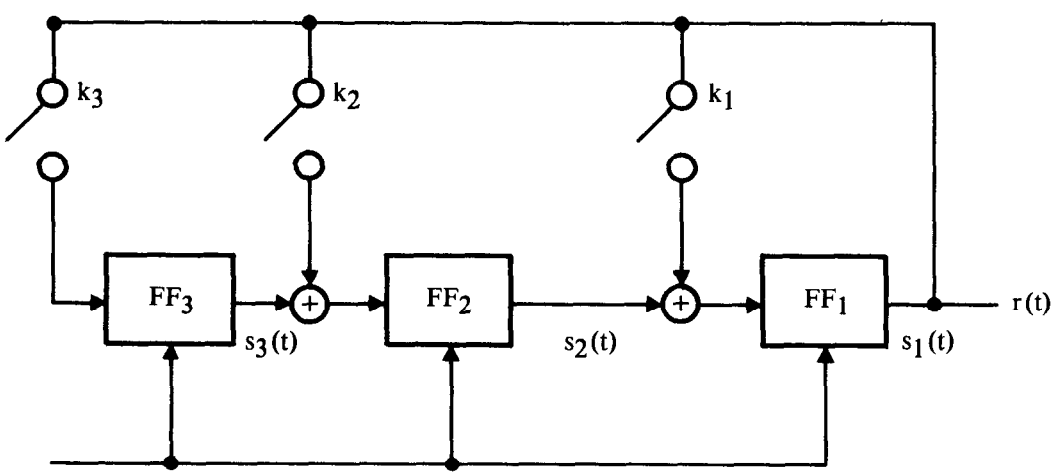
positions of the feedback switches. For all practical purposes, this could be accomplished here if 122 bits of plaintext and corresponding ciphertext were available for analysis.

To appreciate how the cipher under discussion is implemented, consider a 3-stage shift register (Figure 3-6). An analysis of the operation of the shift register at clock times $t = 1, 2, \ldots, 6$ is given in Table 3-1. Since $s_1(t) = r(t)$, it follows that

$$
\begin{aligned}
r(1) &= z_1 \\
r(2) &= z_2 \oplus k_1 r(1) \\
r(3) &= z_3 \oplus k_2 r(1) \oplus k_1 r(2) \\
r(4) &= k_3 r(1) \oplus k_2 r(2) \oplus k_1 r(3) \\
r(5) &= k_3 r(2) \oplus k_2 r(3) \oplus k_1 r(4) \\
r(6) &= k_3 r(3) \oplus k_2 r(4) \oplus k_1 r(5)
\end{aligned}
$$

This can be expressed in matrix notation as

$$
\begin{bmatrix} r(1) \\ r(2) \\ r(3) \\ r(4) \\ r(5) \\ r(6) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ r(1) & 0 & 0 & 0 & 1 & 0 \\ r(2) & r(1) & 0 & 0 & 0 & 1 \\ r(3) & r(2) & r(1) & 0 & 0 & 0 \\ r(4) & r(3) & r(2) & 0 & 0 & 0 \\ r(5) & r(4) & r(3) & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} \qquad (3\text{-}4)
$$



For $t = 1$, the initial conditions hold: $z_1 = s_1(1)$, $z_2 = s_2(1)$, $z_3 = s_3(1)$

Figure 3-6. Key Auto-Key Cipher Design Using a 3-Stage Linear Shift Register

t	$s_3(t)$	$s_3(t) \oplus k_2 r(t)$	$s_2(t)$
1	z_3	$z_3 \oplus k_2 r(1)$	z_2
2	$k_3 r(1)$	$k_3 r(1) \oplus k_2 r(2)$	$z_3 \oplus k_2 r(1)$
3	$k_3 r(2)$	$k_3 r(2) \oplus k_2 r(3)$	$k_3 r(1) \oplus k_2 r(2)$
4	$k_3 r(3)$	$k_3 r(3) \oplus k_2 r(4)$	$k_3 r(2) \oplus k_2 r(3)$
5	$k_3 r(4)$	$k_3 r(4) \oplus k_2 r(5)$	$k_3 r(3) \oplus k_2 r(4)$
6	$k_3 r(5)$	$k_3 r(5) \oplus k_2 r(6)$	$k_3 r(4) \oplus k_2 r(5)$

t	$s_2(t) \oplus k_1 r(t)$	$s_1(t)$
1	$z_2 \oplus k_1 r(1)$	z_1
2	$z_3 \oplus k_2 r(1) \oplus k_1 r(2)$	$z_2 \oplus k_1 r(1)$
3	$k_3 r(1) \oplus k_2 r(2) \oplus k_1 r(3)$	$z_3 \oplus k_2 r(1) \oplus k_1 r(2)$
4	$k_3 r(2) \oplus k_2 r(3) \oplus k_1 r(4)$	$k_3 r(1) \oplus k_2 r(2) \oplus k_1 r(3)$
5	$k_3 r(3) \oplus k_2 r(4) \oplus k_1 r(5)$	$k_3 r(2) \oplus k_2 r(3) \oplus k_1 r(4)$
6	$k_3 r(4) \oplus k_2 r(5) \oplus k_1 r(6)$	$k_3 r(3) \oplus k_2 r(4) \oplus k_1 r(5)$

Initial conditions: $z_1 = s_1(1)$, $z_2 = s_2(1)$, $z_3 = s_3(1)$

Table 3-1. Analysis of a Key Auto-Key Cipher that Uses a 3-Stage Linear Shift Register

While it is possible to solve the k_1, k_2, k_3, z_1, z_2, and z_3 in terms of bits r(1) through r(6) by using the matrices above, generally, one is not interested in the initial conditions z_1, z_2, and z_3. Thus, the key can be found using the reduced form:

$$\begin{bmatrix} r(4) \\ r(5) \\ r(6) \end{bmatrix} = \begin{bmatrix} r(3) & r(2) & r(1) \\ r(4) & r(3) & r(2) \\ r(5) & r(4) & r(3) \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} \tag{3-5}$$

Although the approach seems straightforward, it should be noted that a solution may not exist for certain values of r(1) through r(6). Before elaborating on this idea, assume (without loss of generality) that $k_3 = 1$.

When $k_3 = 1$, only r(1) through r(5) must be known before one can solve for k_1 and k_2 (i.e., only five independent equations are needed). Hence Equation 3-5 can be reduced to the following:

$$\begin{bmatrix} r(4) \oplus r(1) \\ r(5) \oplus r(2) \end{bmatrix} = \begin{bmatrix} r(3) & r(2) \\ r(4) & r(3) \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} \tag{3-6}$$

Solving for k_1 and k_2 using Cramer's Rule, one obtains

$$k_1 = \frac{\begin{vmatrix} r(4) \oplus r(1) & r(2) \\ r(5) \oplus r(2) & r(3) \end{vmatrix}}{\begin{vmatrix} r(3) & r(2) \\ r(4) & r(3) \end{vmatrix}} = \frac{r(2)[r(5) \oplus r(2)] \oplus r(3)[r(4) \oplus r(1)]}{r(2)r(4) \oplus r(3)}$$

$$k_2 = \frac{\begin{vmatrix} r(3) & r(4) \oplus r(1) \\ r(4) & r(5) \oplus r(2) \end{vmatrix}}{\begin{vmatrix} r(3) & r(2) \\ r(4) & r(3) \end{vmatrix}} = \frac{r(3)[r(5) \oplus r(2)] \oplus r(4)[r(4) \oplus r(1)]}{r(2)r(4) \oplus r(3)}$$

provided that $r(2)r(4) \oplus r(3) \neq 0$. The denominator of the two expressions above is zero if either

1. $r(3) = 0$ and $r(2)r(4) = 00, 01$, or 10
2. $r(3) = 1$ and $r(2)r(4) = 11$

However, even in the situation where $r(2)r(3)r(4) = 001$ or 100, a solution can still be found. When $r(2)r(3)r(4) = 001$, Equation 3-6 is written as

$$\begin{bmatrix} 1 \oplus r(1) \\ r(5) \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix}$$

which results in

$k_1 = r(5)$ and $k_2 =$ any value (i.e., k_2 has no unique solution, denoted no solution)

When $r(2)r(3)r(4) = 100$, Equation 3-6 is written as

$$\begin{bmatrix} r(1) \\ 1 \oplus r(5) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix}$$

which results in

$k_1 =$ any value and $k_2 = r(1)$

A summary of these results is given in Table 3-2.

Table 3-3 shows the bit streams produced by the 3-stage linear shift register (calculated in terms of the initial conditions, z_1, z_2, and z_3, by using Equation 3-4), where $k_3 = 1$ and k_1 and k_2 are variables. The maximum period of $2^3 - 1 = 7$ is obtained for keys (switch positions) 101 and 110, while keys 111 and 100 result in bit streams with periods of 4 and 3, respectively.

	r(2) r(3) r(4)	r(2) r(3) r(4)
	010 011 101 110	000 001 100 111
	$D = r(2)r(4) \oplus r(3) \neq 0$	$D = r(2)r(4) \oplus r(3) = 0$
Expression For k_1	$(1/D)[r(2)[r(5) \oplus r(2)] \oplus r(3)[r(4) \oplus r(1)]]$	* r(5) * *
Expression For k_2	$(1/D)[r(3)[r(5) \oplus r(2)] \oplus r(4)[r(4) \oplus r(1)]]$	* * r(1) *

Note: It was assumed that $k_3 = 1$; * denotes no solution

Table 3-2. Solutions for the Key in a Key Auto-Key Cipher Using a 3-Stage Linear Shift Register

Numerical values for the output bits r(1) through r(5) in Table 3-3 can be calculated by assigning z_1, z_2, and z_3 each of their possible values of 0 and 1. The results of such a calculation, where the five output bits are expressed in decimal notation for ease of presentation, are shown in Table 3-4. Out of the 32 possible values for bits r(1) through r(5), the values 1, 2, 3, 8, 15, 16, 17, 24, and 30 do not occur. The values 0, 9, 14, 18, 25, and 31 occur respectively, 4, 2, 2, 2, 2, and 2 times. As the number of stages is increased, the frequency of each value, r(1) through r(5), will be about the same, and hence the distribution will become more uniform.

r(t)	Switch Position Combinations: k_3, k_2, k_1			
	100	101	110	111
r(1)	z_1	z_1	z_1	z_1
r(2)	z_2	$z_1 \oplus z_2$	z_2	$z_1 \oplus z_2$
r(3)	z_3	$z_1 \oplus z_2 \oplus z_3$	$z_1 \oplus z_3$	$z_2 \oplus z_3$
r(4)		$z_2 \oplus z_3$	$z_1 \oplus z_2$	z_3
r(5)		$z_1 \oplus z_3$	$z_1 \oplus z_2 \oplus z_3$	
r(6)		z_2	$z_2 \oplus z_3$	
r(7)		z_3	z_3	

It is assumed that $k_3 = 1$; the periods for the bit streams produced by the keys 100, 101, 110, and 111 are 3, 7, 7, and 4, respectively.

Table 3-3. Pseudo-Random Bit Streams Produced by a 3-Stage Linear Shift Register for Different Switch Positions (Keys)

Initial Conditions	Feedback Switch Position (k_1, k_2)			
	00	01	10	11
z_1 z_2 z_3	Output r(1) through r(5) for given values of k's and z's			
0 0 0	0	0	0	0
0 0 1	4	5	7	6
0 1 0	9	11	14	12
0 1 1	13	14	9	10
1 0 0	18	23	29	25
1 0 1	22	18	26	31
1 1 0	27	28	19	21
1 1 1	31	25	20	19

Note: the value of r(1), r(2), ..., r(5) for $z_1z_2z_3 = 100$ and $k_1k_2 = 01$ is 010111 in base 2 and 23 in base 10

Table 3-4. Output from the First Stage of a 3-Stage Linear Shift Register

The mathematical equations for k_1 and k_2 given in Table 3-2 are next evaluated for each of the output values, r(1) through r(5), given in Table 3-4. From this, one gets a rough idea of the number of times that these equations either have a solution (identified by 0 or 1), or have no solution (identified by an *), as shown in Table 3-5. Since the row and column headings in Table 3-5 were chosen to match those in Table 3-4, one can estimate the probability of successfully solving for the key.

Consider the output value (r(1)r(2) $\cdots$ r(5) = 00100 (equivalent to decimal 4), which is produced by feedback switch positions $k_1k_2 = 00$ and initial conditions $z_1z_2z_3 = 001$ (see Table 3-4). Using only the output value 00100 and the equations in Table 3-2, it follows that

$$D = (0 \cdot 0) \oplus 1 = 1$$

and therefore that a unique solution for k_1 and k_2 can be obtained as

$$k_1 = (1/1)[0(0 \oplus 0) \oplus 1(0 \oplus 0)] = 0$$
$$k_2 = (1/1)[1(0 \oplus 0) \oplus 0(0 \oplus 0)] = 0$$

Hence the entry in Table 3-5 corresponding to column 00 ($k_1k_2 = 00$) and row 001 ($z_1z_2z_3 = 001$) is 00. Now consider the output value r(1)r(2) $\cdots$ r(5) = 10011 (equivalent to decimal 19), which is produced by feedback switch positions $k_1k_2 = 10$ and initial conditions $z_1z_2z_3 = 110$, or by $k_1k_2 = 11$ and $z_1z_2z_3 = 111$ (see Table 3-4). The output value 10011 and the equations in Table 3-2 lead to

$$D = (0 \cdot 1) \oplus 0 = 0$$

and therefore the equations for k_1 and k_2, as given by

$$k_1 = (1/0)[0(1 \oplus 0) \oplus 0(1 \oplus 1)] = 0/0$$
$$k_2 = (1/0)[0(1 \oplus 0) \oplus 1(1 \oplus 1)] = 0/0$$

are indeterminate forms. This condition can be resolved for k_1, since r(2)r(3) r(4) = 001 and thus k_1 = r(5) = 1 (see Table 3-2). However, there is no solution for k_2. Hence, the entries in Table 3-5 corresponding to column 10 and row 110, and column 11 and row 111, are labeled 1*.

By repetition of this approach for each entry in Table 3-4, a solution for k_1 is obtained in 22 out of 32 cases, for k_2 in 17 cases, and for both k_1 and k_2 in 14 cases (see Table 3-5). The probability of obtaining a solution is therefore about 0.5. The values of k_1k_2 that cause the output bits r(1)r(2) · · · r(5) to be equal to 00000 or 11111 (decimal 0 or 31) should not be used, since the plaintext and ciphertext are forced to be either identical or complements of each other. If those instances are excluded from Table 3-4, a solution for k_1 is obtained in 21 out of 26 cases, k_2 in 17 cases, and both k_1 and k_2 in 14 cases.

Initial Conditions for which r(1) through r(5) are Evaluated	Feedback Switch Positions (k_1, k_2) for which r(1) through r(5) are Evaluated							
	00		01		10		11	
z_1 z_2 z_3	Evaluated Values of (k_1, k_2) Based on Knowledge of r(1) through r(5)							
	k_1	k_2	k_1	k_2	k_1	k_2	k_1	k_2
0 0 0	*	*	*	*	*	*	*	*
0 0 1	0	0	0	1	1	0	1	1
0 1 0	*	0	0	1	*	*	1	1
0 1 1	0	0	*	*	*	0	1	1
1 0 0	0	*	0	1	1	0	1	1
1 0 1	0	*	0	*	1	0	1	*
1 1 0	0	*	0	*	1	*	1	1
1 1 1	*	*	*	1	1	0	1	*

P[solution for k_1] = 22/32; (21/26) (see footnote 3)
P[solution for k_2] = 17/32; (17/26) (see footnote 3)
P[solution for k_1 and k_2] = 14/32; (14/26) (see footnote 3)

* denotes no solution

Table 3-5. Solution of Feedback Switch Positions for a 3-Stage Linear Shift Register

[3] Instances where r(1) · · · r(5) = 00000 or 11111 are excluded from the calculation.

The calculation of keys—based on knowledge of r(1)r(2) · · · r(2n), where n is the number of stages—is successful in about 50 percent of the cases. The likelihood that a key can be solved for is increased if the number of bits available for analysis is greater than 2n. For example, if 2n + 9 bits are available, and consecutive bit streams of 2n bits are used, then 10 sets of 2n output bits would be available for analysis. Now, for each set of 2n output bits, assume that the probability of obtaining no solution for the key is 0.5—an assumption that is reasonable for the previous example. Thus the probability that no solution is obtained for all 10 sets of 2n output bits is $0.5^{10} = 1/1024$, and so the probability of a successful attack is 1023/1024.

When two or more linear shift registers are used one after another, the resulting algorithm still employs linear functions, and an identical method of analysis can be applied.

Breaking a Plaintext Auto-Key Cipher Using Linear Shift Registers[4]

The key auto-key cipher analyzed in the previous section does not employ feedback from either plaintext or ciphertext, but rather uses only internal feedback. To show that a similar analysis also applies to a cipher (based on linear shift registers) that uses an external feedback, a plaintext auto-key cipher is analyzed.

To reduce the number of required calculations, but still provide enough insight into the problem, the analysis is limited to a 5-stage linear shift register (n = 5), whose key is $k_1k_2 \cdot \cdot \cdot k_5 = 00101$. Enciphering and deciphering with the shift register is shown in Figures 3-7 and 3-8, respectively.

The shift-register circuits in Figures 3-7 and 3-8 can be represented analytically by the delay operator (D). If τ represents the delay of one shift register, then the transfer function of n shift register stages can be represented by $D^n\tau$. The input and output sequences are represented by $x(t - n\tau)$ and $y(t - n\tau)$, respectively.

All additions are done with Exclusive-ORs and therefore obey modulo 2 rules. All operations are linear. From Figure 3-7, it follows that

$$\begin{aligned} y(t) &= x(t)[(D^{2\tau} \oplus 1)D^{3\tau} \oplus 1] \\ &= x(t)(D^5 \oplus D^3 \oplus 1) \end{aligned} \tag{3-7}$$

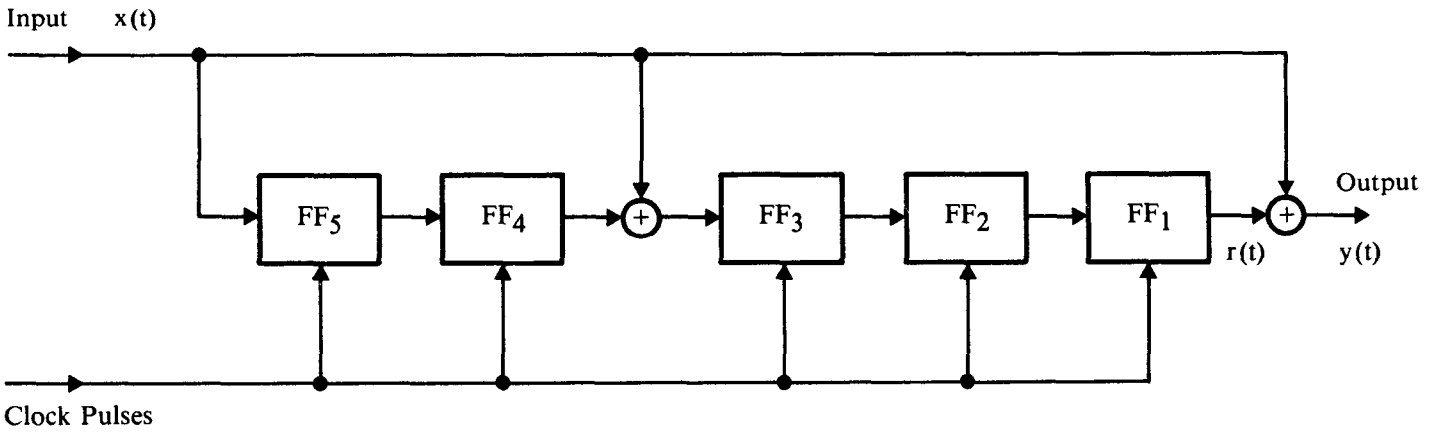
where the τ notation is dropped for purposes of convenience. Similarly, analysis of the circuit in Figure 3-8 shows that

$$\begin{aligned} w(t) &= [w(t)[(D^2 \oplus 1)D^2] \oplus y(t)]D \\ &= w(t)(D^5 \oplus D^3) \oplus y(t)D \end{aligned}$$

and so

$$y(t)D = w(t)(D^5 \oplus D^3 \oplus 1) \tag{3-8}$$

[4]© 1972 Hayden Publishing Co. The text describing the method to be used is reprinted from *Electronic Design*, November 9, 1972 [10].



