7.4.2 DES algorithm

DES is a Feistel cipher which processes plaintext blocks of $n = 64$ bits, producing 64-bit ciphertext blocks (Figure 7.8). The effective size of the secret key K is $k = 56$ bits; more precisely, the input key K is specified as a 64-bit key, 8 bits of which (bits $8, 16, \ldots, 64$) may be used as parity bits. The 2^{56} keys implement (at most) 2^{56} of the $2^{64}!$ possible bijections on 64-bit blocks. A widely held belief is that the parity bits were introduced to reduce the effective key size from 64 to 56 bits, to intentionally reduce the cost of exhaustive key search by a factor of 256.

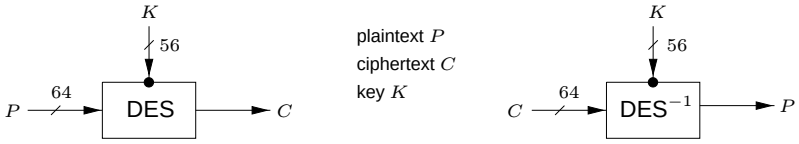

Figure 7.8: *DES input-output.*

Full details of DES are given in Algorithm 7.82 and Figures 7.9 and 7.10. An overview follows. Encryption proceeds in 16 stages or *rounds*. From the input key K, sixteen 48-bit subkeys K_i are generated, one for each round. Within each round, 8 fixed, carefully selected 6-to-4 bit substitution mappings (*S-boxes*) S_i, collectively denoted S, are used. The 64-bit plaintext is divided into 32-bit halves L_0 and R_0. Each round is functionally equivalent, taking 32-bit inputs L_{i-1} and R_{i-1} from the previous round and producing 32-bit outputs L_i and R_i for $1 \leq i \leq 16$, as follows:

$$L_i = R_{i-1}; \tag{7.4}$$

$$R_i = L_{i-1} \oplus f(R_{i-1},\ K_i), \quad \text{where } f(R_{i-1},\ K_i) = P(S(E(R_{i-1}) \oplus K_i)) \tag{7.5}$$

Here E is a fixed expansion permutation mapping R_{i-1} from 32 to 48 bits (all bits are used once; some are used twice). P is another fixed permutation on 32 bits. An initial bit permutation (IP) precedes the first round; following the last round, the left and right halves are exchanged and, finally, the resulting string is bit-permuted by the inverse of IP. Decryption involves the same key and algorithm, but with subkeys applied to the internal rounds in the reverse order (Note 7.84).

A simplified view is that the right half of each round (after expanding the 32-bit input to 8 characters of 6 bits each) carries out a key-dependent substitution on each of 8 characters, then uses a fixed bit transposition to redistribute the bits of the resulting characters to produce 32 output bits.

Algorithm 7.83 specifies how to compute the DES round keys K_i, each of which contains 48 bits of K. These operations make use of tables PC1 and PC2 of Table 7.4, which are called *permuted choice 1* and *permuted choice 2*. To begin, 8 bits $(k_8, k_{16}, \ldots, k_{64})$ of K are discarded (by PC1). The remaining 56 bits are permuted and assigned to two 28-bit variables C and D; and then for 16 iterations, both C and D are rotated either 1 or 2 bits, and 48 bits (K_i) are selected from the concatenated result.

7.82 Algorithm Data Encryption Standard (DES)

INPUT: plaintext $m_1 \dots m_{64}$; 64-bit key $K = k_1 \dots k_{64}$ (includes 8 parity bits).
OUTPUT: 64-bit ciphertext block $C = c_1 \dots c_{64}$. (For decryption, see Note 7.84.)