$y(t) = x(t) \oplus r(t)$

Figure 3-7. Plaintext Auto-Key Cipher Operating in Encipher Mode

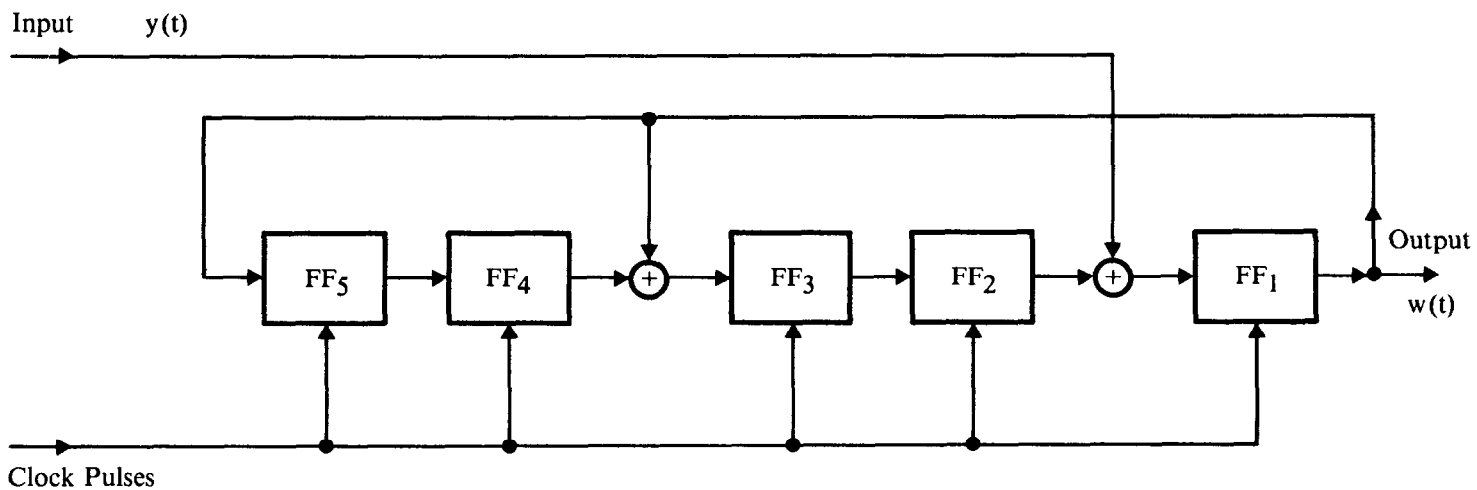
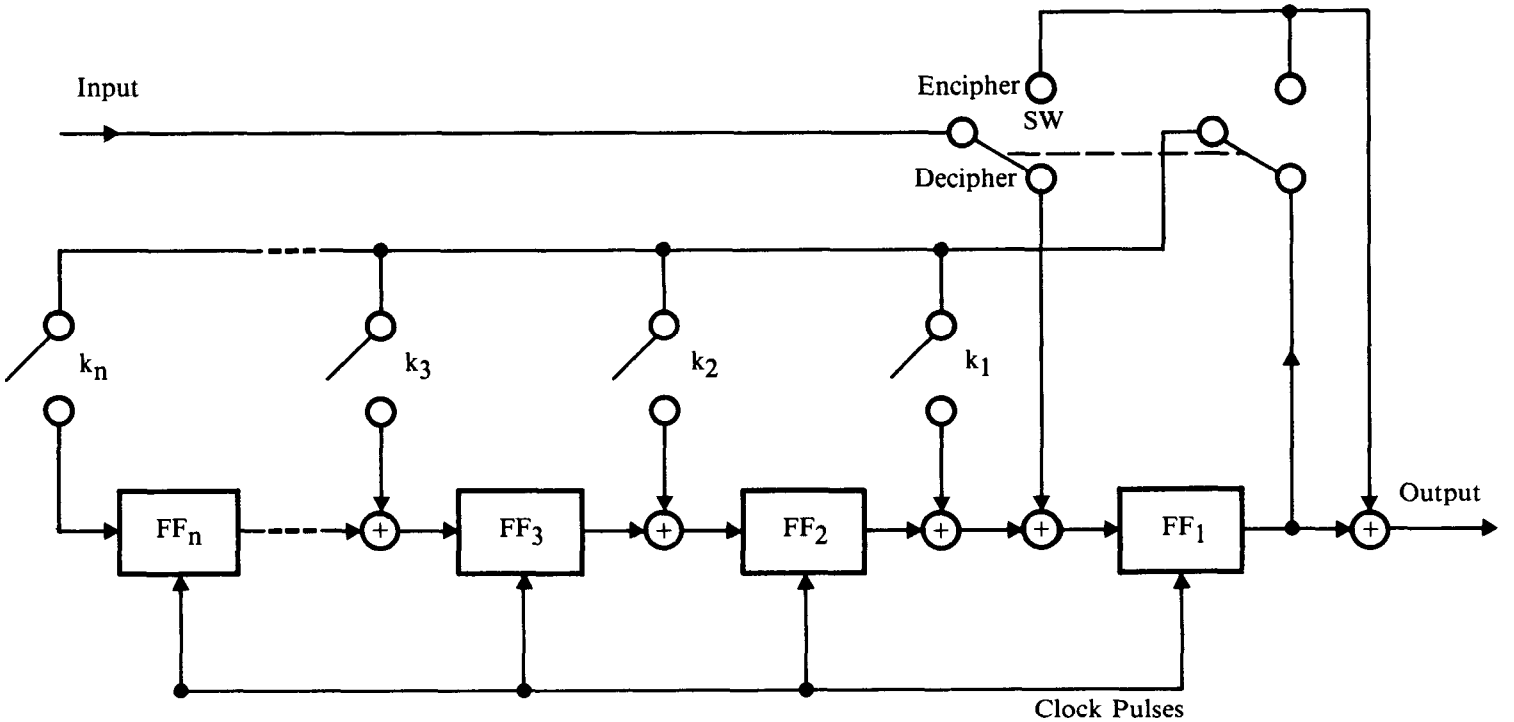


Figure 3-8. Plaintext Auto-Key Cipher Operating in Decipher Mode



Switch SW is shown set in decipher mode.

Figure 3-9. Plaintext Auto-Key Cipher Using Linear Shift Registers

Multiplying Equation 3-7 by D results in

$$y(t)D = x(t)D(D^5 \oplus D^3 \oplus 1)$$

and so, from Equation 3-8 it follows that

$$w(t) = x(t)D$$

Thus the final output of the pair of cascaded circuits shown in Figures 3-7 and 3-8 equals the input delayed by 1 bit. The original input text x(t) is correctly recovered as w(t) delayed by 1 bit.

A generalized combined encipher/decipher circuit is shown in Figure 3-9. In general, the scheme with arbitrary feedback—with switches k_1 to k_n arbitrarily set (see Figure 3-9)—can be broken with any 2n bits of clear and corresponding ciphertext. Breaking the cipher involves determining the switch settings and the initial states of the flip-flop stages. Once these conditions are known, the complete text can be deciphered.

Consider the n-stage circuit in Figure 3-9. Assume that the plaintext data consisting of bits x(1), x(2), . . . , x(2n) and the corresponding ciphertext data y(1), y(2), . . . , y(2n) are known. The most important information—the cryptographic key—is represented by the states (open and closed) of the various switches $k_1, k_2, \ldots, k_n$.

An open switch at point i is specified by $k_i = 0$, whereas a closed switch at point i is specified by $k_i = 1$. The initial states of the shift register stages are designated by $z_1, z_2, \ldots, z_n$, where each z_i can assume the values of 1 or 0.

An analysis of the 5-stage linear shift register scheme indicates that the cipher can be represented by

$$\begin{aligned}
x(1) \oplus y(1) &= z_1 \\
x(2) \oplus y(2) &= k_1x(1) \oplus z_2 \\
&\vdots \\
x(6) \oplus y(6) &= k_1x(5) \oplus k_2x(4) \oplus k_3x(3) \oplus \cdots \oplus k_5x(1) \\
&\vdots \\
x(10) \oplus y(10) &= k_1x(9) \oplus k_2x(8) \oplus k_3x(7) \oplus \cdots \oplus k_5x(5)
\end{aligned}$$

and is also represented in matrix form in Figure 3-10. All additions and multiplications are performed modulo 2.

Let the switch settings of $k_1, k_2, \ldots, k_5$ correspond to the configuration of Figure 3-7. Therefore, the ratio of y(t) to x(t) within the circuit, as previously derived in Equation 3-7, is

$$k_5 \oplus k_3 \oplus 1 = D^5 \oplus D^3 \oplus 1$$

For example, the input signal

$$x(1), x(2), \ldots, x(10) = 1011100001$$

is represented by delay operators as

$$(1 \oplus D^2 \oplus D^3 \oplus D^4 \oplus D^9)$$

Multiplying

$$(D^5 \oplus D^3 \oplus 1)(1 \oplus D^2 \oplus D^3 \oplus D^4 \oplus D^9)$$

and noting that in modulo 2 addition $D^n \oplus D^n = 0$ and ignoring values where $n > 10$ results in

$$\begin{aligned} y(1), y(2), \ldots, y(10) &= 1 \oplus D^2 \oplus D^4 \oplus D^6 \oplus D^8 \\ &= 1010101010 \end{aligned}$$

and

$$r(1), r(2), \ldots, r(10) = 0001001011$$

Although the 10 unknowns, k_1 through k_5 and z_1 through z_5, can be derived from the 10 equations represented by the matrix shown in Figure 3-10, to simplify the computation it is assumed that all register stages are initially reset ($z_1 = z_2 = \ldots = z_5 = 0$). In this case, only the entries in the lower left quarter of the matrix equation must be derived. This results in

$$\begin{bmatrix} x(6) \oplus y(6) \\ x(7) \oplus y(7) \\ x(8) \oplus y(8) \\ x(9) \oplus y(9) \\ x(10) \oplus y(10) \end{bmatrix} = \begin{bmatrix} x(5) & x(4) & x(3) & x(2) & x(1) \\ x(6) & x(5) & x(4) & x(3) & x(2) \\ x(7) & x(6) & x(5) & x(4) & x(3) \\ x(8) & x(7) & x(6) & x(5) & x(4) \\ x(9) & x(8) & x(7) & x(6) & x(5) \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \end{bmatrix}$$

Substituting values for x and y, one obtains

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \end{bmatrix}$$

which results in

$$\begin{aligned} k_1 \oplus k_2 \oplus k_3 \oplus k_5 &= 0 \\ k_2 \oplus k_3 \oplus k_4 &= 1 \\ k_3 \oplus k_4 \oplus k_5 &= 0 \\ k_4 \oplus k_5 &= 1 \\ k_5 &= 1 \end{aligned}$$

$$
\begin{bmatrix} x(1)\oplus y(1) \\ x(2)\oplus y(2) \\ x(3)\oplus y(3) \\ x(4)\oplus y(4) \\ x(5)\oplus y(5) \\ \hline x(6)\oplus y(6) \\ x(7)\oplus y(7) \\ x(8)\oplus y(8) \\ x(9)\oplus y(9) \\ x(10)\oplus y(10) \end{bmatrix} =
\left[\begin{array}{ccccc:ccccc}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
x(1) & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
x(2) & x(1) & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
x(3) & x(2) & x(1) & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
x(4) & x(3) & x(2) & x(1) & 0 & 0 & 0 & 0 & 0 & 1 \\ \hdashline
x(5) & x(4) & x(3) & x(2) & x(1) & 0 & 0 & 0 & 0 & 0 \\
x(6) & x(5) & x(4) & x(3) & x(2) & 0 & 0 & 0 & 0 & 0 \\
x(7) & x(6) & x(5) & x(4) & x(3) & 0 & 0 & 0 & 0 & 0 \\
x(8) & x(7) & x(6) & x(5) & x(4) & 0 & 0 & 0 & 0 & 0 \\
x(9) & x(8) & x(7) & x(6) & x(5) & 0 & 0 & 0 & 0 & 0
\end{array}\right]
\begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix}
$$

By arranging the enciphering circuit's variables and constants in a matrix, the established rules of matrix maniputation can be used to crack the cipher with only (2n) bits of plaintext and corresponding ciphertext (n = flip-flop stages).

Figure 3-10. Matrix Representation of the Variables and Constants for a 5-Stage Linear Shift Register Enciphering Circuit Using Plaintext Feedback

Solving these five simultaneous equations yields

$$k_1 = 0$$
$$k_2 = 0$$
$$k_3 = 1$$
$$k_4 = 0$$
$$k_5 = 1$$

Instead of this procedure, a more elegant approach is to calculate the inverse of the 5 × 5 matrix:

$$\begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

In the previous section, it was noted that some randomly chosen sets of 2n bits of plaintext and corresponding ciphertext will not provide the information needed to solve the 2n unknowns. These same considerations apply to the present situation. Hence to increase the likelihood of a solution more than 2n bits are required (an additional 9 bits will suffice in most situations).

For those applications where an accomplice can be used to send a predetermined message, a shortcut for obtaining the key can be employed. First, enough zero bits are sent to reset all the shift register stages to zero. (This has been achieved when the ciphertext consists of all zeros.) Next, any convenient message (at least n + 1 bits long) starting with a 1 bit is sent. The key can then be determined by a simple tabular approach.

Since all initial conditions are represented by zeros, the following general relationship holds:

$$x(t) \oplus y(t) = \sum_{i=1}^{n} k_i x(t - i\tau)$$

Initially, it is assumed that all possible feedback paths are used. The input message X = 1011100001 is used as before and delayed step by step, one unit at a time, up to 5 units (see Table 3-6). (Note that Y = 1010101010.) Because all shift register stages are reset to 0, zeros can be inserted to fill out the locations in front of the delayed input message. Asterisks indicate the zeros that are appended to the front of the message (see Table 3-6).

Bits in position 2 show that there are zeros in the plaintext and ciphertext. Because $x(t - \tau)$ shows a 1 in bit position 2, it can be concluded that $x(t - \tau)$ cannot contribute to the output. Hence it follows that $k_1 = 0$.

Bit Position	1	2	3	4	5	6	7	8	9	10	Conclusion
Input X(t)	1	0	1	1	1	0	0	0	0	1	
$X(t-\tau)$	*	1	0	1	1	1	0	0	0	0	$k_1 = 0$
$X(t-2\tau)$	*	*	1	0	1	1	1	0	0	0	$k_2 = 0$
$X(t-3\tau)$	*	*	*	1	0	1	1	1	0	0	$k_3 = 1$
$X(t-4\tau)$	*	*	*	*	1	0	1	1	1	0	$k_4 = 0$
$X(t-5\tau)$	*	*	*	*	*	1	0	1	1	1	$k_5 = 1$
Output Y(t)	1	0	1	0	1	0	1	0	1	0	

Table 3-6. Chosen Plaintext Attack Against a Plaintext Auto-Key Cipher

In a similar manner, $x(t - 2\tau)$ and $x(t - 4\tau)$ cannot contribute. Therefore, $k_2 = k_4 = 0$. Finally, it can be concluded that $k_3 = k_5 = 1$, a result that agrees (again) with the actual switch positions. It should be observed that only n + 1 bits of plaintext and corresponding ciphertext were needed in this case to determine the key.

Designing a Cipher[5]

The methods for attacking a cryptographic algorithm fall into two categories: cryptanalysis and "brute force," or exhaustive, methods. Exhaustive methods can be further divided into two subcategories: key exhaustion and message exhaustion.

Exhaustive attacks are easily thwarted either by adjusting certain parameters in the algorithm, such as blocksize (if a block cipher is used) and key length, and/or by restricting the way a cryptographic procedure or system uses the algorithm, such as requiring the use of initializing vectors and chaining. The major challenge in algorithm design is to devise a procedure that can withstand determined efforts at cryptanalysis.

A successful method of cryptanalysis is often called a shortcut solution. A shortcut solution is defined here as a cryptanalytic break to distinguish it from an exhaustive break. In the former case the algorithm must be redesigned, whereas in the latter case the problem may be remedied by the way the algorithm is implemented (e.g., by using block chaining, multiple encryption, and the like).

[5]© 1978 IEEE. Reprinted from *Proceedings COMPCON 78* [11].

Shortcut Methods

Cryptanalytic or shortcut methods can be divided into two subcategories: deterministic or analytical methods, and statistical methods. In a deterministic approach, the cryptanalyst first attempts to express a desired unknown quantity (such as the key or message) in terms of some other known quantity or quantities (such as given ciphertext, or given plaintext and corresponding ciphertext) whose relationship to the unknown quantity depends on the nature of the algorithm. Then the cryptanalyst solves for the unknown quantity.

Let Y denote the ciphertext produced by enciphering plaintext X with cryptographic key K, and let f_K represent the function that relates X and Y:

$$Y = f_K(X)$$

In a deterministic attack against the key, the opponent tries to find a function F, where

$$K = F(X, Y)$$

such that F can be represented by an easily computed procedure.

In a poorly designed algorithm, it may be possible to solve for the key by decoupling F into a set of equations

$$\begin{aligned} k_1 &= F_1(Y, X) \\ k_2 &= F_2(Y, X, k_1) \\ &\vdots \qquad\qquad \vdots \\ k_n &= F_n(Y, X, k_1, \ldots, k_{n-1}) \end{aligned}$$

and then to solve for the key bits $k_1, k_2, \ldots, k_n$, one at a time. While analytical methods will generally succeed in breaking an algorithm that uses linear functions, this method of attack can be effectively thwarted if the algorithm makes use of nonlinear functions of sufficient complexity.

In a statistical approach, the cryptanalyst attempts to exploit statistical relationships between plaintext, ciphertext, and key. Consider a simple substitution cipher on English text. It can be shown (Chapter 12) that about 100,000 characters of ciphertext are required to deduce the key when only letter frequency statistics are used in the analysis. However, only about 300 characters of ciphertext are required to recover the key when digram statistics are used [12]. It can be shown that the theoretical limit is about 25 characters (Chapter 12).

To thwart statistical attacks, the algorithm's output (ciphertext) should be pseudo-random. In other words, for a large set of plaintext and key inputs, one must not be able, on the basis of statistical analysis, to reject the hypothesis that the output bit stream is random.

During the validation of DES, no shortcut solution could be found by its investigators, including the NSA [13]. This same conclusion was reaffirmed

in September 1977 at a workshop conducted by the National Bureau of Standards, Institute for Computer Sciences and Technology (ICST), to investigate the complexity of the DES algorithm [14]. Though additional attempts to break DES have been made, the authors are unaware of any shortcut method that can solve for even a single bit in the key.

Brute Force Methods

In a brute force approach, one attempts to find a desired unknown quantity (such as the message or key) by using a method of direct search, trial and error, or exhaustion. In key exhaustion, a known plaintext is enciphered with a trial key and the output is compared for equality with a given ciphertext. (The attack assumes that the cryptographic algorithm is known to the opponent, and that plaintext and corresponding ciphertext are available for analysis.) If the comparison is favorable, then the trial key is a candidate for the unknown key. While in theory the correct key can always be found by repeated trials, in practice the attack is thwarted if the computational and data storage requirements are too great, or if the cost of the attack is too great.

In August 1976, ICST sponsored a workshop to determine the feasibility of building a machine that could recover a 56-bit DES key from a given fragment of plaintext and corresponding ciphertext [15]. The workshop participants were asked to design a hypothetical key-exhaustion machine using their own specialized knowledge and taking full advantage of anticipated technical advances. Factors to be considered were the architecture of such machines, types of circuitry, speed of operation, reliability and maintainability, size, power, and cooling requirements.

The members of the workshop, which included 20 representatives from industry, research organizations, universities, and government agencies, together with a number of ICST staff members, reached the following conclusion:

> A machine which finds, on the average, one key per day could probably not be built until 1990 and the probability factor of it being available even then is estimated to be between .1 and .2. In addition, the cost of such a machine would be several tens of millions of dollars.

While it is important to determine the work factor for key exhaustion, and therefore to establish how vulnerable the algorithm is to this method of attack, it is far more important to establish that the algorithm has no shortcut solution. When the algorithm has no shortcut solution, its *effective key length*, and hence the work factor to perform key exhaustion, is determined by the way the algorithm is implemented in a particular cryptographic procedure or system.

By using multiple encryption methods, DES's effective key length can be increased to any desired value [16]. And there are efficient ways to expand the key in a migratable way. This, in effect, allows strong DES-based cryptographic procedures and systems to be used for an indefinite period (as long as no shortcut solution is found).

Under the assumption that an opponent can exercise the cryptographic device containing an unknown key, it is possible in theory to build a dictionary of plaintext and corresponding ciphertext—by enciphering all possible plaintext combinations—that would then allow the opponent to recover messages without ever solving for or knowing the key. But in practice this form of message exhaustion is thwarted if the computational and data storage requirements to build and store the dictionary are too great.

To improve efficiency, exhaustive methods may use a combination of precomputed tables together with a method of direct search, although the construction of such tables may also take days, months, or even years.

Effective countermeasures to thwart exhaustive methods are obtained by

1. Making both the effective key length and blocksize (if a block cipher is used) large enough, and/or
2. "Whitening" the plaintext and/or ciphertext by adding pseudo-random "noise" to the message. In a block cipher, this can be achieved by the use of initializing vectors combined with block chaining techniques, whereas in a stream cipher, it is automatically achieved as a result of the required initializing vector (see Chapter 2).

For some applications, whitening may be necessary even if the algorithm's key length and blocksize are large. For example, if "buy" and "sell" are the only possible messages ever sent by the application, then a dictionary with only two entries would be enough to defeat the intended security, and therefore some type of whitening would be mandatory.

In summary, a well-designed cryptographic algorithm is one that will withstand all known shortcut and brute force methods of solution. But it should also be realized that if an algorithm has no shortcut solution, then it can always be implemented in such a way that the minimum work factor of all brute force attacks is larger than any desired value.

Classified Design Principles

History has shown that many supposedly strong ciphers were broken using cryptanalysis (e.g., the Japanese PURPLE cipher and the German Enigma cipher used during World War II). These ciphers were cracked despite the fact that the design principles and the methods of analysis to validate the algorithm's strength were not available to the cryptanalyst. In fact, it can be argued that a cryptanalyst would be better off unprejudiced by knowledge of the design principles and methods of analysis used to validate the algorithm's strength.

Some of the methods of analysis used by IBM to validate the DES, and all such methods used by NSA, have been classified by the U.S. government. Therefore, some critics of DES have inferred that one cannot be sure that statements by IBM and NSA about DES are as claimed. To answer this criticism, the Senate Committee on Intelligence conducted an investigation into the matter. The following summarizes the conclusions that were reached [13].

1. DES is more than adequate for its intended applications.
2. IBM invented and designed DES.
3. NSA did not tamper with the design.
4. NSA certified that the DES was free of any known statistical/mathematical weakness.
5. NSA recommended that the Federal Reserve Board use DES for electronic funds transfer applications.

DESCRIPTION OF THE DATA ENCRYPTION STANDARD

Although a complete description of the DES algorithm has been published [5], the reader may find that treatment difficult to follow. For this reason, a more detailed description of the algorithm is given here, providing the reader with a greater insight and understanding of DES's operation. A numerical example of a one-round encryption is also given. (Frequent references are made to the description of the DES, which is reprinted in its entirety, including original page numbers, in Appendix A.)

A block cipher design consisting of n rounds of encipherment/decipherment is described in Chapter 2. The steps performed at each round are summarized below.

1. The input block is split into two parts, a left half and a right half.
2. The right half (step 1) is then operated on using a cipher function g (see Figures 2-5 and 2-6).
3. This output (step 2) is combined (via an Exclusive-OR operation) with the left half (step 1).

The particular design has the property that the ciphering process can be reversed regardless of the nature of function g. This is accomplished merely by reversing the order in which the keys are exercised at each round in the ciphering process. For example, if during encipherment the keys are exercised in the order K(1) through K(n), then during decipherment they must be exercised in the order K(n) through K(1).

The block cipher design described above is also used in DES, where n = 16 and K(1) through K(16) are 48-bit keys. For reasons of security, all of these keys should be different. This is achieved by selecting, at each round, a different subset of 48 bits from the 56-bit key supplied to the algorithm. This procedure (key schedule calculation) is based on a simple shifting and bit-selection algorithm. Figure 3-11 illustrates the key schedule calculation used for encipherment. For decipherment, left shifts become right shifts, except for the shift performed between (C_0, D_0) and (C_1, D_1), which is not required. (See the discussion below.)

The key schedule calculation begins with an initial permutation defined by permuted choice 1, PC-1 (see page 16 of [5]). PC-1, which is the same for encipherment and decipherment, selects 56 of the 64 external key bits (in

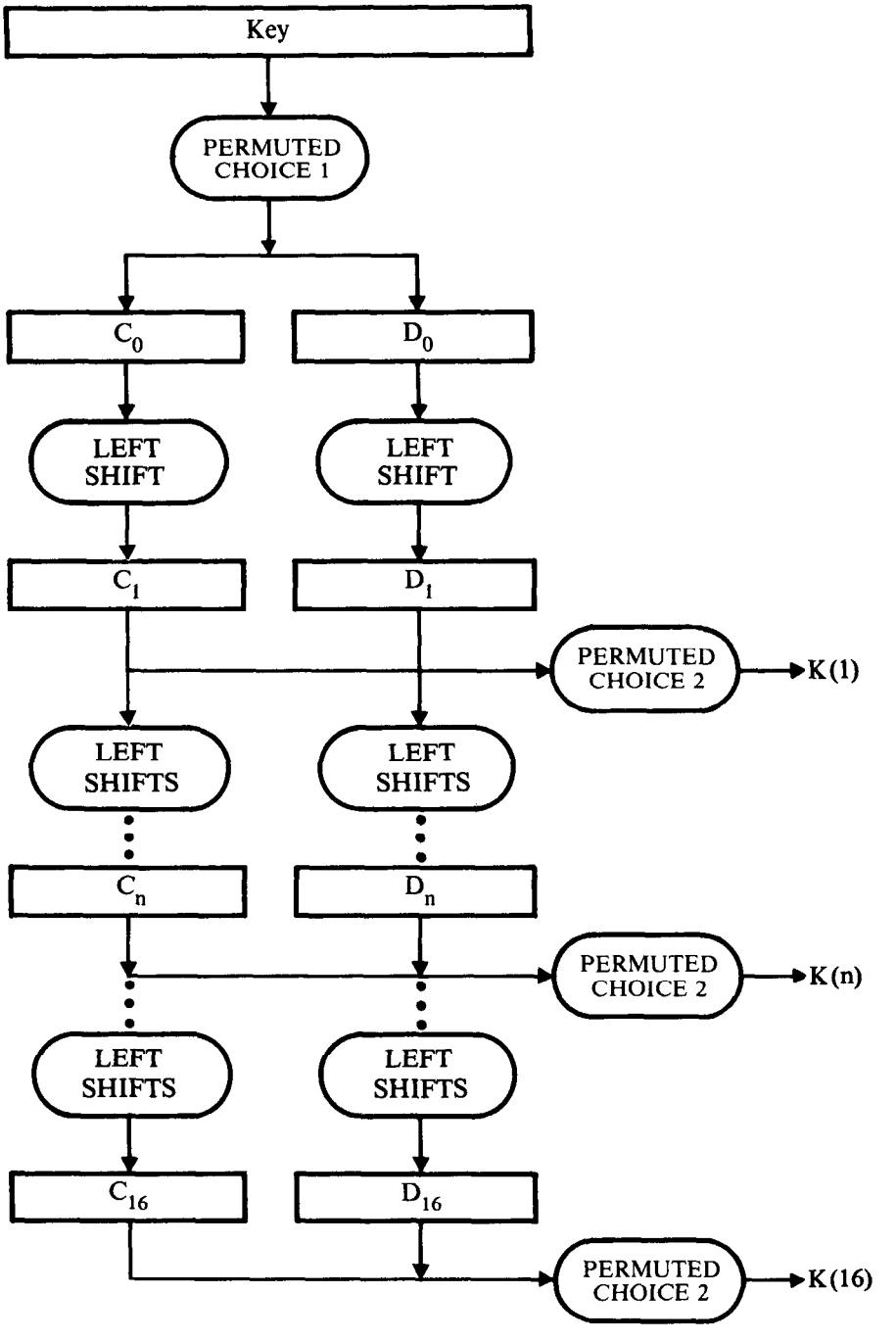


Figure 3-11. Key Schedule Calculation for Encipherment

Iteration Number i	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Table 3-7. Shift Schedule for Encipherment

effect stripping off the parity bits) and loads them into two 28-bit shift registers (C and D). Thus parity checking of the external key must be performed prior to PC-1.

During an enciphering operation, the contents of registers C_{i-1} and D_{i-1} are shifted one or two positions to the left, according to the schedule of left shift operations (page 18 of [5]), as shown in Table 3-7. K(i) is then derived from (C_i, D_i) via a second permutation defined by permuted choice 2, PC-2 (page 18 of [5]). Moreover, the shift schedule is such that $C_{16} = C_0$ and $D_{16} = D_0$.

During a decipher operation, K(16) must be used in round one, K(15) in round two, and so forth. But the contents of registers C_0 and D_0 are the same for enciphering and deciphering, since the external key is loaded in both cases via PC-1. This means that K(16) can be created at round one merely by omitting the first shift operation, K(15) can be created at round two by shifting C_0 (C_{16}) and D_0(D_{16}) one bit to the right, and the remainder of the internal keys can be created in the same manner using the shift schedule in Table 3-7, except that left shifts are changed to right shifts.

Generation of Key Vectors Used for Each Round of DES

Let the externally entered key K (including parity bits stripped off before the key is used by DES) be defined as

$$K = k_1, k_2, \ldots, k_{64}$$

The permuted choice PC-1 (page 16 of [5]) determines how 56 of these initial 64 bits are loaded into two 28-bit registers, C_0 and D_0:

$$\begin{aligned} C_0 = {} & k_{57}, k_{49}, k_{41}, k_{33}, k_{25}, k_{17}, k_9, \\ & k_1, k_{58}, k_{50}, k_{42}, k_{34}, k_{26}, k_{18}, \\ & k_{10}, k_2, k_{59}, k_{51}, k_{43}, k_{35}, k_{27}, \\ & k_{19}, k_{11}, k_3, k_{60}, k_{52}, k_{44}, k_{36}, \\ D_0 = {} & k_{63}, k_{55}, k_{47}, k_{39}, k_{31}, k_{23}, k_{15}, \\ & k_7, k_{62}, k_{54}, k_{46}, k_{38}, k_{30}, k_{22}, \\ & k_{14}, k_6, k_{61}, k_{53}, k_{45}, k_{37}, k_{29}, \\ & k_{21}, k_{13}, k_5, k_{28}, k_{20}, k_{12}, k_4 \end{aligned}$$

It can be seen that bits 57, 49, and 41 of K are the first, second, and third bits of register C_0, respectively, while bits 63, 55, and 47 of K are the first, second, and third bits of register D_0, respectively. It can also be observed that the 64 input bits have been reduced to 56 bits, because the parity bits k_8, k_{16}, k_{24}, k_{32}, k_{40}, k_{48}, k_{56}, and k_{64} have been systematically removed as part of the initial loading process.

The key vectors K(1), K(2), . . . , k(16), which consist of 48 key bits each, cannot be created until the vectors C_1 through C_{16} and D_1 through D_{16} (Figure 3-11), which consist of 28 key bits, have been formed. These vectors are derived by using the schedule given on page 18 of [5] and Table 3-7.

Let (C_i, D_i) denote the concatenation of registers C_i and D_i. In general, (C_{i+1}, D_{i+1}) is produced from (C_i, D_i) by shifting the bits in C_i and D_i, respectively, one or two positions to the left. The shifting employs wrap-around (i.e., bits shifted off the left side of the register are reinserted at the right side of the register). The results are shown in Tables 3-8 and 3-9. As an example, it can be seen that $C_1, D_1)$ is derived from (C_0, D_0) by shifting the bits in C_0 and D_0 one position to the left.

Permuted choice 2 (PC-2) is the rule (page 18 of [5]) that defines how the 48-bit key vectors K(1), K(2), . . . , K(16) are derived from the vectors (C_1, D_1), (C_2, D_2), . . . , (C_{16}, D_{16}), respectively. The bit patterns stored in registers C_i and D_i are referred to here also as vectors C_i and D_i, respectively. Specifically, K(i) is derived from (C_i, D_i) by taking the key bits located in positions

$$\begin{aligned} &14, 17, 11, 24, \ 1, 5, \ 3, 28, 15, \ 6, 21, 10 \\ &23, 19, 12, \ 4, 26, 8, 16, \ 7, 27, 20, 13, \ 2 \end{aligned} \qquad (3\text{-}9a)$$

from vector C_i and concatenating them with the key bits located in positions

$$\begin{aligned} &41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48 \\ &44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32 \end{aligned} \qquad (3\text{-}9b)$$

from vector D_i.

Round (i)	Index of Elements in Vector C_i																											Round (1)
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	
1	49	41	33	25	17	9	1	58	50	42	34	26	18	10	2	59	51	43	35	27	19	11	3	60	52	44	36	57	1
2	41	33	25	17	9	1	58	50	42	34	26	18	10	2	59	51	43	35	27	19	11	3	60	52	44	36	57	49	2
3	25	17	9	1	58	50	42	34	26	18	10	2	59	51	43	35	27	19	11	3	60	52	44	36	57	49	41	33	3
4	9	1	58	50	42	34	26	18	10	2	59	51	43	35	27	19	11	3	60	52	44	36	57	49	41	33	25	17	4
5	58	50	42	34	26	18	10	2	59	51	43	35	27	19	11	3	60	52	44	36	57	49	41	33	25	17	9	1	5
6	42	34	26	18	10	2	59	51	43	35	27	19	11	3	60	52	44	36	57	49	41	33	25	17	9	1	58	50	6
7	26	18	10	2	59	51	43	35	27	19	11	3	60	52	44	36	57	49	41	33	25	17	9	1	58	50	42	34	7
8	10	2	59	51	43	35	27	19	11	3	60	52	44	36	57	49	41	33	25	17	9	1	58	50	42	34	26	18	8
9	2	59	51	43	35	27	19	11	3	60	52	44	36	57	49	41	33	25	17	9	1	58	50	42	34	26	18	10	9
10	51	43	35	27	19	11	3	60	52	44	36	57	49	41	33	25	17	9	1	58	50	42	34	26	18	10	2	59	10
11	35	27	19	11	3	60	52	44	36	57	49	41	33	25	17	9	1	58	50	42	34	26	18	10	2	59	51	43	11
12	19	11	3	60	52	44	36	57	49	41	33	25	17	9	1	58	50	42	34	26	18	10	2	59	51	43	35	27	12
13	3	60	52	44	36	57	49	41	33	25	17	9	1	58	50	42	34	26	18	10	2	59	51	43	35	27	19	11	13
14	52	44	36	57	49	41	33	25	17	9	1	58	50	42	34	26	18	10	2	59	51	43	35	27	19	11	3	60	14
15	36	57	49	41	33	25	17	9	1	58	50	42	34	26	18	10	2	59	51	43	35	27	19	11	3	60	52	44	15
16	57	49	41	33	25	17	9	1	58	50	42	34	26	18	10	2	59	51	43	35	27	19	11	3	60	52	44	36	16

k_{49}, k_{41}, k_{33}, ..., etc. are the 1st, 2nd, 3rd, ..., etc. key bits in C_1, i.e., in register C during the 1st round.

Table 3-8. Key Bits Stored in Register (C) for Each Individual Round

Round (i)	Index of Elements in Vector D_i																											Round (i)
	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	
1	55	47	39	31	23	15	7	62	54	46	38	30	22	14	6	61	53	45	37	29	21	13	5	28	20	12	4	63	1
2	47	39	31	23	15	7	62	54	46	38	30	22	14	6	61	53	45	37	29	21	13	5	28	20	12	4	63	55	2
3	31	23	15	7	62	54	46	38	30	22	14	6	61	53	45	37	29	21	13	5	28	20	12	4	63	55	47	39	3
4	15	7	62	54	46	38	30	22	14	6	61	53	45	37	29	21	13	5	28	20	12	4	63	55	47	39	31	23	4
5	62	54	46	38	30	22	14	6	61	53	45	37	29	21	13	5	28	20	12	4	63	55	47	39	31	23	15	7	5
6	46	38	30	22	14	6	61	53	45	37	29	21	13	5	28	20	12	4	63	55	47	39	31	23	15	7	62	54	6
7	30	22	14	6	61	53	45	37	29	21	13	5	28	20	12	4	63	55	47	39	31	23	15	7	62	54	46	38	7
8	14	6	61	53	45	37	29	21	13	5	28	20	12	4	63	55	47	39	31	23	15	7	62	54	46	38	30	22	8
9	6	61	53	45	37	29	21	13	5	28	20	12	4	63	55	47	39	31	23	15	7	62	54	46	38	30	22	14	9
10	53	45	37	29	21	13	5	28	20	12	4	63	55	47	39	31	23	15	7	62	54	46	38	30	22	14	6	61	10
11	37	29	21	13	5	28	20	12	4	63	55	47	39	31	23	15	7	62	54	46	38	30	22	14	6	61	53	45	11
12	21	13	5	28	20	12	4	63	55	47	39	31	23	15	7	62	54	46	38	30	22	14	6	61	53	45	37	29	12
13	5	28	20	12	4	63	55	47	39	31	23	15	7	62	54	46	38	30	22	14	6	61	53	45	37	29	21	13	13
14	20	12	4	63	55	47	39	31	23	15	7	62	54	46	38	30	22	14	6	61	53	45	37	29	21	13	5	28	14
15	4	63	55	47	39	31	23	15	7	62	54	46	38	30	22	14	6	61	53	45	37	29	21	13	5	28	20	12	15
16	63	55	47	39	31	23	15	7	62	54	46	38	30	22	14	6	61	53	45	37	29	21	13	5	28	20	12	4	16

Table 3-9. Key Bits Stored in Register (D) for Each Individual Round

This rule used in conjunction with the C_i and D_i vectors given in Tables 3-8 and 3-9 allows the evaluation of key vector K(i). The first 24 bits of K(i) which are derived from C_i are shown in Table 3-10. The second 24 bits of K(i) which are derived from D_i are shown in Table 3-11. The total key vector K(i) associated with the ith round is thus obtained by concatenating row i from Table 3-10 with row i from Table 3-11. The entries in Tables 3-10 and 3-11 correspond to the indices of the associated key bits within the key $k_1, k_2, \ldots, k_{64}$ which are to be used. For example, from Tables 3-10 and 3-11, it can be seen that

$$\begin{aligned} K(1) &= k_{10}, k_{51}, \ldots, k_{41}, k_{22}, k_{28}, \ldots, k_{31} \\ K(2) &= k_2, \ k_{43}, \ldots, k_{33}, k_{14}, k_{20}, \ldots, k_{23} \\ &\vdots \qquad\qquad\qquad\qquad \vdots \\ K(16) &= k_{18}, k_{59}, \ldots, k_{49}, k_{30}, k_5, \ \ldots, k_{39} \end{aligned}$$

Observe that $C_{16} = C_0$ and $D_{16} = D_0$. This results in the following difference between encipherment and decipherment. Since the externally supplied key is the same for encipherment and decipherment, the bit patterns in registers C_0 and D_0 are independent of the chosen operation. During encipherment, the process starts with K(1). This requires that (C_1, D_1) be created from (C_0, D_0) by one left shift, and K(1) be created from (C_1, D_1). During decipherment, the process starts with K(16). However, since (C_{16}, D_{16}) equals (C_0, D_0), it follows that K(16) can be created from (C_0, D_0) directly (i.e., an initial left shift is not required). The shift schedule is, of course, traced then in reverse order, left shifts becoming right shifts.

Weak and Semiweak Keys

It must be realized that the mathematical complexity of the DES algorithm, and hence its cryptographic strength, would be reduced if the internal keys at each round were the same. For this reason, the condition $K(1) = K(2) = \ldots = K(16)$ should be avoided.

There is, however, a set of *weak* keys within DES which satisfy the above condition. This occurs whenever the bits in register C are all ones or zeros, and the bits in register D are all ones or zeros. In this case, C_1 and D_1 are, respectively (see Tables 3-8 and 3-9),

$$k_{49} = k_{41} = \ldots = k_{57} = 0 \text{ or } 1$$

and

$$k_{55} = k_{47} = \ldots = k_{63} = 0 \text{ or } 1$$

Round (i)	Index of Elements in Vector K(i)																								Round (i)
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	
	Index of Selected Element in Vector C_i (Obtained from PC-2)																								
	14	17	11	24	1	5	3	28	15	6	21	10	23	19	12	4	26	8	16	7	27	20	13	2	
1	10	51	34	60	49	17	33	57	2	9	19	42	3	35	26	25	44	58	59	1	36	27	18	41	1
2	2	43	26	52	41	9	25	49	59	1	11	34	60	27	18	17	36	50	51	58	57	19	10	33	2
3	51	27	10	36	25	58	9	33	43	50	60	18	44	11	2	1	49	34	35	42	41	3	59	17	3
4	35	11	59	49	9	42	58	17	27	34	44	2	57	60	51	50	33	18	19	26	25	52	43	1	4
5	19	60	43	33	58	26	42	1	11	18	57	51	41	44	35	34	17	2	3	10	9	36	27	50	5
6	3	44	27	17	42	10	26	50	60	2	41	35	25	57	19	18	1	51	52	59	58	49	11	34	6
7	52	57	11	1	26	59	10	34	44	51	25	19	9	41	3	2	50	35	36	43	42	33	60	18	7
8	36	41	60	50	10	43	59	18	57	35	9	3	58	25	52	51	34	19	49	27	26	17	44	2	8
9	57	33	52	42	2	35	51	10	49	27	1	60	50	17	44	43	26	11	41	19	18	9	36	59	9
10	41	17	36	26	51	19	35	59	33	11	50	44	34	1	57	27	10	60	25	3	2	58	49	43	10
11	25	1	49	10	35	3	19	43	17	60	34	57	18	50	41	11	59	44	9	52	51	42	33	27	11
12	9	50	33	59	19	52	3	27	1	44	18	41	2	34	25	60	43	57	58	36	35	26	17	11	12
13	58	34	17	43	3	36	52	11	50	57	2	25	51	18	9	44	27	41	42	49	19	10	1	60	13
14	42	18	1	27	52	49	36	60	34	41	51	9	35	2	58	57	11	25	26	33	3	59	50	44	14
15	26	2	50	11	36	33	49	44	18	25	35	58	19	51	42	41	60	9	10	17	52	43	34	57	15
16	18	59	42	3	57	25	41	36	10	17	27	50	11	43	34	33	52	1	2	9	44	35	26	49	16

Table 3-10. First Set of 24 Key Bits in K(i), the Key Used at Round (i)

Round (i)	Index of Elements in Vector K(i)																								Round (i)
	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	
	Index of Selected Element in Vector D_i (Obtained from PC-2)																								
	41	52	31	37	47	55	30	40	51	45	33	48	44	49	39	56	34	53	46	42	50	36	29	32	
1	22	28	39	54	37	4	47	30	5	53	23	29	61	21	38	63	15	20	45	14	13	62	55	31	1
2	14	20	31	46	29	63	39	22	28	45	15	21	53	13	30	55	7	12	37	6	5	54	47	23	2
3	61	4	15	30	13	47	23	6	12	29	62	5	37	28	14	39	54	63	21	53	20	38	31	7	3
4	45	55	62	14	28	31	7	53	63	13	46	20	21	12	61	23	38	47	5	37	4	22	15	54	4
5	29	39	46	61	12	15	54	37	47	28	30	4	5	63	45	7	22	31	20	21	55	6	62	38	5
6	13	23	30	45	63	62	38	21	31	12	14	55	20	47	29	54	6	15	4	5	39	53	46	22	6
7	28	7	14	29	47	46	22	5	15	63	61	39	4	31	13	38	53	62	55	20	23	37	30	6	7
8	12	54	61	13	31	30	6	20	62	47	45	23	55	15	28	22	37	46	39	4	7	21	14	53	8
9	4	46	53	5	23	22	61	12	54	39	37	15	47	7	20	14	29	38	31	63	62	13	6	45	9
10	55	30	37	20	7	6	45	63	38	23	21	62	31	54	4	61	13	22	15	47	46	28	53	29	10
11	39	14	21	4	54	53	29	47	22	7	5	46	15	38	55	45	28	6	62	31	30	12	37	13	11
12	23	61	5	55	38	37	13	31	6	54	20	30	62	22	39	29	12	53	46	15	14	63	21	28	12
13	7	45	20	39	22	21	28	15	53	38	4	14	46	6	23	13	63	37	30	62	61	47	5	12	13
14	54	29	4	23	6	5	12	62	37	22	55	61	30	53	7	28	47	21	14	46	45	31	20	63	14
15	38	13	55	7	53	20	63	46	21	6	39	45	14	37	54	12	31	5	61	30	29	15	4	47	15
16	30	5	47	62	45	12	55	38	13	61	31	37	6	29	46	4	23	28	53	22	21	7	63	39	16

Table 3-11. Second Set of 24 Key Bits in K(i), the Key Used at Round (i)

Thus there are four weak keys altogether and they are represented by the following (parity-adjusted) external keys:

01	01	01	01	01	01	01	01
1F	1F	1F	1F	0E	0E	0E	0E
E0	E0	E0	E0	F1	F1	F1	F1
FE	FE	FE	FE	FE	FE	FE	FE

Weak keys also have the property that there is no difference between the operations of encipherment and decipherment. Whereas, in general, the relations $D_K E_K(X) = X$ and $E_K D_K(X) = X$ hold for any key K and plaintext X, the special relation $E_K E_K(X) = X$ also holds for weak keys. Moreover, since $D_K D_K E_K E_K(X) = X$, the relation $D_K D_K(X) = X$ also holds.

There is another set of keys defined as *semiweak*. These have the property that only two different internal keys are produced, each occurring eight times. A semiweak key occurs whenever

1. Register C or D contains bit pattern 0101 . . . 0101 or 1010 . . . 1010.
2. The other register (D or C) contains bit pattern 0000 . . . 0000, 1111 . . . 1111, 0101 . . . 0101, or 1010 . . . 1010.

An alternating sequence of 0 and 1 bits has the interesting property that no matter how the sequence is shifted only two bit patterns are produced in registers C and D, namely, 0101 . . . 0101 and 1010 . . . 1010.

By way of illustration, let register C_0 contain 1010 . . . 1010 and register D_0 contain 0101 . . . 0101. The bit patterns shown in Table 3-12, from which the internal keys are derived, are produced in registers C_i and D_i.

In the above example, C_1 and D_1 (see Tables 3-8 and 3-9) are, respectively,

$$k_{49}, k_{41}, \ldots, k_{36}, k_{57} = 01 \ldots 01$$

and

$$k_{55}, k_{47}, \ldots, k_4, k_{63} = 10 \ldots 10$$

which results in (parity-adjusted) external key

$$K_{01,10} = 1F\ \ E0\ \ 1F\ \ E0\ \ 0E\ \ F1\ \ 0E\ \ F1$$

where the subscripts (01 and 10) indicate the repeating bit patterns in C_1 and D_1.

Let K denote $K_{01,10}$. Note that one can recover plaintext also by applying the encipher operation, since there is a K′ (where $K' \neq K$) whose internal keys satisfy the relation

$$K'(i) = K(17 - i) \qquad \text{for } i = 1, 2, \ldots, 16$$

Index i	Number of Left Shifts	Content of Register C_i (28 bits)	Content of Register D_i (28 bits)
0		1010...1010	0101...0101
1	1	0101...0101	1010...1010
2	1	1010...1010	0101...0101
3	2	1010...1010	0101...0101
4	2	1010...1010	0101...0101
5	2	1010...1010	0101...0101
6	2	1010...1010	0101...0101
7	2	1010...1010	0101...0101
8	2	1010...1010	0101...0101
9	1	0101...0101	1010...1010
10	2	0101...0101	1010...1010
11	2	0101...0101	1010...1010
12	2	0101...0101	1010...1010
13	2	0101...0101	1010...1010
14	2	0101...0101	1010...1010
15	2	0101...0101	1010...1010
16	1	1010...1010	0101...0101

Table 3-12. Example of an Enciphering Key Which Produces Only Two Different Internal Keys

(a fact pointed out by D. W. Davies and W. L. Price of the National Physical Laboratory, Teddington, Middlesex, England). In the example, $(C_1, D_1) = (C_{16}, D_{16}) = (1010 \ldots 1010, 0101 \ldots 0101)$, and therefore C_1 and D_1 (see Tables 3-8 and 3-9) are, respectively,

$$k_{49}, k_{41}, \ldots, k_{36}, k_{57} = 10 \ldots 10$$

and

$$k_{55}, k_{47}, \ldots, k_4, k_{63} = 01 \ldots 01$$

which results in (parity-adjusted) external key

$$K_{10,01} = \text{E0 1F E0 1F F1 0E F1 0E}$$

Note that $K_{10,01}$ is also a semiweak key, and $K_{01,10}$ can be used in an encipher operation to recover plaintext enciphered with $K_{10,01}$. In general, it can be shown that for any semiweak key (K) there is another semiweak key ($K' \neq K$) such that

$$E_K E_{K'}(X) = X \text{ and } E_{K'} E_K(X) = X \qquad (3\text{-}10)$$

for any X. Altogether there are 12 semiweak keys. They are represented by the (parity-adjusted) external keys shown in Table 3-13. The six pairs of semiweak keys shown in Table 3-14 satisfy Equation 3-10.