1. (key schedule) Compute sixteen 48-bit round keys K_i from K using Algorithm 7.83.
2. $(L_0, R_0) \leftarrow \text{IP}(m_1 m_2 \dots m_{64})$. (Use IP from Table 7.2 to permute bits; split the result into left and right 32-bit halves $L_0 = m_{58} m_{50} \dots m_8$, $R_0 = m_{57} m_{49} \dots m_7$.)
3. (16 rounds) for i from 1 to 16, compute L_i and R_i using Equations (7.4) and (7.5) above, computing $f(R_{i-1},\ K_i) = P(S(E(R_{i-1}) \oplus K_i))$ as follows:
   (a) Expand $R_{i-1} = r_1 r_2 \dots r_{32}$ from 32 to 48 bits using E per Table 7.3: $T \leftarrow E(R_{i-1})$. (Thus $T = r_{32} r_1 r_2 \dots r_{32} r_1$.)
   (b) $T' \leftarrow T \oplus K_i$. Represent T' as eight 6-bit character strings: $(B_1, \dots, B_8) = T'$.
   (c) $T'' \leftarrow (S_1(B_1), S_2(B_2), \dots S_8(B_8))$. (Here $S_i(B_i)$ maps $B_i = b_1 b_2 \dots b_6$ to the 4-bit entry in row r and column c of S_i in Table 7.8, page 260 where $r = 2 \cdot b_1 + b_6$, and $b_2 b_3 b_4 b_5$ is the radix-2 representation of $0 \le c \le 15$. Thus $S_1(011011)$ yields $r = 1$, $c = 13$, and output 5, i.e., binary 0101.)
   (d) $T''' \leftarrow P(T'')$. (Use P per Table 7.3 to permute the 32 bits of $T'' = t_1 t_2 \dots t_{32}$, yielding $t_{16} t_7 \dots t_{25}$.)
4. $b_1 b_2 \dots b_{64} \leftarrow (R_{16}, L_{16})$. (Exchange final blocks L_{16}, R_{16}.)
5. $C \leftarrow \text{IP}^{-1}(b_1 b_2 \dots b_{64})$. (Transpose using IP^{-1} from Table 7.2; $C = b_{40} b_8 \dots b_{25}$.)

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Table 7.2: *DES initial permutation and inverse (IP and IP^{-1}).*

E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Table 7.3: *DES per-round functions: expansion E and permutation P.*