There are other keys which exhibit the property of having some identical internal keys. For example, this occurs when

1. Register C or D contains bit pattern 0011 . . . 0011, 0110 . . . 0110, 1001 . . . 1001, or 1100 . . . 1100.
2. The other register (D or C) contains bit pattern 0000 . . . 0000, 0011 . . . 0011, 0101 . . . 0101, 0110 . . . 0110, 1001 . . . 1001, 1010 . . . 1010, 1100 . . . 1100, or 1111 . . . 1111.

Again, by way of illustration, let register C_0 contain 1100 . . . 1100 and register D_0 contain 0011 . . . 0011. The bit patterns produced in registers C_i

$K_{00,01}$	=	E0	FE	E0	FE	F1	FE	F1	FE
$K_{00,10}$	=	FE	E0	FE	E0	FE	F1	FE	F1
$K_{01,00}$	=	1F	FE	1F	FE	0E	FE	0E	FE
$K_{01,01}$	=	01	FE	01	FE	01	FE	01	FE
$K_{01,10}$	=	1F	E0	1F	E0	0E	F1	0E	F1
$K_{01,11}$	=	01	E0	01	E0	01	F1	01	F1
$K_{10,00}$	=	FE	1F	FE	1F	FE	0E	FE	0E
$K_{10,01}$	=	E0	1F	E0	1F	F1	0E	F1	0E
$K_{10,10}$	=	FE	01	FE	01	FE	01	FE	01
$K_{10,11}$	=	E0	01	E0	01	F1	01	F1	01
$K_{11,01}$	=	01	1F	01	1F	01	0E	01	0E
$K_{11,10}$	=	1F	01	1F	01	0E	01	0E	01

Table 3-13. List of Semiweak Keys Represented as (Parity-Adjusted) External Keys

$(K_{00,01}, K_{00,10})$
$(K_{01,00}, K_{10,00})$
$(K_{01,01}, K_{10,10})$
$(K_{01,10}, K_{10,01})$
$(K_{01,11}, K_{10,11})$
$(K_{11,01}, K_{11,10})$

Table 3-14. Pairs of Semiweak Keys (K, K′) that Satisfy the Relation $E_K E_{K'}(X) = E_{K'} E_K(X) = X$ for all X

Index i	Number of Left Shifts	Content of Register C_i (28 bits)	Content of Register D_i (28 bits)
0		1100...1100	0011...0011
1	1	1001...1001	0110...0110
2	1	0011...0011	1100...1100
3	2	1100...1100	0011...0011
4	2	0011...0011	1100...1100
5	2	1100...1100	0011...0011
6	2	0011...0011	1100...1100
7	2	1100...1100	0011...0011
8	2	0011...0011	1100...1100
9	1	0110...0110	1001...1001
10	2	1001...1001	0110...0110
11	2	0110...0110	1001...1001
12	2	1001...1001	0110...0110
13	2	0110...0110	1001...1001
14	2	1001...1001	0110...0110
15	2	0110...0110	1001...1001
16	1	1100...1100	0011...0011

Table 3-15. Example of an Enciphering Key which Produces Only Four Different Internal Keys

and D_i, from which the internal keys are derived, are shown in Table 3-15. Each of the bit patterns–(1100, 0011), (1001, 0110), (0011, 1100), and (0110, 1001)–occurs four times in the registers (C_i, D_i), i = 1, 2, . . . , 16. Moreover, the pattern of recurrence is such that the same internal key never occurs twice in succession, although it will occur alternately. 48 (parity-adjusted) external keys whose internal keys recur in the manner described are listed in Table 3-16.

Finally, the reader should realize that the described set of keys (weak, semiweak, etc.) pose no threat to the algorithm's security. This is because the number of such keys is small in comparison to the total set of 72,057,594,037,927,936 possible different keys. And, provided that keys are randomly selected, the likelihood of selecting such a key in the first place is therefore very small. However, these keys could easily be avoided during key generation (e.g., if they were intended to be installed in a system for relatively long periods of time).

Details of the DES Algorithm

The basic scheme for encipherment used by DES is shown in Figure 3-4. The initial permutation (IP) and the enciphering function (g) are discussed in more detail at this point.

Indices (i,j)		External Key $K_{i,j}$ (Parity-Adjusted)	Indices (i,j)		External Key $K_{i,j}$ (Parity-Adjusted)
0000	0011	1F 1F 01 01 0E 0E 01 01	1001	0000	E0 01 01 E0 F1 01 01 F1
	0110	01 1F 1F 01 01 0E 0E 01		0011	FE 1F 01 E0 FE 0E 01 F1
	1001	1F 01 01 1F 0E 01 01 0E		0101	FE 01 1F E0 FE 01 0E F1
	1100	01 01 1F 1F 01 01 0E 0E		0110	E0 1F 1F E0 F1 0E 0E F1
				1001	FE 01 01 FE FE 01 01 FE
0011	0000	E0 E0 01 01 F1 F1 01 01		1010	E0 1F 01 FE F1 0E 01 FE
	0011	FE FE 01 01 FE FE 01 01		1100	E0 01 1F FE F1 01 0E FE
	0101	FE E0 1F 01 FE F1 0E 01		1111	FE 1F 1F FE FE 0E 0E FE
	0110	E0 FE 1F 01 F1 FE 0E 01			
	1001	FE E0 01 1F FE F1 01 0E	1010	0011	1F FE 01 E0 0E FE 01 F1
	1010	E0 FE 01 1F F1 FE 01 0E		0110	01 FE 1F E0 01 FE 0E F1
	1100	E0 E0 1F 1F F1 F1 0E 0E		1001	1F E0 01 FE 0E F1 01 FE
	1111	FE FE 1F 1F FE FE 0E 0E		1100	01 E0 1F FE 01 F1 0E FE
0101	0011	FE 1F E0 01 FE 0E F1 01	1100	0000	01 01 E0 E0 01 01 F1 F1
	0110	E0 1F FE 01 F1 0E FE 01		0011	1F 1F E0 E0 0E 0E F1 F1
	1001	FE 01 E0 1F FE 01 F1 0E		0101	1F 01 FE E0 0E 01 FE F1
	1100	E0 01 FE 1F F1 01 FE 0E		0110	01 1F FE E0 01 0E FE F1
				1001	1F 01 E0 FE 0E 01 F1 FE
0110	0000	01 E0 E0 01 01 F1 F1 01		1010	01 1F E0 FE 01 0E F1 FE
	0011	1F FE E0 01 0E FE F1 01		1100	01 01 FE FE 01 01 FE FE
	0101	1F E0 FE 01 0E F1 FE 01		1111	1F 1F FE FE 0E 0E FE FE
	0110	01 FE FE 01 01 FE FE 01			
	1001	1F E0 E0 1F 0E F1 F1 0E	1111	0011	FE FE E0 E0 FE FE F1 F1
	1010	01 FE E0 1F 01 FE F1 0E		0110	E0 FE FE E0 F1 FE FE F1
	1100	01 E0 FE 1F 01 F1 FE 0E		1001	FE E0 E0 FE FE F1 F1 FE
	1111	1F FE FE 1F 0E FE FE 0E		1100	E0 E0 FE FE F1 F1 FE FE

Table 3-16. Partial List of (Parity-Adjusted) External Keys that Produces Four Equally Recurring Internal Keys

The 64-bit input block of data to be enciphered,

$$X = x_1, x_2, \ldots, x_{64}$$

is first subjected to an initial permutation IP. This results in

$$\begin{aligned} L(0) = {} & x_{58}, x_{50}, x_{42}, x_{34}, x_{26}, x_{18}, x_{10}, x_2, \\ & x_{60}, x_{52}, x_{44}, x_{36}, x_{28}, x_{20}, x_{12}, x_4, \\ & x_{62}, x_{54}, x_{46}, x_{38}, x_{30}, x_{22}, x_{14}, x_6, \\ & x_{64}, x_{56}, x_{48}, x_{40}, x_{32}, x_{24}, x_{16}, x_8 \end{aligned} \tag{3-11a}$$

$$\begin{aligned} R(0) = {} & x_{57}, x_{49}, x_{41}, x_{33}, x_{25}, x_{17}, x_9, x_1, \\ & x_{59}, x_{51}, x_{43}, x_{35}, x_{27}, x_{19}, x_{11}, x_3, \\ & x_{61}, x_{53}, x_{45}, x_{37}, x_{29}, x_{21}, x_{13}, x_5, \\ & x_{63}, x_{55}, x_{47}, x_{39}, x_{31}, x_{23}, x_{15}, x_7 \end{aligned} \tag{3-11b}$$

L(0) and R(0) can be used in conjunction with the derived key vectors K(1) through K(16) to produce L(16) and R(16), as shown in Figure 3-4. This 64-bit block of pre-output data

$$\begin{aligned} L(16), R(16) = {} & l_1(16), l_2(16), \ldots, l_{32}(16), \\ & r_1(16), r_2(16), \ldots, r_{32}(16) \end{aligned} \tag{3-12}$$

is then subjected to an inverse initial permutation (IP^{-1}), which reverses the effect of the initial permutation, as follows:

$$\begin{aligned} Y &= y_1, y_2, \ldots, y_{64} \\ &= l_8, r_8, l_{16}, r_{16}, l_{24}, r_{24}, l_{32}, r_{32}, \\ &\quad l_7, r_7, l_{15}, r_{15}, l_{23}, r_{23}, l_{31}, r_{31}, \\ &\quad l_6, r_6, l_{14}, r_{14}, l_{22}, r_{22}, l_{30}, r_{30}, \\ &\quad l_5, r_5, l_{13}, r_{13}, l_{21}, r_{21}, l_{29}, r_{29}, \\ &\quad l_4, r_4, l_{12}, r_{12}, l_{20}, r_{20}, l_{28}, r_{28}, \\ &\quad l_3, r_3, l_{11}, r_{11}, l_{19}, r_{19}, l_{27}, r_{27}, \\ &\quad l_2, r_2, l_{10}, r_{10}, l_{18}, r_{18}, l_{26}, r_{26}, \\ &\quad l_1, r_1, l_9, r_9, l_{17}, r_{17}, l_{25}, r_{25} \end{aligned} \tag{3-13}$$

where the indices, which refer to a particular round, have been dropped for convenience of representation. The reader should observe that the inverse initial permutation described above is different from that described on page 9 of [5]. This is because the pre-output in Figure 3-4 is defined as L(16), R(16) whereas the pre-output on page 8 of [5] is defined as R(16), L(16).

It remains now only to discuss the enciphering function (g). The kernel of this operation (i.e., a one-round operation) is shown in Figure 3-12. First, the right half of the input to round i, denoted by

$$R(i-1) = r_1(i-1), r_2(i-1), \ldots, r_{32}(i-1)$$

is expanded from 32 bits to 48 bits, denoted by $E(R(i-1))$, using the E Bit-Selection Table (page 11 of [5]). The result is

$$\begin{aligned} E(R(i-1)) \quad & r_{32}, r_1, r_2, r_3, r_4, r_5, \\ & r_4, r_5, r_6, r_7, r_8, r_9, \\ & r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, \\ & r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, r_{17}, \\ & r_{16}, r_{17}, r_{18}, r_{19}, r_{20}, r_{21}, \\ & r_{20}, r_{21}, r_{22}, r_{23}, r_{24}, r_{25}, \\ & r_{24}, r_{25}, r_{26}, r_{27}, r_{28}, r_{29}, \\ & r_{28}, r_{29}, r_{30}, r_{31}, r_{32}, r_1 \end{aligned} \tag{3-14}$$

where the indices again have been dropped for convenience of representation. The expansion scheme is shown in Figure 3-13. The purpose of using the E expansion is to achieve a dependence of each bit of ciphertext on all plaintext and key bits in as few rounds as possible.

Once $E(R(i-1))$ is generated, it is added modulo 2 to K(i). This results in a 48-bit vector A.

$$\begin{aligned} A &= E(R(i-1)) \oplus K(i) \\ &= a_1, a_2, \ldots, a_{48} \end{aligned} \tag{3-15}$$

(Although A is different for each round, and hence should be indexed by i, to avoid confusion this is not done here. In Figure 3-12, the elements of vector A are used as arguments in the substitution operations (S-boxes) S_1 through S_8 (pages 15 and 16 of [5]). Each S-box is described as a matrix of four rows (labeled 00, 01, 10, 11) and 16 columns (labeled 0000, 0001, . . . , 1111).

Each S-box can now be represented by four substitution functions, S_i^{00}, S_i^{01}, S_i^{10}, S_i^{11} for S-box S_i, where the superscript identifies the row of the matrix, each mapping four input bits (which determine the column of the matrix) to four output bits given by the element of the matrix. The first and last of the six entries to DES substitution box S_i in Figure 3-12 determine which of the four substitution functions in S_i are selected. Because these bits are derived from input bits as well as key bits, the selection of substitution functions depends not only on the key but also on the input data. Thus the E expansion, in effect, introduces an *autoclave* or self-keying feature. Because i ranges from 1 to 8, 32 functions are obtained with the eight S-boxes.

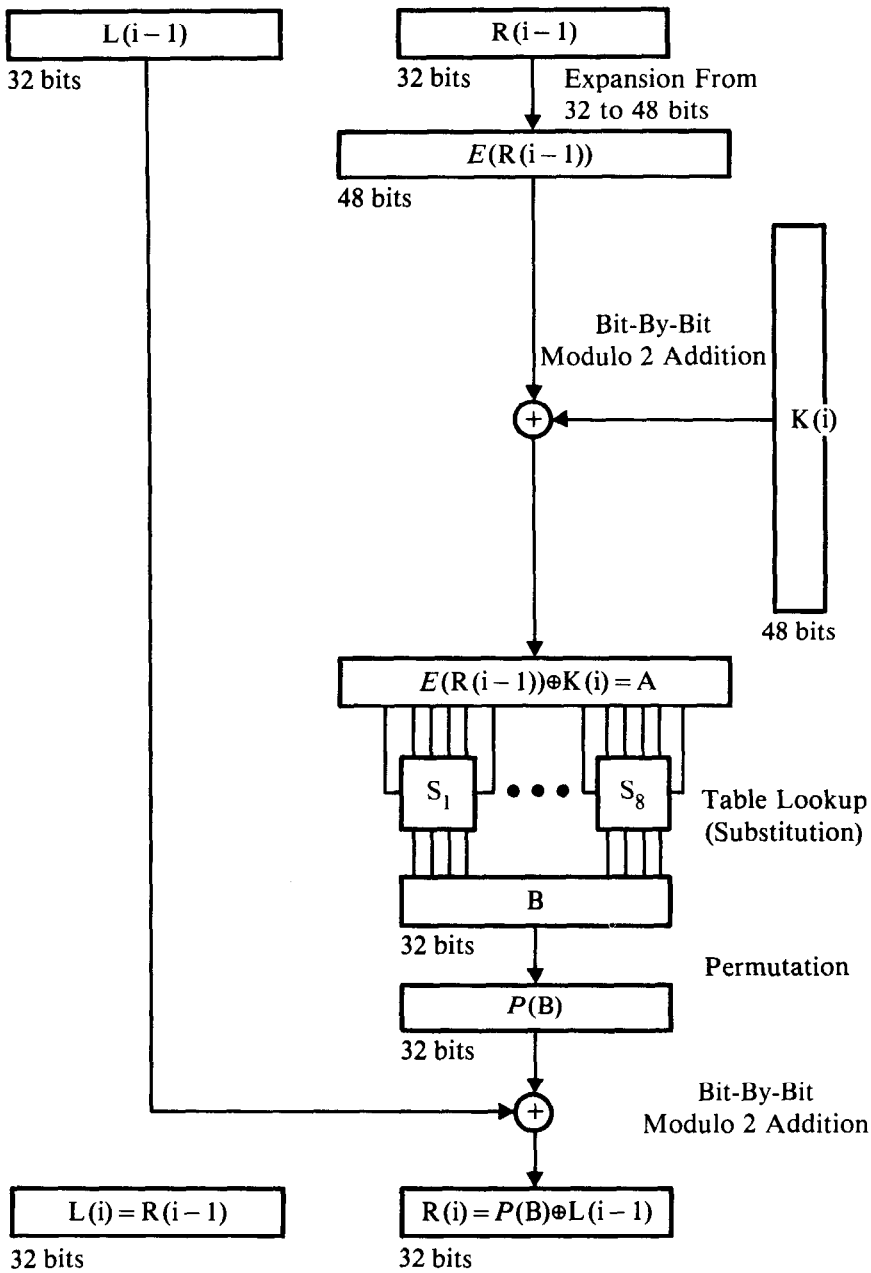


Figure 3-12. Details of Enciphering Function (g)

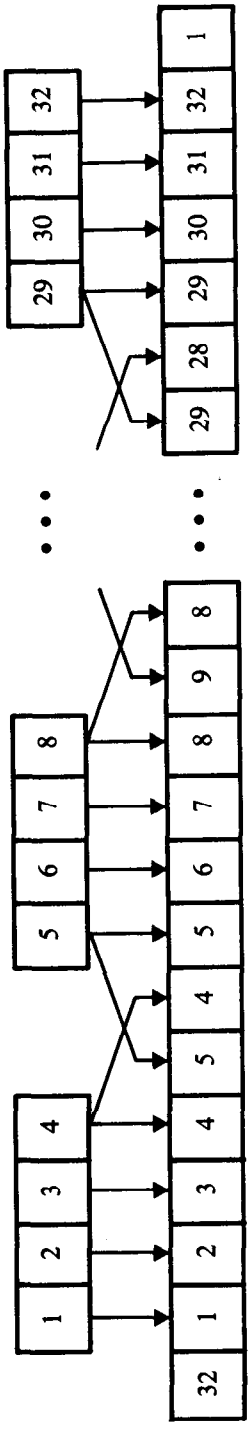


Figure 3-13. Expansion *(E)* of Right Half of Input to Each Round

With a_1 through a_6 being the inputs to the first S-box,

$$S_1^{a_1, a_6}(a_2, a_3, a_4, a_5)$$

represents the first four bits of B (i.e., bits b_1 through b_4). The last four bits, from S_8, (i.e., b_{29} through b_{32}) are given by

$$S_8^{a_{43}, a_{48}}(a_{44}, a_{45}, a_{46}, a_{47})$$

Knowledge of the 48 bits represented by vector A therefore allows one to determine the 32 bits represented by vector B. By permuting the elements of B (page 12 of [5]), one obtains P(B), where P stands for permutation.

$$\begin{aligned} P(B) = {} & b_{16}, \ b_7, b_{20}, b_{21}, b_{29}, b_{12}, b_{28}, b_{17}, \\ & b_1, b_{15}, b_{23}, b_{26}, \ b_5, b_{18}, b_{31}, b_{10}, \\ & b_2, \ b_8, b_{24}, b_{14}, b_{32}, b_{27}, \ b_3, \ b_9, \\ & b_{19}, b_{23}, b_{30}, \ b_6, b_{22}, b_{11}, \ b_4, b_{25} \end{aligned} \tag{3-16}$$

The output of round i thus becomes

$$L(i), R(i) = R(i-1), P(B) \oplus L(i-1)$$

By recognizing that P(B) is a function g of K(i) and R(i − 1), the following relationship can be established:

$$L(i) = R(i-1) \tag{3-17}$$

$$R(i) = L(i-1) \oplus g[K(i), R(i-1)] \tag{3-18}$$

Summary of the DES Procedure

Initialization:

1. Specify the 64-bit, externally supplied, cryptographic key, k_1, k_2, . . . , k_{64} (composed of 56 key bits and 8 nonkey bits).
2. From k_1, k_2, . . . , k_{64}, construct 16 key vectors, K(1) through K(16), of 48 bits each, which are used in rounds 1 through 16, respectively. (K(1) through K(16) are expressed in terms of the external key bits via Tables 3-10 and 3-11.)
3. Specify the 64-bit, externally supplied, input (plaintext), x_1, x_2, . . . , x_{64}.
4. From x_1, x_2, . . . , x_{64}, construct the 32-bit vectors L(0) and R(0), as shown in Equations 3-11a and 3-11b.
5. Set iteration counter to i = 1.

Iteration i:

6. Derive $E(R(i-1))$ from $R(i-1)$ using the expansion function E given by Equation 3-14.
7. Use K(i) if encipherment is being performed, otherwise use $K(17-i)$ for decipherment.
8. Add (modulo 2) the results of steps 6 and 7, and define the result to be the 48-bit vector $A = a_1, a_2, \ldots, a_{48}$.
9. Form $S_1^{a_1,a_6}(a_2, a_3, a_4, a_5)$ and define the 4-bit result as b_1, b_2, b_3, b_4. Repeat this process for

$$S_2^{a_7,a_{12}}(a_8, a_9, a_{10}, a_{11}) \text{ through } S_8^{a_{43},a_{48}}(a_{44}, a_{45}, a_{46}, a_{47})$$

defining the result as $b_5, b_6, \ldots, b_{32}$. The result of this step is the 32-bit vector B.
10. Derive $P(B)$ by permuting B according to the permutation function (P) given by Equation 3-16.
11. Add (modulo 2) $P(B)$ to $L(i-1)$ and define the result to be R(i).
12. Define $L(i) = R(i-1)$.
13. Increment the iteration counter i by 1.
14. If iteration counter is 16 or less, then repeat steps 6 through 14, otherwise derive the output from L(16), R(16), as shown in Equations 3-12 and 3-13.

Numerical Example

In the numerical example presented here, the value L(1), R(1) is derived for a one-round encipherment. It is assumed that

$$X = K = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ A\ B\ C\ D\ E\ F$$

in hexadecimal notation, or

$$X = K = 0000\ 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111$$
$$1000\ 1001\ 1010\ 1011\ 1100\ 1101\ 1110\ 1111$$

in binary notation. From Tables 3-10 and 3-11, it follows that

$$K(1) = 0000\ 1011\ 0000\ 0010\ 0110\ 0111$$
$$1001\ 1011\ 0100\ 1001\ 1010\ 0101 \quad \text{(binary)}$$
$$= 0\ B\ 0\ 2\ 6\ 7\ 9\ B\ 4\ 9\ A\ 5 \quad \text{(hexadecimal)}$$

From Equations 3-11a and 3-11b, it follows that

$$\begin{aligned} L(0) &= 1100\ 1100\ 0000\ 0000\ 1100\ 1100\ 1111\ 1111 \\ &= C\quad C\quad 0\quad 0\quad C\quad C\quad F\quad F \end{aligned}$$

and

$$\begin{aligned} R(0) &= 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010 \\ &= F\quad 0\quad A\quad A\quad F\quad 0\quad A\quad A \end{aligned}$$

Expanding R(0) with the aid of Equation 3-14 yields

$$\begin{aligned} E(R(0)) = {} & 011110\ 100001\ 010101\ 010101 \\ & 011110\ 100001\ 010101\ 010101 \end{aligned}$$

Modulo 2 addition of this with K(1) yields

$$\begin{aligned} A &= E(R(0)) \oplus K(1) \\ &= 011100\ 010001\ 011100\ 110010 \\ &\quad\ 111000\ 010101\ 110011\ 110000 \end{aligned}$$

Grouping these 48 bits into sets of 6 bits allows the convenient evaluation of the substitution operation (page 15 of [5]).

$S_1^{00}(1110) = S_1^0(14) = 0$ (base 10) $= 0000$ (base 2)
$S_2^{01}(1000) = S_2^1(8) = 12 = 1100$
$S_3^{00}(1110) = S_3^0(14) = 2 = 0010$
$S_4^{10}(1001) = S_4^2(9) = 1 = 0001$
$S_5^{10}(1100) = S_5^2(12) = 6 = 0110$
$S_6^{01}(1010) = S_6^1(10) = 13 = 1101$
$S_7^{11}(1001) = S_7^2(9) = 5 = 0101$
$S_8^{10}(1000) = S_8^2(8) = 0 = 0000$

Concatenating all of these results yields

$$\begin{aligned} B &= 0000\ 1100\ 0010\ 0001\ 0110\ 1101\ 0101\ 0000 \\ &= 0\quad C\quad 2\quad 1\quad 6\quad D\quad 5\quad 0 \end{aligned}$$

Applying the permutation P according to Equation 3-16, the result

$$\begin{aligned} P(B) &= 1001\ 0010\ 0001\ 1100\ 0010\ 0000\ 1001\ 1100 \\ &= 9\quad 2\quad 1\quad C\quad 2\quad 0\quad 9\quad C \end{aligned}$$

is obtained. Modulo 2 addition of $P(B)$ with L(0) yields

$$\begin{aligned} R(1) &= 0101\ 1110\ 0001\ 1100\ 1110\ 1100\ 0110\ 0011 \\ &= 5\quad E\quad 1\quad C\quad E\quad C\quad 6\quad 3 \end{aligned}$$

and hence the right half output after round one is obtained. Furthermore,

$$L(1) = R(0) = F\ 0\ A\ A\ F\ 0\ A\ A$$

which completes the one-round sample computation.

Some Remarks about the DES Design

Methods of analysis (attacks), including heuristic approaches, were gathered during an initial study phase. These techniques were then used to obtain a set of design principles, or criteria, that govern the design of the algorithm. By requiring that the candidate algorithm meet all design criteria, the previously defined methods of attack were rendered ineffective.

The initial study phase for DES extended over a 5-year period. The resultant design criteria were then used to design DES's permutation (P-box), substitutions (S-boxes), and key schedules (see Description of the Data Encryption Standard). For example, one design criterion for DES was that the permutation schedule must ensure that each output (ciphertext) bit is a function of all input (plaintext and key) bits after a minimum number of rounds. (See also Analysis of Intersymbol Dependencies for the Data Encryption Standard.)

Since it is common for statisticians to employ randomization techniques in the design of experiments (procedures for collecting, analyzing, and interpreting data), mathematicians, or people with strong mathematical backgrounds, frequently suggest that parameters such as permutation, substitution, and key schedule should be randomly chosen. This indeed was the first thought of the designers of DES. However, realization early in the development process that random parameter selection introduced weaknesses into the algorithm led to abandonment of these random approaches.

In the actual design process, parameters were randomly generated and then tested against the design criteria. A significant portion of the random designs were rejected in this process. For example, S-box functions were selected on the basis of strength and ease of implementation (see Implementation Considerations for the S-Box Design). Thus it should not be surprising that the final solutions contain some structural properties different from those expected to result from use of purely random selection. However, let it again be stated that the results of the DES design effort showed that carefully selected S-box functions will produce a much stronger algorithm than one based on random designs.

The question of S-box structure is an issue that was raised as part of an independent analysis of DES [17], and was also dealt with as part of the

second ICST workshop to investigate the complexity of DES [14]. The members of the ICST workshop concluded that these structures offer no known shortcut solutions.

Implementation Considerations for the S-Box Design

An interesting result, which appears to relate cryptographic strength to the number of logic circuits describing an S-box, was encountered in the design of DES's S-box functions. The minimum number of logic circuits needed to implement the final (nonrandom) S-box design was significantly greater than that required for a preliminary (nearly random) design. This result suggests that the number of logic circuits can be used as an indicator or heuristic to reject weak S-box functions. For example, if an S-box function is selected using a different set of design criteria and the minimum number of logic circuits falls below an established threshold, then the cryptographic strength of the S-box functions is highly suspect.

The measurements upon which this heuristic is based are given below. However, before discussing the measurements, a few preliminaries are necessary. An S-box is a function that maps 6 input bits $(x_1, x_2, \ldots, x_6)$ into 4 output bits $(y_1, y_2, \ldots, y_4)$. Moreover, each output bit (y_i) can be represented as a boolean expression of the 6 input bits, that is, y_i can be represented as one or more *minterms* that are combined using a logical OR operation [18]. In the example, each of the 6 input bits is either 0 or 1, and so there are $2^6 = 64$ minterms, as shown:

1. $x_1 \cdot x_2 \cdot \ldots \cdot x_6$
2. $\overline{x}_1 \cdot x_2 \cdot \ldots \cdot x_6$
3. $x_1 \cdot \overline{x}_2 \cdot \ldots \cdot x_6$
4. $\overline{x}_1 \cdot \overline{x}_2 \cdot \ldots \cdot x_6$

   $\vdots$

64. $\overline{x}_1 \cdot \overline{x}_2 \cdot \ldots \cdot \overline{x}_6$

where "$\cdot$" denotes here the logical AND operation and $\overline{x}$ denotes the complement of x.

To see how to determine the boolean expression for a given S-box function, consider the following example (toy system) in which an S-box has three inputs (x_1, x_2, and x_3) and two outputs (y_1 and y_2), where

		(x_2, x_1)			
		0	1	2	3
(x_3)	0	1	3	2	0
	1	2	1	0	3

"Toy" S-box Function

The outputs (y_2, y_1) can be expressed in terms of the inputs (x_3, x_2, x_1) as follows:

x_3	x_2	x_1	y_2	y_1
0	0	0	0	1
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

For example, the boolean expression for y_1 is

$$y_1 = \bar{x}_3 \cdot \bar{x}_2 \cdot \bar{x}_1 + \bar{x}_3 \cdot \bar{x}_2 \cdot x_1 + x_3 \cdot \bar{x}_2 \cdot x_1 + x_3 \cdot x_2 \cdot x_1$$

where "+" denotes here the logical OR operation. A minterm is included in the boolean expression if the corresponding output bit (y_1) is equal to 1. For example, note that $y_1 = 1$ when $x_1 = 0$, $x_2 = 0$, and $x_3 = 0$.

A derived boolean expression would normally be reduced to find an equivalent expression that would be better with respect to some measure. In the case of the DES algorithm, a single chip design was desired, and therefore it was important at that time (about 1974) to reduce the number of logic circuits needed to implement the S-box functions. The S-boxes were designed using a multidimensional approach in which the boolean expressions for each output bit were reduced jointly [19].

For an early design of the S-box, with only a few design criteria, the distribution of minterms after reduction (based on a sample of 18 S-boxes) was found to be as shown in Table 3-17. However, as more and more design criteria were added, making the design less and less random, a corresponding increase in the number of required minterms was observed. For example, in the final

No. of Minterms per S-box	No. of S-boxes
40	1
41	1
44	3
45	3
46	4
47	2
48	4

Table 3-17. Distribution of Minterms for a Preliminary Design of the S-box

No. of Minterms per S-box	No. of S-boxes
52	3
53	7
54	9
55	22
56	16
57	20
58	4
59	2

Table 3-18. Distribution of Minterms for Final Design of S-box

S-box design, the distribution of minterms after reduction (based on a sample of 83 S-boxes) was found to be as shown in Table 3-18. Therefore, introducing more stringent design criteria caused the required number of minterms to shift significantly, from about 45 to 55, toward the maximum number of 64.

To make it as easy as possible for an LSI logic designer to implement the design on a single chip, the left tail of the distribution was chosen (52 and 53). (Many cryptographic applications that have been proposed require a single chip design for economic and performance reasons.)

ANALYSIS OF INTERSYMBOL DEPENDENCIES FOR THE DATA ENCRYPTION STANDARD[6]

One property of DES is that each bit of ciphertext is a complex function of all plaintext bits and all key bits. A method is developed later which evaluates how fast this dependence (defined as intersymbol dependence) builds up as a function of repeated mathematical operations called rounds. With the DES algorithm, the minimum number of rounds needed to achieve intersymbol dependence for plaintext as well as key is five.

To analyze the intersymbol dependence, consider the basic design approach shown in Figure 3-14. It is not necessary to take into account the initial permutation IP and final permutation IP^{-1} (Figure 3-4). Some of the relations already developed are repeated in this section to make the analysis easier to understand and independent of other sections.

For strength, DES relies on the complexity of the function g, which incorporates substitution as well as transposition (or permutation), and exercising the function g a number of times (defined as rounds).

After 16 rounds have been employed by DES, the 64 bits of plaintext

[6]© 1978 AFIPS press. The material describing the analysis is reprinted in part from *AFIPS Conference Proceedings*, 1978 National Computer Conference [20].

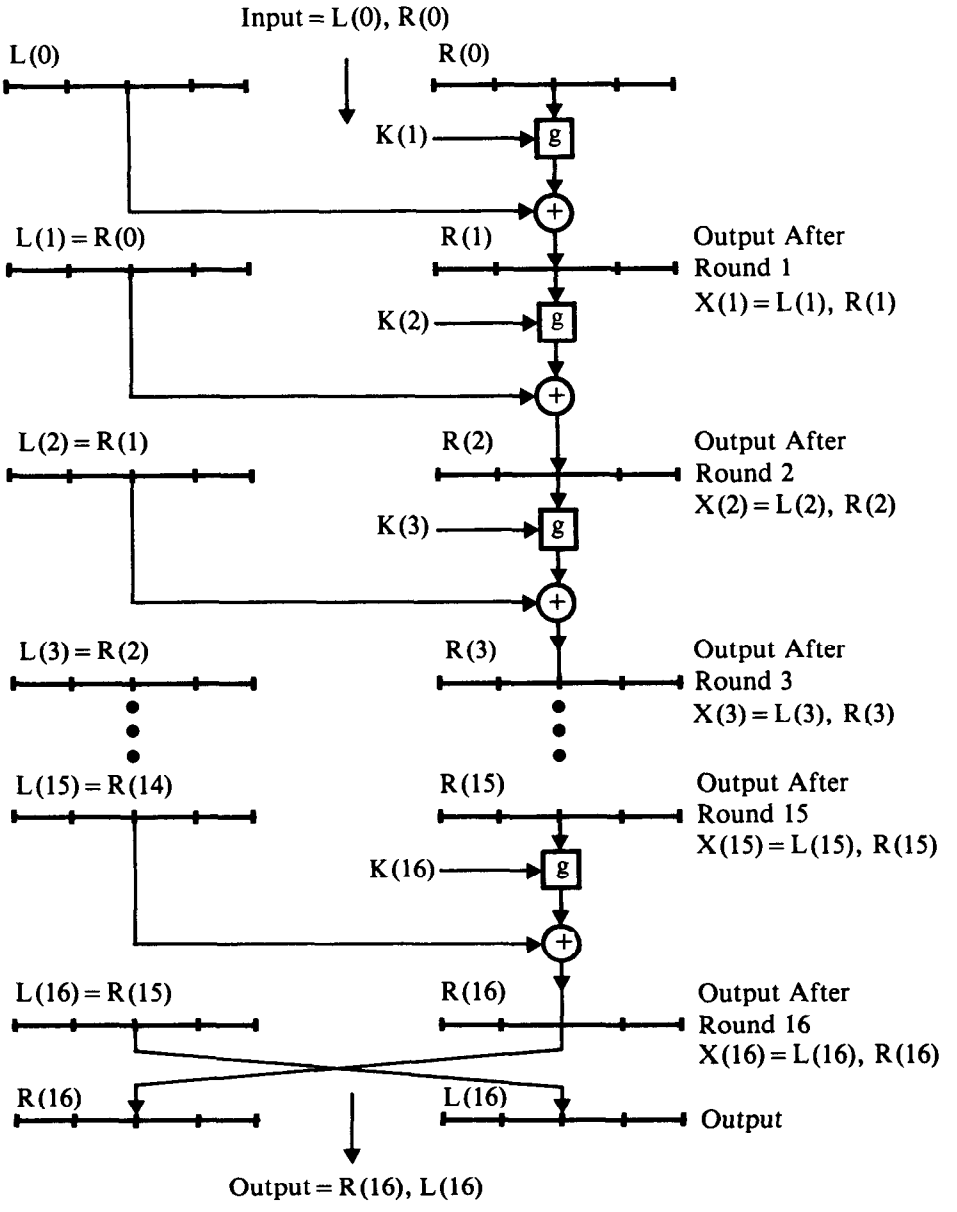


Figure 3-14. Basic Block Cipher Design Used by the Data Encryption Standard

are transformed into 64 bits of ciphertext under the control of a 56-bit key. During each round a subset of the key (defined as K(i) for round i; i = 1, 2, . . . , 16) is used. The input to round i, X(i − 1), is expressed as the concatenation of two 32-bit quantities. L(i − 1) represents the 32 input bits of the left half of the one-round cipher operation (Figure 3-13), and R(i − 1) represets the 32 input bits of the right half:

$$L(i-1) = l_1(i-1), l_2(i-1), \ldots, l_{32}(i-1)$$
$$R(i-1) = r_1(i-1), r_2(i-1), \ldots, r_{32}(i-1)$$

Hence the input to round i is defined as

$$X(i-1) = L(i-1), R(i-1)$$

whereas the output from round i is defined as

$$X(i) = L(i), R(i)$$

Thus each bit of the ciphertext produced by DES is a complex function of all 64 plaintext bits and all 56 key bits.

The following example illustrates the marked change produced in a recovered plaintext when only one bit is changed in the ciphertext or key. Hexadecimal notation is used. If the plaintext 1000000000000001 is enciphered with a (56-bit) key 30000000000000, then the ciphertext 958E6E627A05557B is produced. The original plaintext is recovered if 958E6E627A05557B is deciphered with 30000000000000. However, if the first 9 in the ciphertext is changed to 8 (a 1-bit change) and the ciphertext 858E6E627A05557B is now deciphered with key 30000000000000, the recovered plaintext is 8D4893C2966CC211, not 1000000000000001. On the other hand, if the first 3 in the key is changed to 1 (another 1-bit change) and the ciphertext 958E6E627A05557B is now deciphered with key 10000000000000, the recovered plaintext is 6D4B945376725395. (The same effect is also observed during encipherment.)

The dependence of each ciphertext bit on all bits of the plaintext and key is defined as intersymbol dependence. Many applications can take advantage of this. Two applications, discussed in Chapter 2 and again here, effectively expand the block size of DES by using chaining methods. When block encryption is used, each 64-bit block of data is enciphered separately. In chained block encryption, on the other hand, the encipherment of each block is made dependent on prior information (plaintext, ciphertext, or the like) available when the block is enciphered. Two techniques for block chaining, ciphertext feedback and plaintext-ciphertext feedback, are defined below.

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

$$Y_1 = E_K(X_i \oplus Y_{i-1}) \qquad \text{for } i \geqslant 1$$

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

$$Y_1 = E_K(X_1 \oplus Y_0)$$

and

$$Y_i = E_K(X_i \oplus Y_{i-1} \oplus X_{i-1}) \qquad \text{for } i \geqslant 2.$$

With block chaining, a ciphertext bit in block i depends not only on all plaintext bits in block i, but also on the plaintext bits of block 1 through (i − 1). Thus chained block encryption (or block chaining) can be used to extend the intersymbol dependence between ciphertext and plaintext.

Plaintext-ciphertext feedback has the additional property of error propagation. Corruption of a single bit of ciphertext will cause each subsequent bit of recovered plaintext to be in error with a probability approximately equal to 0.5. Appending a known pattern of bits to the end of the plaintext prior to encryption, and comparing that value to the value recovered, allows the error propagation feature to be used for checking the true content of a message. Chaining techniques are useful for encrypting data, and the block cipher (with no chaining) is useful for key transformation (see Chapters 4 and 5).

In the analysis below, a method is developed which shows how fast the intersymbol dependence builds up as a function of the number of rounds. A basic assumption is made that the substitution functions are *nonaffine*,[7] such that cancellation of dependencies does not occur. Since, in the design of DES, great care was taken to select S-boxes with the nonaffine property, it is assumed in the analysis that the assumption stated above holds for DES.

At each step, the analysis considers whether or not an output bit depends upon an input bit. Although the degree of complexity is not measured, a distinction is made among three kinds of functional relationships. Details of g (the kernel of the DES cryptographic approach), as illustrated in Figure 3-12, must be considered in a discussion of these relationships.

For DES, the right half of the input is expanded from 32 bits to 48 bits (see Figures 3-12 and 3-13, and Equation 3-14), an S-box has six input bits and four output bits. The first and last input of the six entries to the DES substitution box S_i in Figure 3-12 determine which of the four substitution functions in S_i is selected. Because these bits are derived from the 32 input bits (i.e., message bits if the input is interpreted as a message) as well as key bits due to the modulo 2 addition of K(i) and E(R(i − 1)), the selection of substitution functions not only depends on the key but also on the input data. This, in effect, introduces a self-keying feature, or *autoclave.*

In the following two analyses, the dependencies of output X(i) on plaintext X(0) and on the key are treated separately.

Interdependence between Ciphertext and Plaintext

To investigate the functional relationship between the input to round i + 1 (which is equal to the output of round i, i = 1, 2, 3, . . . , 16) and the input to the first round, a matrix is defined. It consists of 64 rows and 64 columns and is referred to as $G_{i,j}$. Its element $a_{l,m}$ in row l and column m shows the

[7] A function is nonaffine if, for an operator □, the condition f(x □ y) ≠ f(x) □ f(y) □ c holds, where c is not dependent on x and y.

type of relationship which exists between the lth bit of X(i) and the mth bit of X(j). In particular, $a_{l,m}$ is blank if a dependency does not exist between $x_l(i)$ and $x_m(j)$. If there is a dependency via message bits only, $a_{l,m}$ is set to x. If the dependency is via autoclave, $a_{l,m}$ is set to —. If message bits as well as autoclave bits influence the output, $a_{l,m}$ is set to *.

In a well designed system, an output bit depends on more and more input bits as the number of rounds increases. These input bits come into play via message dependence (x), autoclave dependence (—), or both (*). The design goal is to have each output bit depend on each input bit after only a few rounds, with autoclave as well as message dependence being achieved.

It is advantageous to partition matrix $G_{i,j}$ into four submatrices of 32 rows and 32 columns each:

$$G_{i,j} = \begin{bmatrix} G_{i,j}^{(L,L)} & G_{i,j}^{(L,R)} \\ G_{i,j}^{(R,L)} & G_{i,j}^{(R,R)} \end{bmatrix}$$

Using the definition of $G_{i,j}$, one can see that the elements of the submatrices express the relationships shown in Table 3-19. Let the relationships between the output and the input of one round ($G_{i,i-1}$) be evaluated first. The dependence of the output bits from the substitution operation (vector B in Figure 3-12) on the input bits R(i — 1) is shown in Figure 3-15. (Note that the dependence of B on R(i — 1) is identical to the dependence of R(i) on R(i — 1) if the permutation is not present.)

The selection of the substitution function (S-function) in the first S-box, S_1, depends on bits 32 and 5 of R(i — 1). This follows from the fact that the first and sixth input bits to S_1 select the S-function, and the first and sixth bits of the expanded version of R(i — 1), which are determined by E(R(i — 1)), are equal to $r_{32}(i-1)$ and $r_5(i-1)$, respectively, according to Equation 3-14 and Figure 3-13. Therefore, all output bits from S_1 (b_1 through b_4) depend on $r_{32}(i-1)$ and $r_5(i-1)$ via autoclave. However, they depend on $r_1(i-1)$ through $r_4(i-1)$ via message. R(i — 1) is considered to be the second 32 bits of input message in this case. Hence the entries in rows 1

Submatrix	Relationship Expressed in Submatrix
$G_{i,j}^{(L,L)}$	L(i) vs. L(j)
$G_{i,j}^{(L,R)}$	L(i) vs. R(j)
$G_{i,j}^{(R,L)}$	R(i) vs. L(j)
$G_{i,j}^{(R,R)}$	R(i) vs. R(j)

Table 3-19. Functional Relationships

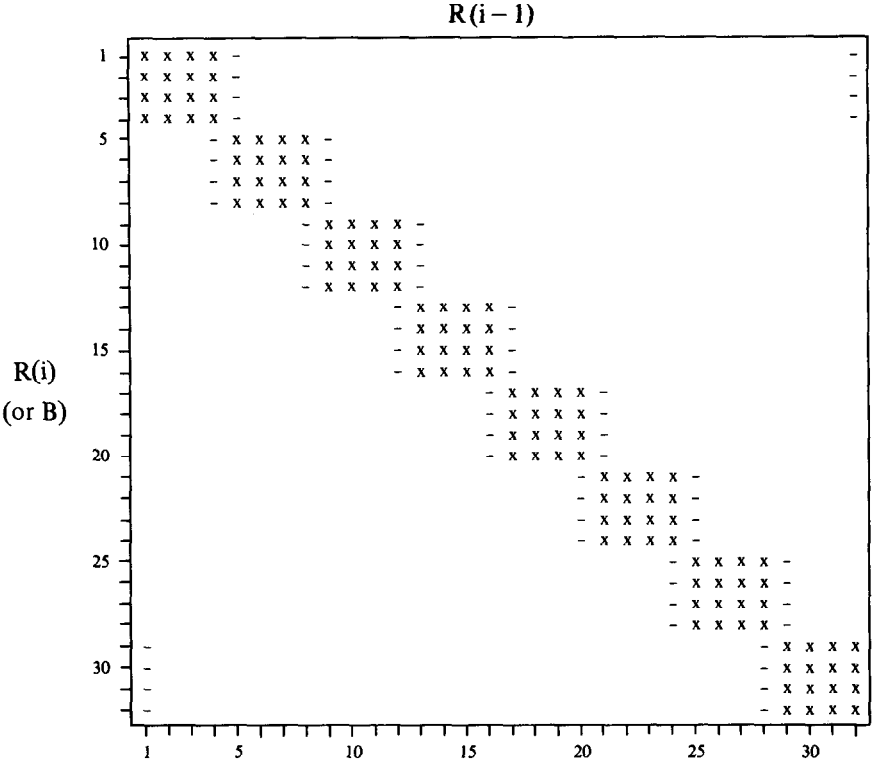


Figure 3-15. Functional Dependence of R(i) on R(i – 1) without Permutation

through 4, columns 32 and 5 in Figure 3-15, are equal to –. The corresponding entries in columns 1 through 4 are equal to x. All other entries of rows 1 through 4 are blank since b_1 through b_4 only depend on six input bits. Repeating this analysis for S_2 through S_8 results in the matrix of Figure 3-15.