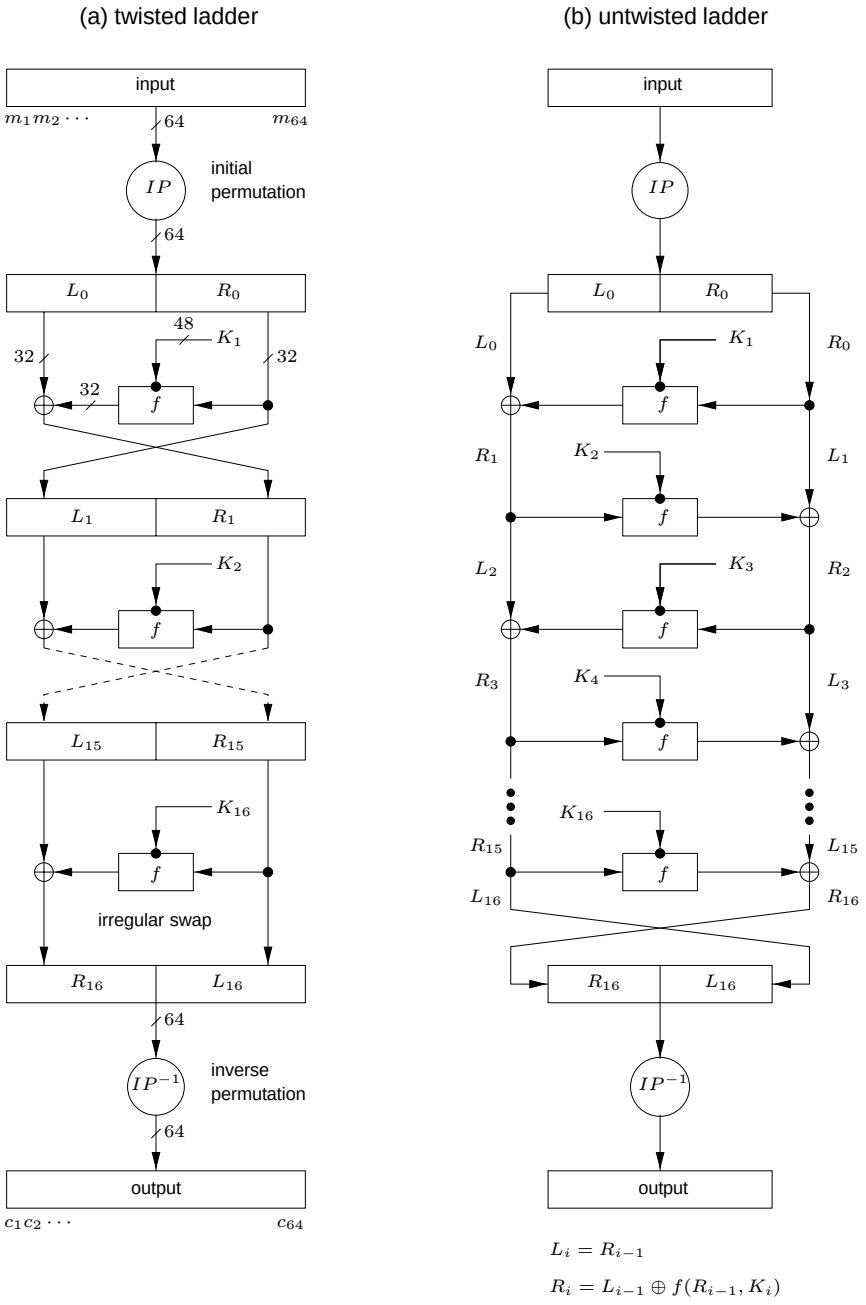

Figure 7.9: *DES computation path.*

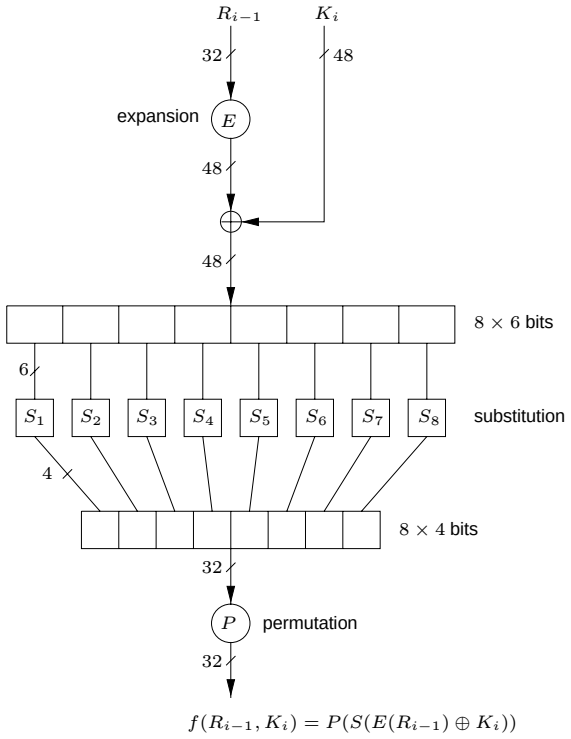

Figure 7.10: *DES inner function f.*

7.83 Algorithm DES key schedule

INPUT: 64-bit key $K = k_1 \dots k_{64}$ (including 8 odd-parity bits).
OUTPUT: sixteen 48-bit keys K_i, $1 \le i \le 16$.

1. Define v_i, $1 \le i \le 16$ as follows: $v_i = 1$ for $i \in \{1, 2, 9, 16\}$; $v_i = 2$ otherwise. (These are left-shift values for 28-bit circular rotations below.)
2. $T \leftarrow$ PC1(K); represent T as 28-bit halves (C_0, D_0). (Use PC1 in Table 7.4 to select bits from K: $C_0 = k_{57}k_{49} \dots k_{36}$, $D_0 = k_{63}k_{55} \dots k_4$.)
3. For i from 1 to 16, compute K_i as follows: $C_i \leftarrow (C_{i-1} \hookleftarrow v_i)$, $D_i \leftarrow (D_{i-1} \hookleftarrow v_i)$, $K_i \leftarrow$ PC2(C_i, D_i). (Use PC2 in Table 7.4 to select 48 bits from the concatenation $b_1 b_2 \dots b_{56}$ of C_i and D_i: $K_i = b_{14} b_{17} \dots b_{32}$. '$\hookleftarrow$' denotes left circular shift.)

If decryption is designed as a simple variation of the encryption function, savings result in hardware or software code size. DES achieves this as outlined in Note 7.84.

7.84 Note (*DES decryption*) DES decryption consists of the encryption algorithm with the same key but reversed key schedule, using in order $K_{16}, K_{15}, \dots, K_1$ (see Note 7.85). This works as follows (refer to Figure 7.9). The effect of IP^{-1} is cancelled by IP in decryption, leaving (R_{16}, L_{16}); consider applying round 1 to this input. The operation on the left half yields, rather than $L_0 \oplus f(R_0, K_1)$, now $R_{16} \oplus f(L_{16}, K_{16})$ which, since $L_{16} = R_{15}$ and $R_{16} = L_{15} \oplus f(R_{15}, K_{16})$, is equal to $L_{15} \oplus f(R_{15}, K_{16}) \oplus f(R_{15}, K_{16}) = L_{15}$. Thus round 1 decryption yields (R_{15}, L_{15}), i.e., inverting round 16. Note that the cancellation

PC1						
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
above for C_i; below for D_i						
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

PC2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	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

Table 7.4: *DES key schedule bit selections (PC1 and PC2).*

of each round is independent of the definition of f and the specific value of K_i; the swapping of halves combined with the XOR process is inverted by the second application. The remaining 15 rounds are likewise cancelled one by one in reverse order of application, due to the reversed key schedule.