Taking permutation into account, the matrix rows have to be rearranged according to the permutation schedule given by $P(B)$ in Equation 3-16. The result is shown in Figure 3-16 and represents the type of relationship that exists between R(i) and R(i – 1); thus, $G_{i,i-1}^{(R,R)}$ is obtained (see Table 3-19).

Since L(i) = R(i – 1) (Equation 3-17), one can see that L(i) does not depend on L(i – 1) but has a linear dependence on R(i – 1). Hence the elements of $G_{i,i-1}^{(L,L)}$ are blank since they express the relationship between L(i) and L(i – 1).

The relationship between L(i) and R(i – 1) is expressed by the elements of $G_{i,i-1}^{(L,R)}$. Due to the linear relationship between L(i) and R(i – 1), the elements located on the diagonal are set to x as shown in Figure 3-17. Equation 3-18 shows the relationship among R(i), L(i – 1), K(i), and R(i – 1). Since the dependence on K(i) is not of interest here, the expression

$$R(i) = L(i-1) \oplus h[R(i-1)] \qquad (3\text{-}19)$$

is used henceforth. The relationship between R(i) and L(i – 1) is also linear.

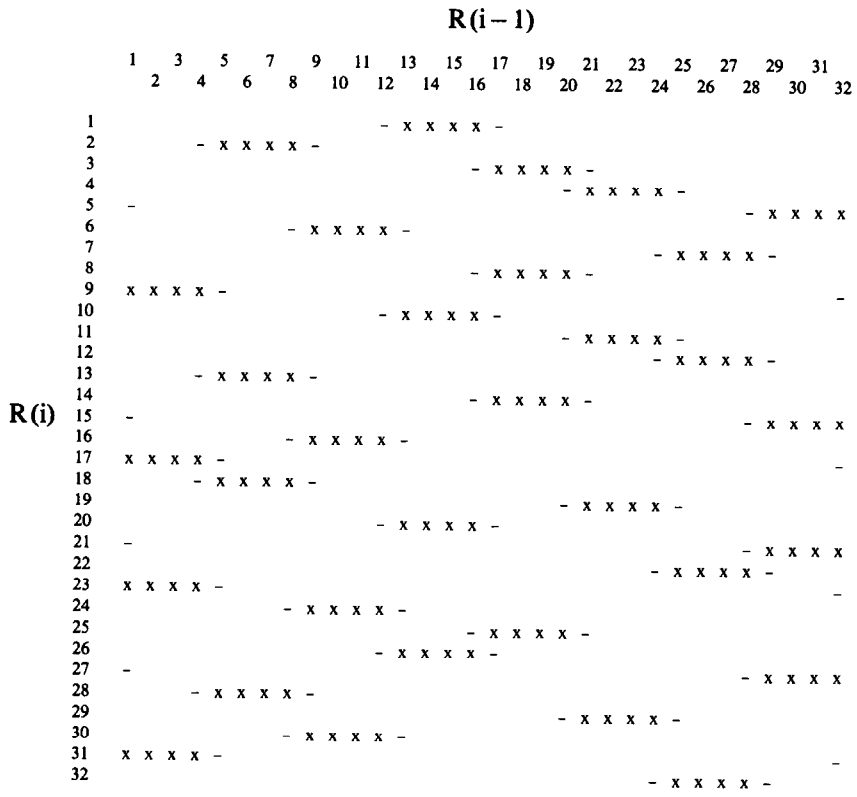


Figure 3-16. Functional Dependence of R(i) on R(i − 1), Matrix $G^{(R,R)}_{i,i-1}$

Hence the diagonal elements of $G^{(R,L)}_{i,i-1}$, which relate R(i) and L(i − 1), are set to x, whereas the remaining elements are blank (Figure 3-17). Since $G^{(L,L)}_{i,i-1}$ through $G^{(R,R)}_{i,i-1}$ have now been evaluated, the matrix $G_{i,i-1}$ is completely defined. It is shown in Figure 3-17.

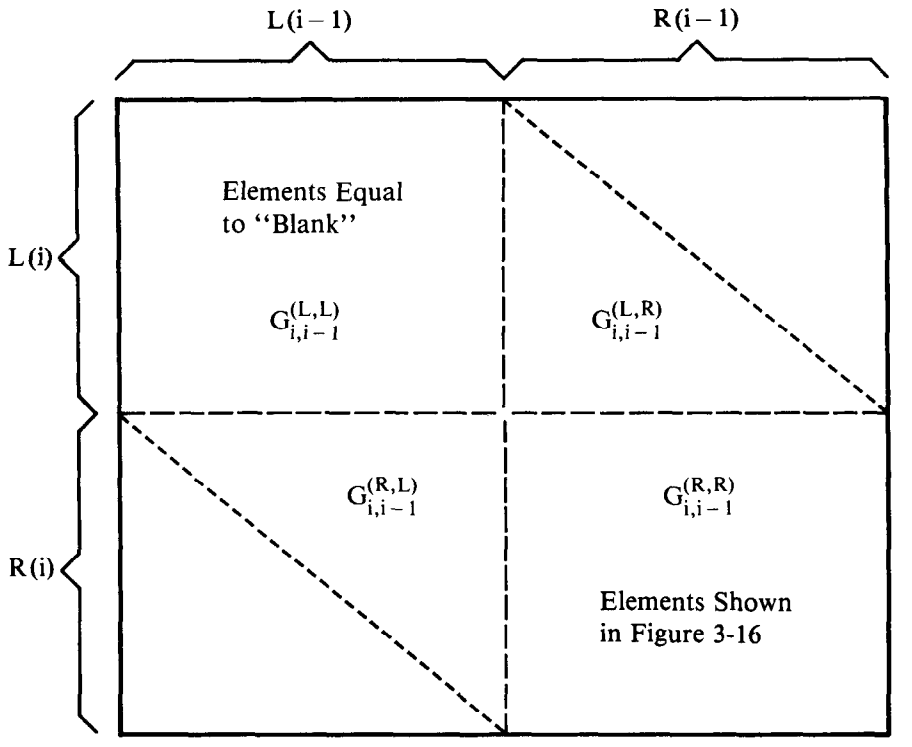
Let $G_{i+1,i-1}$ be expressed next in terms of $G_{i,i-1}$. From Equation 3-17, the relation L(i + 1) = R(i) can be established. Thus the functional dependence of L(i + 1) on X(i − 1) is identical to the functional dependence of R(i) on X(i − 1). Since the dependence of L(i + 1) on X(i − 1) = [L(i − 1), R(i − 1)] is given by $G^{(L,L)}_{i+1,i-1}$, $G^{(L,R)}_{i+1,i-1}$, and the dependence of R(i) on X(i − 1) is given by $G^{(R,L)}_{i,i-1}$, $G^{(R,R)}_{i,i-1}$, according to Table 3-19, it follows that

$$G^{(L,L)}_{i+1,i-1} = G^{(R,L)}_{i,i-1} \tag{3-20}$$

and

$$G^{(L,R)}_{i+1,i-1} = G^{(R,R)}_{i,i-1} \tag{3-21}$$

Before the derivation of $G^{(R,L)}_{i+1,i-1}$ and $G^{(R,R)}_{i+1,i-1}$, a more general case is considered, the evaluation of $G^{(R,L)}_{j+1,i-1}$ and $G^{(R,R)}_{j+1,i-1}$ from $G_{j,i-1}$ ($j \geq i$). (Note that for j = i the special case described above is obtained.)

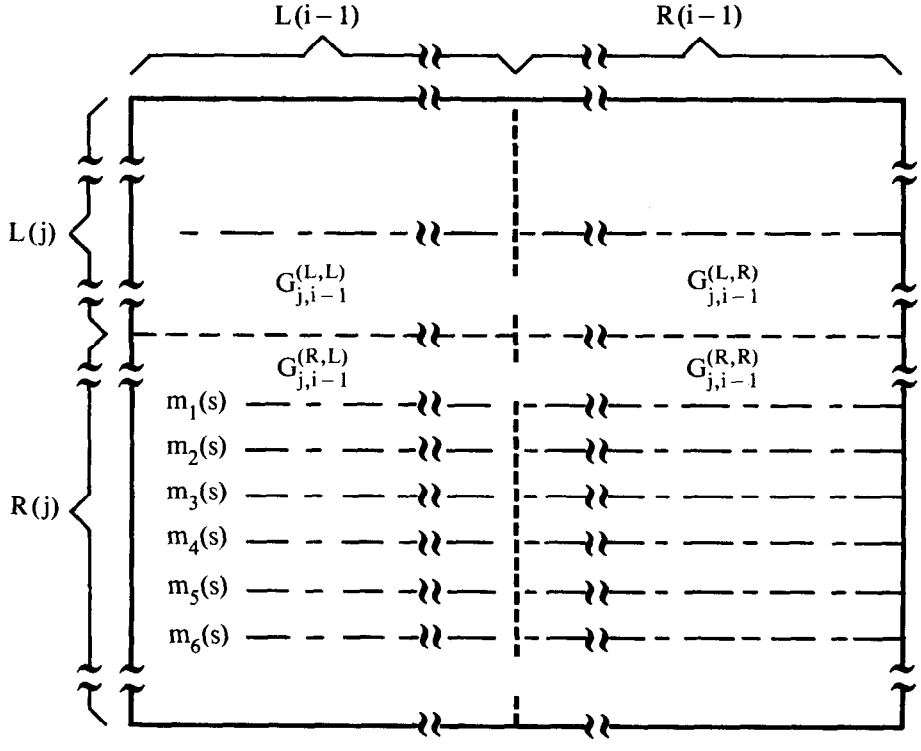


Elements of $G^{(L,R)}_{i,i-1}$ and $G^{(R,L)}_{i,i-1}$ located on the indicated diagonal are equal to "x" whereas the remaining elements are equal to blank.

Figure 3-17. Functional Relationship between $X(i) = L(i)$, $R(i)$ and $X(i-1) = L(i-1)$, $R(i-1)$, Matrix $G_{i,i-1}$

The elements of row s of $G^{(R,L)}_{j+1,i-1}$ give the relationship between bit s of R(j + 1) (i.e., $r_s(j + 1)$) and L(i − 1), whereas the elements of row s of $G^{(R,R)}_{j+1,i-1}$ give the relationship between bit s of R(j + 1) and R(i − 1). To evaluate these elements, one must first obtain the relationship between $r_s(j + 1)$ and X(j) = [L(j), R(j)] by using Figures 3-16 and 3-17. Bit s of R(j + 1) linearly depends on L(j), as shown in Figure 3-17 (i.e., on bit s of L(j) via message dependence (x)). From Figure 3-16, it follows that bit s of R(j + 1) depends on R(j) via —xxxx— (i.e., bit s depends on two bits of R(j) via autoclave and on four bits of R(j) via a message relationship). Let the columns of these entries in $G^{(R,R)}_{i,i-1}$ (equal to $G^{(R,R)}_{j+1,j}$ if i = j +1) be designated as $m_1(s)$ through $m_6(s)$. For example (Figure 3-16), the following is obtained for s = 4: $m_1(s) = 20$, $m_2(s) = 21$, $m_3(s) = 22$, $m_4(s) = 23$, $m_5(s) = 24$, and $m_6(s) = 25$. Since the dependence of X(j) on X(i − 1) is given by $G_{j,i-1}$ and the dependence of bit s of R(j + 1) on X(j) is known, the relationship between R(j + 1) and X(i − 1), given by rows 33 to 64 of matrix $G_{j+1,i-1}$ can be determined by properly combining elements of row s, $m_1(s) + 32$, . . . , $m_6(s) + 32$ of $G_{j,i-1}$. This method is indicated in Figure 3-18.

The rule that should be used to combine the rows of matrix $G_{j,i-1}$ so that the dependence of $x_{s+32}(j+1)$ on $X(i-1)$ can be evaluated still must be decided. Since the main objective of this analysis is to determine how fast the functional dependence between output and input builds up, it would be sufficient simply to indicate if a functional relationship between an output bit and an input bit exists. However, to provide more insight on the influence of autoclave, a rule is derived which highlights autoclave influence. Thus all elements of $G_{j,i-1}$ for rows $m_1(s)+32$ and $m_6(s)+32$ are set to $-$, indicating an autoclave relationship. Note that the elements of row $s+32$ of $G_{j+1,i-1}$ describe the dependence of $x_{s+32}(j+1)$ on $X(i-1)$. Since $x_{s+32}(j+1)$ depends on $x_{m_1(s)+32}(j)$ and $x_{m_6(s)+32}(j)$ via autoclave, the dependence of $x_{m_1(s)+32}(j)$ and $x_{m_6(s)+32}(j)$ on $X(i-1)$ is changed from $-$, x, or * to an autoclave dependence $-$. Adoption of this rule permits the



Note: $X(j+1)=[L(j+1), R(j+1)]$ vs. $X(i-1)$ is obtained by combining the elements of row s, $m_1(s)$ through $m_6(s)$, where $m_1(s)$ through $m_6(s)$ are the columns in which the elements in row s of $G^{(R,R)}_{i,i-1}$ occur. ($1 \leqslant s \leqslant 32$.) $G^{(R,R)}_{i,i-1}$ is defined in Figure 3-16.

Figure 3-18. Evaluation of Functional Dependence of $X(j+1)$ vs. $X(i-1)$ from $X(j)$ vs. $X(i-1)$.

elimination of previous history as far as the autoclave entry to an S-box is concerned, thereby increasing the emphasis on the last autoclave dependence.

In order to take into account previous history for the entries to an S-box not associated with autoclave, the elements of $G_{j,i-1}$ are left unchanged for rows s, $m_2(s) + 32$, $m_3(s) + 32$, $m_4(s) + 32$, and $m_5(s) + 32$.

When the proper rows of $G_{j,i-1}$ are combined, one or more entries can occur in the same column. Only if entries are either all x or all — will the final dependence be correspondingly set. If mixed entries (x and —) occur, the final dependence is set to *. If the entry * occurs at least once, then the final dependence is set to *, regardless of the other entries.

Summary of the Procedure

The elements of row s + 32 of matrix $G_{j+1,i-1}$, for $1 \leqslant s \leqslant 32$, can be generated from matrix $G_{j,i-1}$ by determining $m_1(s)$ through $m_6(s)$ from the given matrix $G_{i,i-1}^{(R,R)}$. They are the columns in which the nonblank elements in row s of $G_{i,i-1}^{(R,R)}$ are located. (Figure 3-16 shows these entries.) Take matrix $G_{j,i-1}$ and select row s, $m_1(s) + 32$ through $m_6(s) + 32$. The entries in row s + 32 of matrix $G_{j+1,i-1}$ are now obtained by combining the elements in the indicated rows using the established rules.

As a corollary, the following can be stated. Elements in row s of $G_{j+1,i-1}^{(R,L)}$ are obtained by combining row s of $G_{j,i-1}^{(L,L)}$ with rows $m_1(s)$ through $m_6(s)$ of $G_{j,i-1}^{(R,L)}$. Elements in row s of $G_{j+1,i-1}^{(R,R)}$ are obtained by combining row s of $G_{j,i-1}^{(L,R)}$ with rows $m_1(s)$ through $m_6(s)$ of $G_{j,i-1}^{(R,R)}$.

Consider now the special case j = i (i.e., let $G_{i+1,i-1}$ be evaluated from $G_{i,i-1}$). The submatrix $G_{i+1,i-1}^{(R,R)}$ can be derived by using the stated rules. The result is shown in Figure 3-19.

For ease of understanding, the detailed steps for obtaining the elements of row s (for s = 4) of $G_{i+1,i-1}^{(R,R)}$ from $G_{i,i-1}$ are shown in Figure 3-20. Note that for s = 4, one obtains (from Figure 3-16) $m_1(s), \ldots, m_6(s)$ equal to 20, 21, 22, 23, 24, 25, as stated above.

$G_{i+1,i-1}^{(R,L)}$ can now be computed. According to the established rule, elements of row s of $G_{i,i-1}^{(L,L)}$ and of rows $m_1(s)$ through $m_6(s)$ of $G_{i,i-1}^{(R,L)}$ must be combined. But the elements of $G_{i,i-1}^{(L,L)}$ are all blank, so that it does not come into play. Elements of row $m_1(s)$ through $m_6(s)$ at $G_{i,i-1}^{(R,L)}$ are equal to x at columns $m_1(s)$ through $m_6(s)$. (See Figure 3-17.) According to the established rule, the entries at columns $m_1(s)$ and $m_6(s)$ are changed to — before all entries are combined to obtain the elements of row s in $G_{i+1,i-1}^{(R,L)}$. These entries are then equal to —xxxx— at columns $m_1(s)$ through $m_6(s)$, that is, identical to the ones in $G_{i,i-1}^{(R,R)}$.

Therefore, it can be concluded that

$$G_{i+1,i-1}^{(R,L)} = G_{i,i-1}^{(R,R)}$$

Combining this result with Equation 3-21 yields the following relationship:

$$G_{i+1,i-1}^{(R,L)} = G_{i+1,i-1}^{(L,R)} = G_{i,i-1}^{(R,R)} \tag{3-22}$$

Equation 3-20 permits the evaluation of $G_{i+1,i-1}$ in terms of $G_{i,i-1}$.

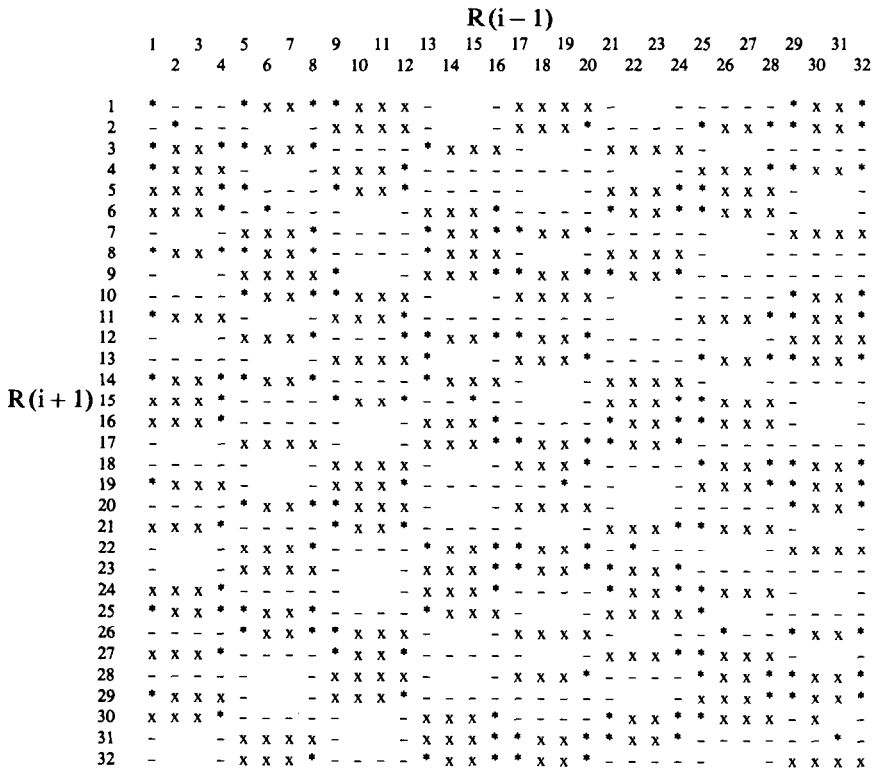


Figure 3-19. Functional Dependence of R(i + 1) on R(i − 1), Matrix $G^{(R,R)}_{i+1,i-1}$

The proof by induction is used next to show that the following general rule (for $j \geqslant i$) holds:

$$G^{(L,L)}_{j+1,i-1} = G^{(R,L)}_{j,i-1} \tag{3-23}$$

$$G^{(L,R)}_{j+1,i-1} = G^{(R,R)}_{j,i-1} \tag{3-24}$$

$$G^{(R,L)}_{j+1,i-1} = G^{(L,R)}_{j+1,i-1} = G^{(R,R)}_{j,i-1} \tag{3-25}$$

The first stage of the proof—the demonstration that these statements are correct for the smallest value of j, j = i—is given above. (See Equations 3-20, 3-21, and 3-22.)

From the relationship (Equation 3-17)

$$L(j+1) = R(j)$$

it can be determined that L(j + 1) depends on X(i − 1) in the same way R(j) depends on X(i − 1). Thus

$$G^{(L,L)}_{j+1,i-1} = G^{(R,L)}_{j,i-1}$$

$$G^{(L,R)}_{j+1,i-1} = G^{(R,R)}_{j,i-1}$$

which proves Equations 3-23 and 3-24.

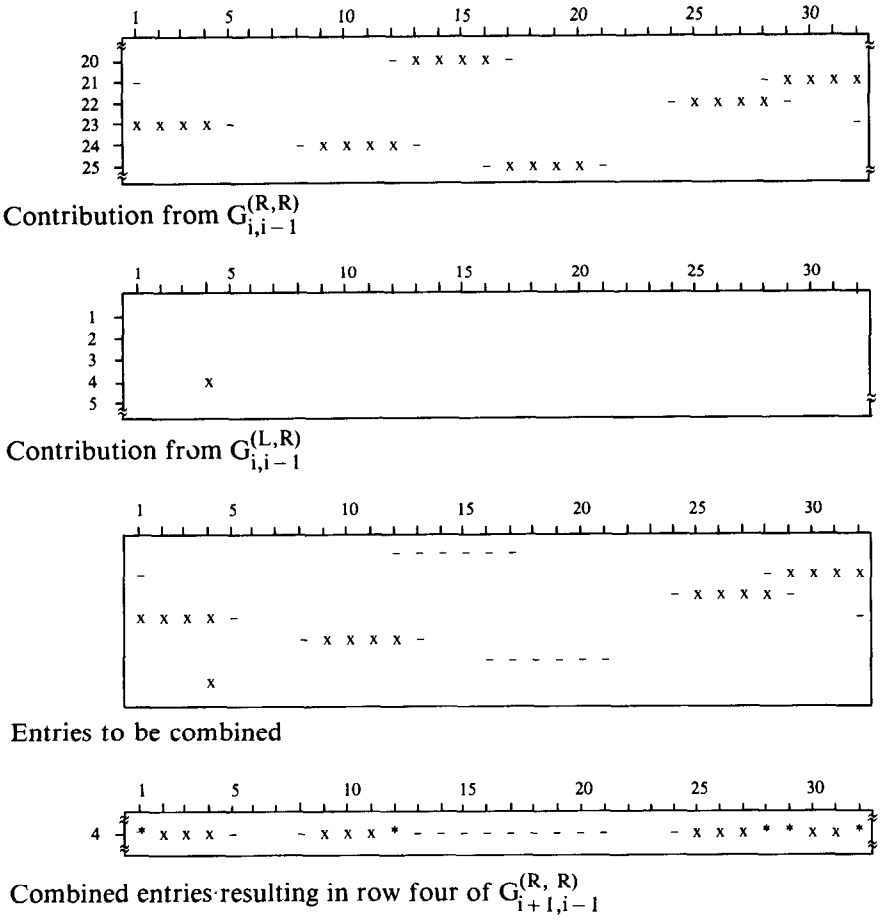


Figure 3-20. Functional Dependence of 4th Bit of R(i + 1) on R(i − 1)

The results stated in the corollary are used to prove Equation 3-25 as follows. $G^{(R,L)}_{j+1,i-1}$ is obtained by combining the elements of $G^{(L,L)}_{j,i-1}$ (row s) and $G^{(R,L)}_{j,i-1}$ (rows $m_1(s)$ through $m_6(s)$). But $G^{(L,L)}_{j,i-1} = G^{(R,L)}_{j-1,i-1}$, $G^{(R,L)}_{j,i-1} = G^{(R,R)}_{j-1,i-1}$, if the stated rule holds. $G^{(L,R)}_{j+1,i-1} = G^{(R,R)}_{j,i-1}$ is obtained by combining the elements of $G^{(L,R)}_{j-1,i-1}$ (row s) and $G^{(R,R)}_{j-1,i-1}$ (rows $m_1(s)$ through $m_6(s)$). But $G^{(L,R)}_{j-1,i-1} = G^{(R,L)}_{j-1,i-1}$ if the stated rule holds. Hence, $G^{(R,L)}_{j+1,i-1}$ is obtained just as $G^{(R,R)}_{j,i-1}$ was and, likewise, $G^{(L,R)}_{j+1,i-1}$ is obtained. Therefore, $G^{(R,L)}_{j+1,i-1} = G^{(L,R)}_{j+1,i-1}$. That completes the proof. (See Figure 3-21.)