7.85 Note (*DES decryption key schedule*) Subkeys $K_1, \ldots, K_{16}$ may be generated by Algorithm 7.83 and used in reverse order, or generated in reverse order directly as follows. Note that after K_{16} is generated, the original values of the 28-bit registers C and D are restored (each has rotated 28 bits). Consequently, and due to the choice of shift-values, modifying Algorithm 7.83 as follows generates subkeys in order $K_{16}, \ldots, K_1$: replace the left-shifts by right-shift rotates; change the shift value v_1 to 0.

7.86 Example (*DES test vectors*) The plaintext "Now is the time for all ", represented as a string of 8-bit hex characters (7-bit ASCII characters plus leading 0-bit), and encrypted using the DES key specified by the hex string K = `0123456789ABCDEF` results in the following plaintext/ciphertext:
$P =$ `4E6F772069732074 68652074696D6520 666F7220616C6C20`
$C =$ `3FA40E8A984D4815 6A271787AB8883F9 893D51EC4B563B53`. □

7.4.3 DES properties and strength

There are many desirable characteristics for block ciphers. These include: each bit of the ciphertext should depend on all bits of the key and all bits of the plaintext; there should be no statistical relationship evident between plaintext and ciphertext; altering any single plaintext or key bit should alter each ciphertext bit with probability $\frac{1}{2}$; and altering a ciphertext bit should result in an unpredictable change to the recovered plaintext block. Empirically, DES satisfies these basic objectives. Some known properties and anomalies of DES are given below.

(i) Complementation property

7.87 Fact Let E denote DES, and $\overline{x}$ the bitwise complement of x. Then $y = E_K(x)$ implies $\overline{y} = E_{\overline{K}}(\overline{x})$. That is, bitwise complementing both the key K and the plaintext x results in complemented DES ciphertext.

Justification: Compare the first round output (see Figure 7.10) to (L_0, R_0) for the uncomplemented case. The combined effect of the plaintext and key being complemented results

in the inputs to the XOR preceding the S-boxes (the expanded R_{i-1} and subkey K_i) both being complemented; this double complementation cancels out in the XOR operation, resulting in S-box inputs, and thus an overall result $f(R_0, K_1)$, as before. This quantity is then XORed (Figure 7.9) to $\overline{L}_0$ (previously L_0), resulting in $\overline{L}_1$ (rather than L_1). The same effect follows in the remaining rounds.

The complementation property is normally of no help to a cryptanalyst in known-plaintext exhaustive key search. If an adversary has, for a fixed unknown key K, a chosen-plaintext set of (x, y) data (P_1, C_1), $(\overline{P_1}, C_2)$, then $C_2 = E_K(\overline{P_1})$ implies $\overline{C_2} = E_{\overline{K}}(P_1)$. Checking if the key K with plaintext P_1 yields either C_1 or $\overline{C_2}$ now rules out two keys with one encryption operation, thus reducing the expected number of keys required before success from 2^{55} to 2^{54}. This is not a practical concern.

(ii) Weak keys, semi-weak keys, and fixed points

If subkeys K_1 to K_{16} are equal, then the reversed and original schedules create identical subkeys: $K_1 = K_{16}$, $K_2 = K_{15}$, and so on. Consequently, the encryption and decryption functions coincide. These are called weak keys (and also: *palindromic keys*).

7.88 Definition A DES *weak key* is a key K such that $E_K(E_K(x)) = x$ for all x, i.e., defining an involution. A pair of DES *semi-weak* keys is a pair (K_1, K_2) with $E_{K_1}(E_{K_2}(x)) = x$.

Encryption with one key of a semi-weak pair operates as does decryption with the other.

7.89 Fact DES has four weak keys and six pairs of semi-weak keys.

The four DES weak keys are listed in Table 7.5, along with corresponding 28-bit variables C_0 and D_0 of Algorithm 7.83; here $\{0\}^j$ represents j repetitions of bit 0. Since C_0 and D_0 are all-zero or all-one bit vectors, and rotation of these has no effect, it follows that all subkeys K_i are equal and an involution results as noted above.

The six pairs of DES semi-weak keys are listed in Table 7.6. Note their defining property (Definition 7.88) occurs when subkeys K_1 through K_{16} of the first key, respectively, equal subkeys K_{16} through K_1 of the second. This requires that a 1-bit circular left-shift of each of C_0 and D_0 for the first 56-bit key results in the (C_0, D_0) pair for the second 56-bit key (see Note 7.84), and thereafter left-rotating C_i and D_i one or two bits for the first results in the same value as right-rotating those for the second the same number of positions. The values in Table 7.6 satisfy these conditions. Given any one 64-bit semi-weak key, its paired semi-weak key may be obtained by splitting it into two halves and rotating each half through 8 bits.