Minimum Number of Rounds Required to Achieve Ciphertext/Plaintext Intersymbol Dependence

For each output bit to depend on all plaintext bits after round j, no element of $G_{j,0}$ can be blank. When this condition is satisfied, it follows from the relationships developed between $G_{j,0}$ and $G_{j-1,0}$ and $G_{j-2,0}$ that no ele-

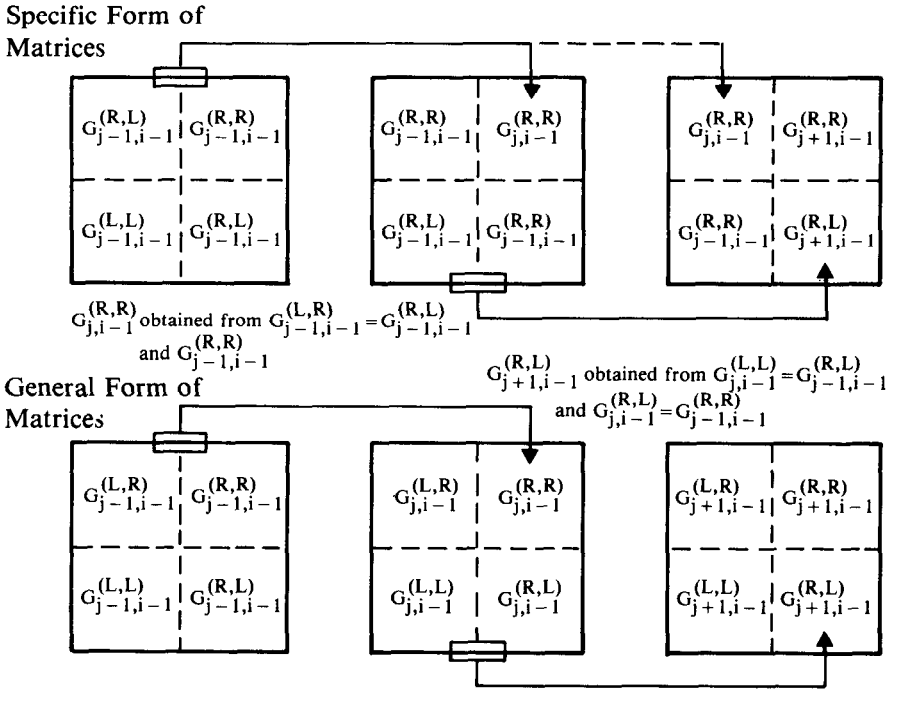


Figure 3-21. Graphical Presentation of Proof that $G_{j+1,i-1}^{(R,L)} = G_{j+1,i-1}^{(L,R)}$

ment of $G_{j-1,0}^{(L,R)} = G_{j-1,0}^{(R,L)}$ can be blank and no element of $G_{j-2,0}^{(R,R)}$ can be blank.

If all row entries of the matrices were independent, an approximate result for the minimum number of rounds could be obtained as shown below. (In actuality, the elements of eight sets of four rows each of $G_{i,i-1}^{(R,R)}$ are highly correlated. Therefore, the increase of intersymbol dependence will be slower than predicted by the approximation.) Since $G_{1,0}^{(R,R)}$ has six entries in each row (Figure 3-16) and $G_{1,0}^{(L,R)}$ has one entry in each row (Figure 3-17), then $G_{2,0}^{(R,R)}$ can have at most $1 + 6 \cdot 6$ or 37 entries. (Note that $G_{2,0}^{(R,R)}$ is obtained from $G_{1,0}^{(L,R)}$ and $G_{1,0}^{(R,R)}$.) There are, however, only 32 columns in $G_{2,0}^{(R,R)}$. Therefore, it is possible that none of the elements of $G_{2,0}^{(R,R)}$ are blank. Thus it is possible to fill all 32^2 entries of $G_{2,0}^{(R,R)}$, $G_{3,0}^{(L,R)} = G_{3,0}^{(R,L)}$, $G_{4,0}^{(R,R)}$ after rounds 2, 3, and 4, respectively. Hence all 64^2 entries of $G_{4,0}$ could be filled after four rounds.

The approximate analysis shows that four is the minimum number of rounds needed to achieve intersymbol dependence between ciphertext and plaintext. The accurate analysis below shows that after four rounds complete interdependence has not been achieved and for the specific design of DES five rounds are required.

From the evaluation of $G_{i+1,i-1}^{(R,R)}$ (Figure 3-19) one can see that there are 28 nonblank elements in each row, except for row 30 which contains 29 elements. It can also be shown that all entries of $G_{i+2,i-1}^{(R,R)}$ are equal to *.

Round	Output/Input Relation				
j	L(j)vs.L(0)	L(j)vs.R(0)	R(j)vs.L(0)	R(j)vs.R(0)	X(j)vs.X(0)
1	0.00	3.13	3.13	18.75	6.25
2	3.13	18.75	18.75	87.60	32.06
3	18.75	87.60	87.60	100.00	73.49
4	87.60	100.00	100.00	100.00	96.90
5	100.00	100.00	100.00	100.00	100.00

Note: Table entries express the degree of intersymbol dependence, i.e., the percentage of nonblank elements in the appropriate relation.

Table 3-20. Ciphertext/Plaintext Intersymbol Dependence

Thus after round 4 each output bit of L(4), except bit 30, depends on 28 bits of L(0) and all 32 bits of R(0). Bit 30 of L(4) depends on 29 bits of L(0) and all 32 bits of R(0). Each bit of R(4), on the other hand, individually depends on the 32 plaintext bits of L(0) and R(0).

Let the degree of intersymbol dependence be measured by a factor ξ, which is obtained by evaluating the percentage of nonblank elements. Respectively, for $G_{1,0}^{(L,L)}$ through $G_{1,0}^{(R,R)}$ (Figures 3-16 and 3-17), ξ is 0%; $100 \cdot 32/32^2 = 3.125\%$; 3.125%; and $100 \cdot 32 \cdot 6/32^2 = 18.75\%$. With the total $G_{1,0}$, ξ is equal to $(0 + 3.125 + 3.125 + 18.75)/4$ or 6.25%. Using either the results above or Figure 3-19, $\xi = 100[32^2 - ((31 \cdot 4) + 3)]/32^2 = 87.60\%$ for $G_{2,0}^{(R,R)}$. Hence according to the rules for obtaining $G_{3,0}$ from $G_{1,0}$, the values of ξ associated with $G_{2,0}^{(L,L)}$ through $G_{2,0}^{(R,R)}$ are 3.13, 18.75, 18.75, and 87.60%, respectively, whereas ξ for $G_{2,0}$ is equal to $\frac{1}{4}$ (3.125 + 18.75 + 18.75 + 87.60) or 32.06%. Because none of the elements of $G_{3,0}^{(R,R)}$ are blank (they are actually all *), ξ, as associated with $G_{3,0}^{(R,R)}$, is equal to 100%. The results are summarized in Table 3-20.

Interdependence Between Ciphertext and Key

For the investigation of the functional relationship between the input to the (i + 1)th round (equal to the output of the (i)th round, i = 1, 2, . . . 16) and the key, a matrix F_i of 64 rows and 56 columns is defined. The elements of the matrix, $a_{l,m}$, for row l and column m show the type of relationship which exists between the lth bit of X(i) and the mth bit of U where

$$U = u_1, u_2, \ldots, u_{56}$$

is a vector whose elements are related to the externally entered key

$$K = k_1, k_2, \ldots, k_{64}$$

as follows:

$$\begin{aligned} U = {} & k_{49}, k_{41}, k_{33}, k_{25}, k_{17}, k_9, k_1, \\ & k_{58}, k_{50}, k_{42}, k_{34}, k_{26}, k_{18}, k_{10}, \\ & k_2, k_{59}, k_{51}, k_{43}, k_{35}, k_{27}, k_{19}, \\ & k_{11}, k_3, k_{60}, k_{52}, k_{44}, k_{36}, k_{57}, \\ & k_{55}, k_{47}, k_{39}, k_{31}, k_{23}, k_{15}, k_7, \\ & k_{62}, k_{54}, k_{46}, k_{38}, k_{30}, k_{22}, k_{14}, \\ & k_6, k_{61}, k_{53}, k_{45}, k_{37}, k_{29}, k_{21}, \\ & k_{13}, k_5, k_{28}, k_{20}, k_{12}, k_4, k_{63} \end{aligned} \tag{3-26}$$

The vector U represents the key obtained by loading the external key of 64 bits into two 28-bit shift registers after the parity bits k_8, k_{16}, k_{24}, k_{32}, k_{40}, k_{48}, k_{56}, and k_{64} have been systematically removed as part of the loading procedure (Tables 3-8 and 3-9). Of the 56 register positions, 48 are connected to the Exclusive-OR circuits which perform the modulo 2 addition between R(i − 1) and K(i) (Figure 3-12). If the register positions are labeled 1 through 56, it follows that K(i) is obtained by taking the key bits located in positions

14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10,
23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2

of the first register, and positions

41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48,
44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32

of the second register. (See Equations 3-9a and 3-9b.)

Since U represents the key stored in the register at the start of the enciphering process, K(1) is related to U as follows:

$$\begin{aligned} K(1) = {} & u_{14}, u_{17}, u_{11}, u_{24}, u_1, u_5, u_3, u_{28}, u_{15}, u_6, u_{21}, u_{10}, \\ & u_{23}, u_{19}, u_{12}, u_4, u_{26}, u_8, u_{16}, u_7, u_{27}, u_{20}, u_{13}, u_{29}, \\ & u_{41}, u_{52}, u_{31}, u_{37}, u_{47}, u_{55}, u_{30}, u_{40}, u_{51}, u_{45}, u_{33}, u_{48}, \\ & u_{44}, u_{49}, u_{39}, u_{56}, u_{34}, u_{53}, u_{46}, u_{42}, u_{50}, u_{36}, u_{29}, u_{32} \end{aligned} \tag{3-27}$$

After each round, the key bits located in the shift register are shifted to the left by either one or two positions. The shifting employs wraparound (i.e., bits shifted off the left side of the registers are reinserted at the right side of the registers).

K(2) is obtained by shifting the contents of each register 1 bit to the left, whereas K(3) is obtained by a 2-bit shift to the left. (See Reference 5 or Tables 3-8 and 3-9.) Hence with K(1) specified, the relationship of the other keys, K(2) and K(3), to U can be expressed as follows:

$$\begin{aligned} K(2) = {} & u_{15}, u_{18}, u_{12}, u_{25}, u_2, u_6, u_4, u_1, \\ & u_{16}, u_7, u_{22}, u_{11}, u_{24}, u_{20}, u_{13}, u_5, \\ & u_{27}, u_9, u_{17}, u_8, u_{28}, u_{21}, u_{14}, u_3, \\ & u_{42}, u_{53}, u_{32}, u_{38}, u_{48}, u_{56}, u_{31}, u_{41}, \\ & u_{52}, u_{46}, u_{34}, u_{49}, u_{45}, u_{50}, u_{40}, u_{29}, \\ & u_{35}, u_{54}, u_{47}, u_{43}, u_{51}, u_{37}, u_{30}, u_{33} \end{aligned} \tag{3-28}$$

$$\begin{aligned} K(3) = {} & u_{17}, u_{20}, u_{14}, u_{27}, u_4, u_8, u_6, u_3, \\ & u_{18}, u_9, u_{24}, u_{13}, u_{26}, u_{22}, u_{15}, u_7, \\ & u_1, u_{11}, u_{19}, u_{10}, u_2, u_{23}, u_{16}, u_5, \\ & u_{44}, u_{55}, u_{34}, u_{40}, u_{50}, u_{30}, u_{33}, u_{43}, \\ & u_{54}, u_{48}, u_{36}, u_{51}, u_{47}, u_{52}, u_{42}, u_{31}, \\ & u_{37}, u_{56}, u_{49}, u_{45}, u_{53}, u_{39}, u_{32}, u_{35} \end{aligned} \tag{3-29}$$

The functional relations can be seen if F_i is partitioned into two matrices of 32 rows and 56 columns:

$$F_i = \begin{bmatrix} F_i^{(L)} \\ F_i^{(R)} \end{bmatrix}$$

where the elements in the submatrices express the relations shown in Table 3-21.

Evaluate F_1 first. From Equation 3-17, it follows that

$$L(1) = R(0)$$

hence L(1) does not depend on any key bits. Define the elements of the matrix F_i to be blank, as before, if no dependence on U exists. The elements of $F_1^{(L)}$ are thus blank.

Similarly, from Equation 3-18,

$$R(1) = L(0) \oplus g[K(1), R(0)]$$

hence R(1) depends on U via K(1) since L(0) and R(0) do not depend on U. As a first step in obtaining $F_1^{(R)}$, the dependence of the output bits from the substitution operation, vector B in Figure 3-12, on the key bits K(i) is evaluated in Figure 3-22. (Note that the dependence of B on K(i) is identical to

Submatrix	Relationship Expressed in Submatrix
$F_i^{(L)}$	L(i) vs. U
$F_i^{(R)}$	R(i) vs. U

Table 3-21. Functional Relationships

the dependence of R(i) on K(i) if the permutation is not present.) The selection of the substitution function in the first S-box, S_1, depends on the first and sixth bits of K(i) because the values of these bits are Exclusive-ORed with the first and sixth input bits to the S-box. The result of this operation in turn determines which of the four S-functions in S_1 is selected. Key bits 1 and 6 of K(i) therefore affect the output bits from S_1(b_1 through b_4) by influencing the selection of an S-function. This functional relationship is indicated by —. The other four key bits (bits 2, 3, 4, and 5 of K(i)) affect b_1 through b_4 by influencing the arguments of the S-function. This kind of dependence is indicated by the symbol x. If a bit from the key affects the

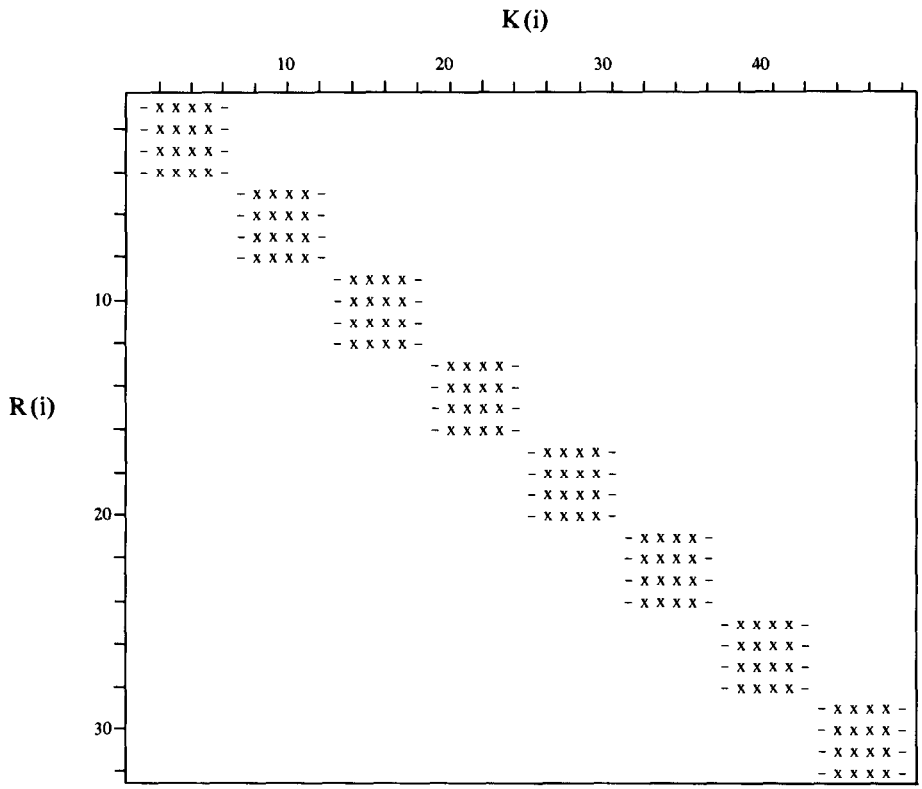


Figure 3-22. Functional Dependence of R(i) on K(i) without Permutation

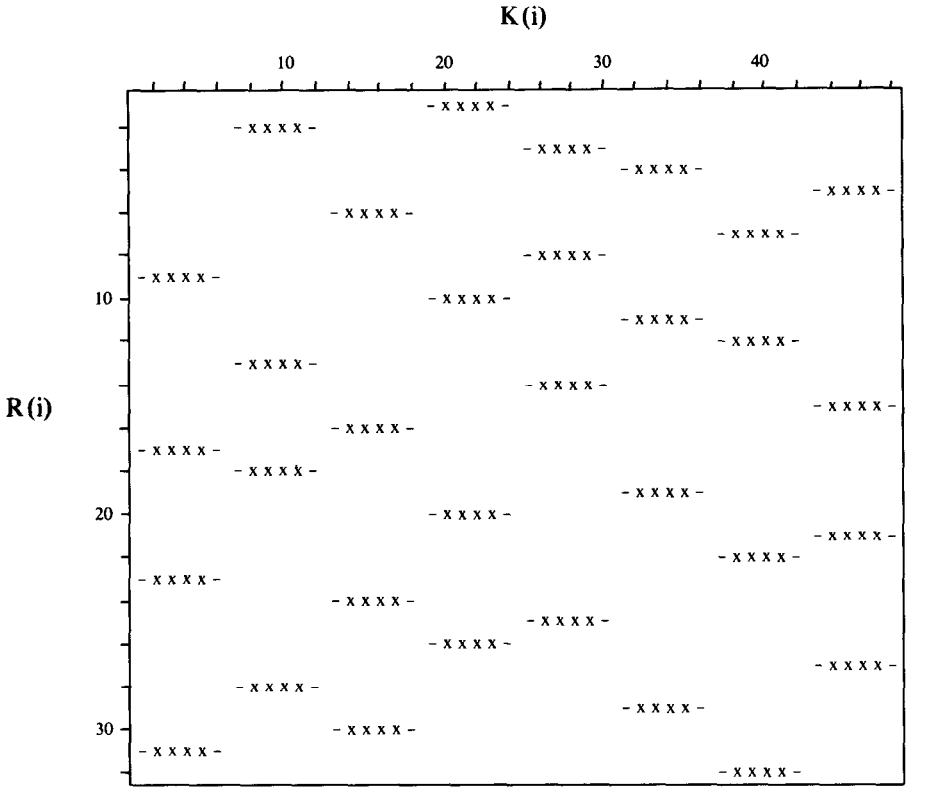


Figure 3-23. Functional Dependence of R(i) on K(i)

output by selecting an S-function as well as by selecting an argument, the symbol * is used. If the output does not depend on the key, a blank is used. This approach, used for all other bits of K(i), leads to Figure 3-22.

Because of this permutation, the rows of Figure 3-22 must be rearranged according to the relation given by $P(B)$ in Equation 3-16. The result is shown in Figure 3-23 and represents the type of relationship that exists between R(i) and K(i). The dependence of R(1) on U can be determined, in order to construct $F_1^{(R)}$, by using Equation 3-27, which relates each bit of K(1) to each bit of U. Figure 3-24 is constructed by replacing each bit of K(1) with the appropriate bit of U, and thus $F_1^{(R)}$ is obtained.

Evaluate F_2 next. From Equation 3-17, it follows that L(2) = R(1) and hence L(2) depends on U in the same way R(1) does. Since the dependence of L(2) on U is shown by $F_2^{(L)}$, and the dependence of R(1) on U is shown by $F_1^{(R)}$, according to Table 3-21 it follows that

$$F_2^{(L)} = F_1^{(R)} \tag{3-30}$$

From Equation 3-18, one obtains

$$R(2) = L(1) \oplus g\,[K(2), R(1)]$$

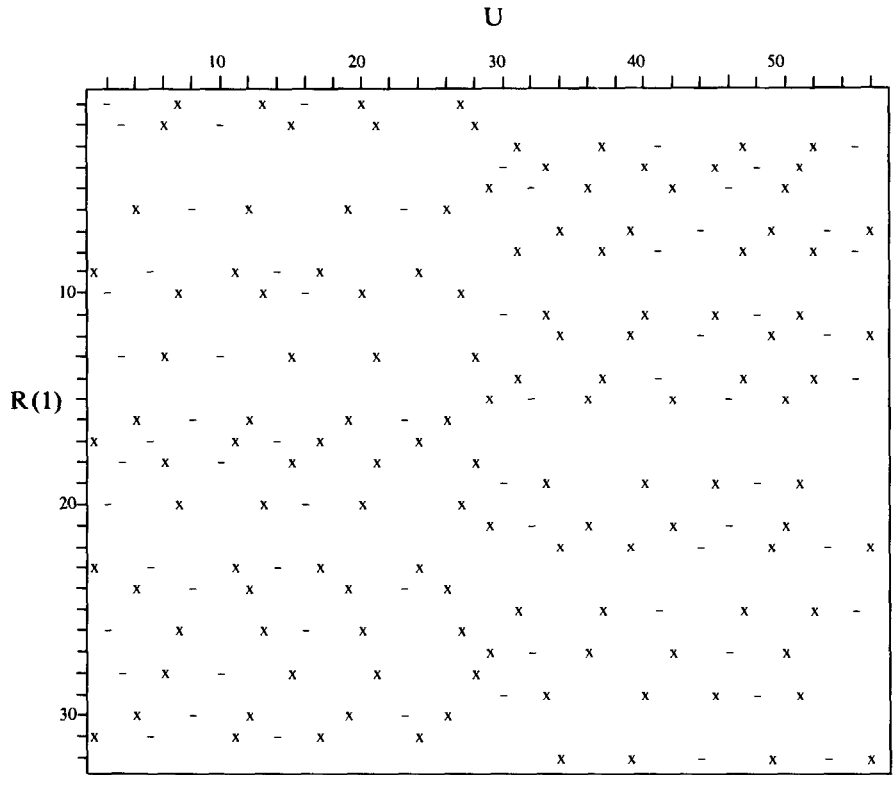


Figure 3-24. Functional Dependence of R(1) on U, Matrix $F_1^{(R)}$

The relationship between R(2) and R(1) was already established with the evaluation of $G_{i,i-1}^{(R,R)}$ (see Figures 3-16 and 3-17 for i = 2), whereas the relationship between R(1) and U is given by $F_1^{(R)}$ (see Figure 3-24). The relationship between R(2) and L(1) was also established before with the evaluation of $G_{i,i-1}^{(R,L)}$ (see Figure 3-17 for i = 2). But since L(1) does not depend on U, it can be ignored here. By the reasoning which led to Figure 3-18, it can be concluded that row s of $F_2^{(R)}$ can be constructed from F_1 by:

1. Combining rows $m_1(s)$ through $m_6(s)$ of $F_1^{(R)}$ with row s of $F_1^{(L)}$ (the elements of which are blank),
2. Taking into account the influence of K(2).

The impact of K(2) can be evaluated by using the entries of row s of Figure 3-23 and translating the elements of K(2) into the appropriate U-values via Equation 3-28. The method illustrated is by the construction of the fourth row of $F_2^{(R)}$. From Figure 3-23, it follows that the dependence of R(2) on K(2) is indicated by —xxxx— for bits 31 through 36 of K(2). The bits correspond to u_{31}, u_{41}, u_{52}, u_{46}, u_{34}, and u_{49}, respectively, according to Equation 3-28. The appropriate entries are shown in Figure 3-25. From Figure 3-16, one observes that for s = 4, $m_1(s)$ through $m_6(s)$ are equal to 20 through 25,

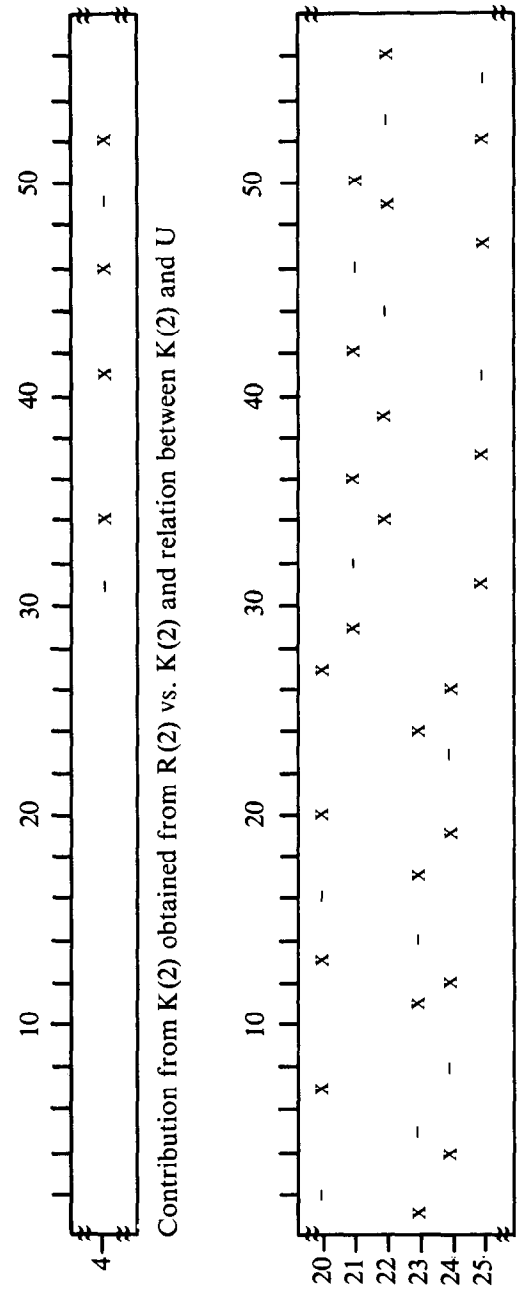


Figure 3-25. Detailed Steps to obtain Functional Dependence of 4th Bit of R(2) on U

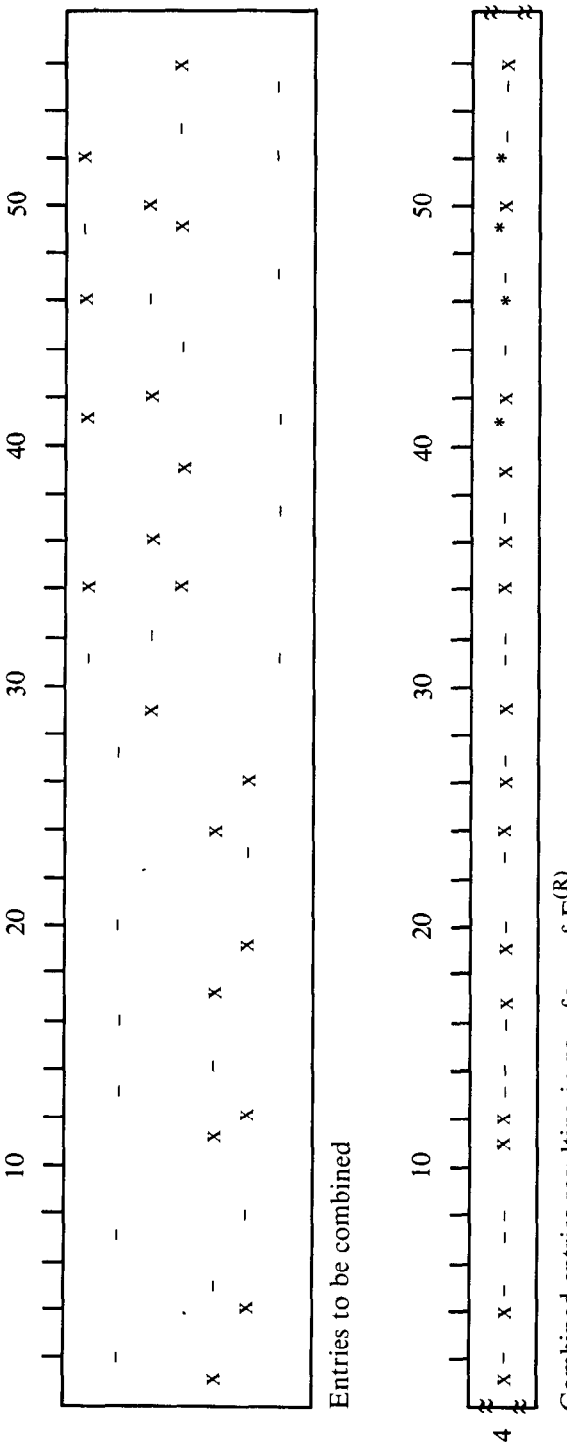


Figure 3-26. Functional Dependence of 4*th* Bit of R(2) on U

respectively. Hence rows 20 through 25 of Figure 3-24 must be used to obtain the contribution of R(1) to U. Details are shown in Figure 3-25. Due to rules similar to the ones established above for the ciphertext/plaintext dependence, the entries in the rows $m_1(s)$ and $m_6(s)$ of $F_1^{(R)}$ are changed to —, yielding the entries shown in Figure 3-26. The selection of an S-function is similar to autoclave-dependence, whereas the selection of an argument was treated like message-dependence above. By the same rules for combining several elements that were used above, the final result of the functional dependence of the fourth bit of R(2) on U is obtained as shown in Figure 3-26. Repetition of this procedure for $1 \leqslant s \leqslant 32$ yields Figure 3-27 and hence $F_2^{(R)}$ is obtained. Since $F_2^{(L)} = F_1^{(R)}$, as shown before, the matrix F_2 is thus obtained by combining Figure 3-24 and Figure 3-27. Let F_3 be evaluated next. The steps followed to obtain the functional dependence of the fourth bit of R(3) on U are shown in Figure 3-28. The final result (also shown in Figure 3-28) is obtained by following the same rules that led to the combination of Figure 3-25 and Figure 3-26. Repeating this procedure, all rows of $F_3^{(R)}$ shown in Figure 3-29 can be obtained. By using the relation $F_3^{(L)} = F_2^{(R)}$ the matrix F_3 is then obtained by combining Figure 3-27 and Figure 3-29.

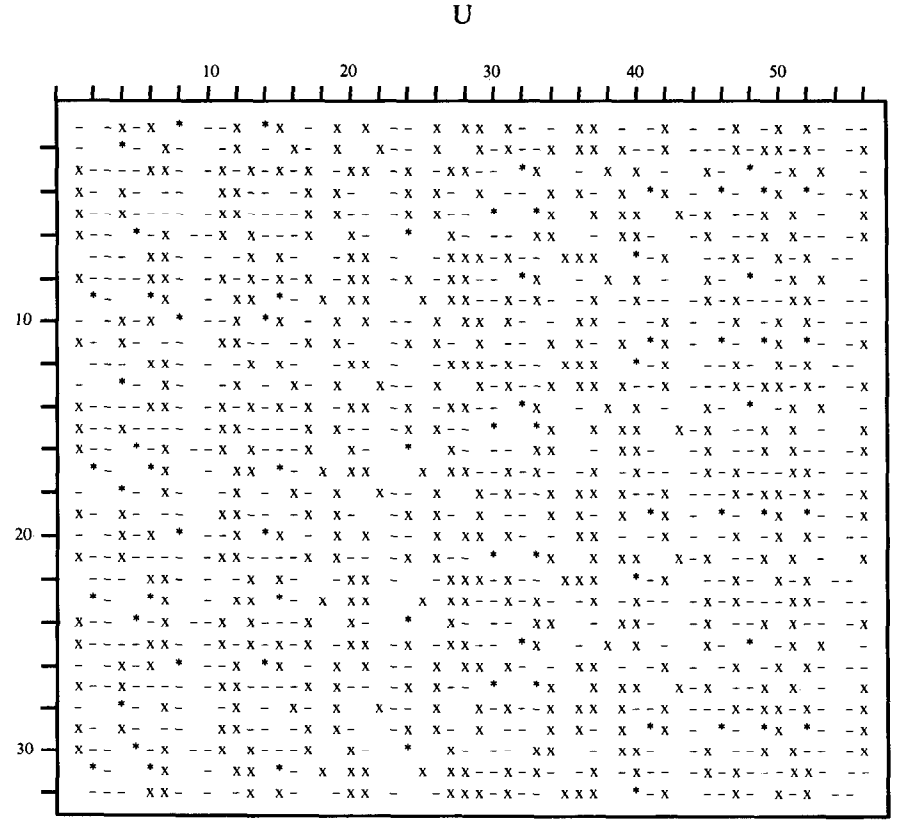
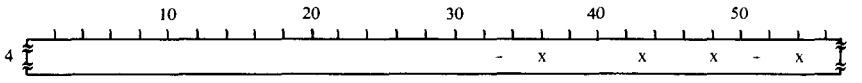
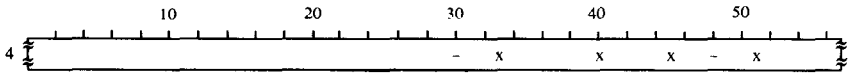


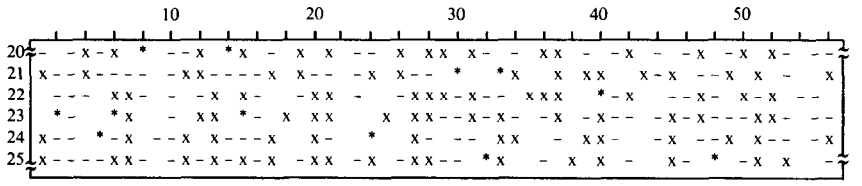
Figure 3-27. Funtional Dependence of R(2) on U, Matrix $F_2^{(R)}$



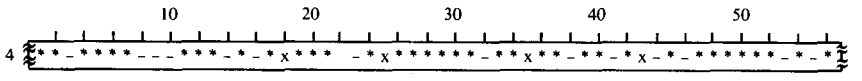
Contribution from K(3) obtained from R(3) vs. K(3), Figure 3-23, and relation between K(3) and U, Equation 3-29.



Contribution from $F_2^{(1)}$



Contribution from $F_2^{(R)}$. (Entries in row 20 and 25 must be set equal to - before combining.)



Combined entries resulting in row four of $F_3^{(R)}$

Figure 3-28. Functional Dependence of 4th Bit of R(3) on U

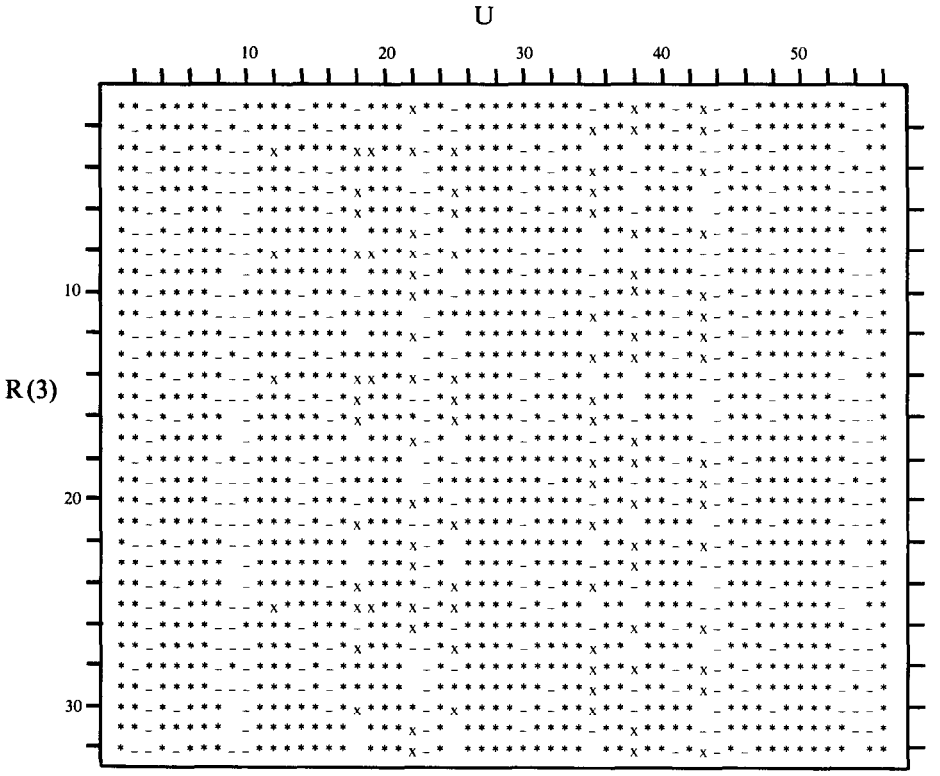


Figure 3-29. Functional Dependence of R(3) on U, Matrix $F_3^{(R)}$

Minimum Number of Rounds Required to Achieve Ciphertext/Key Intersymbol Dependence

For each output bit to depend on all bits of the key after round j, no element of F_j must be blank. If this condition is satisfied, then it follows from Equation 3-30 that no element of $F_{j-1}^{(R)}$ is blank.

An approximate calculation of the minimum number of rounds can be performed as follows. Figure 3-23 shows that K(j) always affects six columns in each row of $F_j^{(R)}$. Since L(0) and R(0) do not depend on the key, there are six nonblank entries in each row of $F_1^{(R)}$ (see also Figure 3-24), whereas all entries of $F_1^{(L)}$ are blank as shown before. Six rows of $F_1^{(R)}$ of six nonblank entries each and one row of $F_1^{(R)}$ (entries all blank) are combined to obtain $F_2^{(R)}$ to take into account the relationship between R(2) and X(1), whereas K(2) is taken into account in deriving six entries in each row (see also Figure 3-25). The maximum possible number of nonblank entries in each row of $F_2^{(R)}$ is thus $(6 \cdot 6) + 6 = 42$. (In the accurate analysis for DES shown in Figure 3-26, only 36 entries of the fourth row of matrix $F_2^{(R)}$ are nonblank since there is an overlap between elements of the rows to be combined.) Six rows of $F_2^{(R)}$ of at most 42 nonblank entries each, one row of $F_1^{(R)}$ of six nonblank entries each and six entries determined by the relationship between R(3) and K(3) are combined to obtain $F_3^{(R)}$. Since the total, $(6 \cdot 42) + 6 + 6$, exceeds the maximum number of possible entries (56) it is concluded that all rows of $F_2^{(R)}$ could be nonblank, which implies that at most all elements of $F_4^{(L)}$ could be nonblank. (In the accurate analysis for DES shown in Figure 3-28, there is still one nonblank entry, in row four of matrix $F_3^{(R)}$, again due to overlap between entries to be combined.) The conclusion of this approximate analysis is therefore that all $64 \cdot 56$ entries of F_4 could be nonblank after four rounds.

If an accurate analysis to calculate the percentage of nonblank elements, ξ, is performed, it is seen from Figures 3-24, 3-27, and 3-29 that $F_1^{(2)}$, $F_2^{(2)}$,

Round	Output/Input Relation		
j	L(j) vs. U	R(j) vs. U	X(j) vs. U
1	0.00	10.71	5.36
2	10.71	79.02	44.87
3	79.02	96.43	87.72
4	96.43	100.00	98.21
5	100.00	100.00	100.00

Note: Table entries express the degree of intersymbol dependence, i.e., the percentage of nonblank elements in the appropriate relations.

Table 3-22. Ciphertext/Key Intersymbol Dependence

$F_3^{(2)}$ have 1600, 376, and 64 blank entries, respectively. With this information and the fact that $F_j^{(2)} = F_{j-1}^{(1)}$ the values of ξ shown in Table 3-22 can be obtained. Since U is directly related to the supplied key (ignoring parity bits) according to Equation 3-26, the table entries do not change when U is replaced by K. Hence the relationships shown in Table 3-22 also indicate how the intersymbol dependence between ciphertext and key build up.

Summary and Conclusions

One property of DES is that each bit of ciphertext is a function of all plaintext bits and all cipher key bits. This property, defined as intersymbol dependence, has been analyzed above by evaluating how fast intersymbol dependence was achieved as a function of repeated mathematical operations defined as rounds. Each of these operations consists basically of substitution and transposition.

Three different forms of dependencies were considered:

1. If an input bit affects the selection of a substitution function in one round, the corresponding output was said to have an autoclave dependence on the input.
2. If an input bit affects the argument of a substitution function in one round, the corresponding output was said to have a message dependence on the input.
3. Finally, since the basic operation involving substitution and transposition is repeated several times, the functional relation between a ciphertext bit and a plaintext bit can be a combination of both of the relations defined above.

It has been shown that after five rounds each ciphertext bit depends on all plaintext bits via message- as well as autoclave-dependence. In addition, a similar analysis has revealed that each ciphertext bit depends on all key bits after five rounds.

The method applied to DES is, in general, applicable to ciphers which split up the input into two parts, operate on one part first, and combine the results of that operation with the other part using modulo 2 addition. In addition, the assumption is made that the S-box functions are nonaffine such that cancellation of dependencies does not occur.

REFERENCES

1. Shannon, C. E., "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, **28**, 656–715 (1949).
2. Feistel, H., "Block Cipher Cryptographic System," U.S. Patent No. 3,798,359 (March 19, 1974).
3. *IBM Research Reports*, **7**, No. 4, IBM Research, Yorktown Heights, NY (1971).

4. Ehrsam, W. F., Meyer, C. H., Powers, R. L., Smith, J. L., and Tuchman, W. L. "Product Block Cipher System for Data Security," U.S. Patent No. 3,962,539 (June 8, 1976).
5. *Data Encryption Standard,* Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington, DC (January 1977).
6. Vernam, G. S., "Cipher Printing Telegraph Systems for Secret Wire and Radio Telegraphic Communications," *Journal of the AIEE,* **45**, 109–115 (February 1926).
7. Kolman, B., *Elementary Linear Algebra,* Macmillan, London, 1971.
8. Golomb, S. W., *Shift Register Sequences,* Holden-Day, San Francisco, 1967.
9. Oystein, O., *Number Theory and Its History,* McGraw-Hill, New York, 1948.
10. Meyer, C. H. and Tuchman, W. L., "Pseudorandom Codes can be Cracked," *Electronic Design,* **23**, 74–76 (November 9, 1972).
11. Tuchman, W. L. and Meyer, C. H., "Efficacy of the Data Encryption Standard in Data Processing," *Proceedings COMPCON,* **78**, 340–347 (September 1978).
12. Bahl, L., "An Algorithm for Solving Simple Substitution Cryptograms," *International Symposium on Information Theory,* Ithaca, NY (October 1977).
13. Unclassified Summary, Involvement of NSA in the Development of the Data Encryption Standard, United States Senate Select Committee on Intelligence, Washington, DC (April 1978).
14. Branstad, D., Gait, J., and Katzke, S., *Report of the Workshop on Cryptography in Support of Computer Security,* Held at the National Bureau of Standards, NBSIR 77-1291, Systems and Software Division Institute for Computer Sciences and Technology, National Bureau of Standards, Washington, DC 29234 (September 1977).
15. Meissner, P., *Report of the 1976 Workshop on Estimation of Significant Advances in Computer Technology,* Held at the National Bureau of Standards, August 30-31, 1976 NBSIR 76-1199, Computer Systems Engineering Division, Institute for Computer Sciences and Technology, National Bureau of Standards, Washington, DC 29234 (December 1976).
16. Hoffman, L. J., *Modern Methods for Computer Security and Privacy,* Prentice-Hall, Englewood Cliffs, NJ, 1977.
17. Hellman, M., Merkle, R., Schroeppel, R., Washington, L., Diffie, W., Pohlig, S., and Schweitzer, P., *Results of an Initial Attempt to Cryptanalyze the NBS Data Encryption Standard,* Information Systems Laboratory Report, Stanford University (September 9, 1976) (Revised November 10, 1976.)
18. Harrison, M. A., *Introduction to Switching and Automata Theory,* McGraw-Hill, New York, 1965.
19. Hong, S. J., Cain, R. G., and Ostapko, D. L., "MINI: A Heuristic Approach for Logic Minimization," *IBM Journal of Research and Development,* **18**, No. 5, 445–458 (September 1974).
20. Meyer, C. H., "Ciphertext/Plaintext and Ciphertext/Key Dependence vs. Number of Rounds for the Data Encryption Standard," *AFIPS Conference Proceedings,* **47**, 1119–1126 (June 1978).

Other Publications that Treat Cryptanalysis

21. Coppersmith, D., and Grossman, E., "Generators for Certain Alternating Groups with Applications to Cryptography," *SIAM Journal on Applied Mathematics,* **29**, 624–627 (December 1975).
22. Grossman, E., "Group Theoretic Remarks on Cryptographic Systems Based on Two Types of Addition," IBM T. J. Watson Research Center, Yorktown Heights, NY, RC 4742 (February 1974).

23. Tuckerman, B., "A Study of the Vigenere-Vernam Single and Multiple Loop Enciphering Systems," IBM T. J. Watson Research Center, Yorktown Heights, NY, RC 2879 (May 1970).
24. Tuckerman, B., "Solution of a Substitution Fractionation Transposition Cipher," IBM T. J. Watson Research Center, Yorktown Heights, NY, RC 4537 (September 1973).
25. Konheim, A. G., *Cryptography: A Primer,* John Wiley, New York, 1981.