7.90 Fact Let E denote DES. For each of the four DES weak keys K, there exist 2^{32} *fixed points* of E_K, i.e., plaintexts x such that $E_K(x) = x$. Similarly, four of the twelve semi-weak keys K each have 2^{32} *anti-fixed points*, i.e., x such that $E_K(x) = \overline{x}$.

The four semi-weak keys of Fact 7.90 are in the upper portion of Table 7.6. These are called *anti-palindromic keys*, since for these $K_1 = \overline{K_{16}}$, $K_2 = \overline{K_{15}}$, and so on.

(iii) DES is not a group

For a fixed DES key K, DES defines a permutation from $\{0,1\}^{64}$ to $\{0,1\}^{64}$. The set of DES keys defines 2^{56} such (potentially different) permutations. If this set of permutations was closed under composition (i.e., given any two keys K_1, K_2, there exists a third key K_3 such that $E_{K_3}(x) = E_{K_2}(E_{K_1}(x))$ for all x) then multiple encryption would be equivalent to single encryption. Fact 7.91 states that this is not the case for DES.

weak key (hexadecimal)	C_0	D_0
0101 0101 0101 0101	$\{0\}^{28}$	$\{0\}^{28}$
FEFE FEFE FEFE FEFE	$\{1\}^{28}$	$\{1\}^{28}$
1F1F 1F1F 0E0E 0E0E	$\{0\}^{28}$	$\{1\}^{28}$
E0E0 E0E0 F1F1 F1F1	$\{1\}^{28}$	$\{0\}^{28}$

Table 7.5: *Four DES weak keys.*

C_0	D_0	semi-weak key pair (hexadecimal)	C_0	D_0
$\{01\}^{14}$	$\{01\}^{14}$	01FE 01FE 01FE 01FE, FE01 FE01 FE01 FE01	$\{10\}^{14}$	$\{10\}^{14}$
$\{01\}^{14}$	$\{10\}^{14}$	1FE0 1FE0 0EF1 0EF1, E01F E01F F10E F10E	$\{10\}^{14}$	$\{01\}^{14}$
$\{01\}^{14}$	$\{0\}^{28}$	01E0 01E0 01F1 01F1, E001 E001 F101 F101	$\{10\}^{14}$	$\{0\}^{28}$
$\{01\}^{14}$	$\{1\}^{28}$	1FFE 1FFE 0EFE 0EFE, FE1F FE1F FE0E FE0E	$\{10\}^{14}$	$\{1\}^{28}$
$\{0\}^{28}$	$\{01\}^{14}$	011F 011F 010E 010E, 1F01 1F01 0E01 0E01	$\{0\}^{28}$	$\{10\}^{14}$
$\{1\}^{28}$	$\{01\}^{14}$	E0FE E0FE F1FE F1FE, FEE0 FEE0 FEF1 FEF1	$\{1\}^{28}$	$\{10\}^{14}$

Table 7.6: *Six pairs of DES semi-weak keys (one pair per line).*

7.91 Fact The set of 2^{56} permutations defined by the 2^{56} DES keys is not closed under functional composition. Moreover, a lower bound on the size of the group generated by composing this set of permutations is 10^{2499}.

The lower bound in Fact 7.91 is important with respect to using DES for multiple encryption. If the group generated by functional composition was too small, then multiple encryption would be less secure than otherwise believed.

(iv) Linear and differential cryptanalysis of DES

Assuming that obtaining enormous numbers of known-plaintext pairs is feasible, linear cryptanalysis provides the most powerful attack on DES to date; it is not, however, considered a threat to DES in practical environments. Linear cryptanalysis is also possible in a ciphertext-only environment if some underlying plaintext redundancy is known (e.g., parity bits or high-order 0-bits in ASCII characters).

Differential cryptanalysis is one of the most general cryptanalytic tools to date against modern iterated block ciphers, including DES, Lucifer, and FEAL among many others. It is, however, primarily a chosen-plaintext attack. Further information on linear and differential cryptanalysis is given in §7.8.

7.92 Note (*strength of DES*) The complexity (see §7.2.1) of the best attacks currently known against DES is given in Table 7.7; percentages indicate success rate for specified attack parameters. The 'processing complexity' column provides only an estimate of the expected cost (operation costs differ across the various attacks); for exhaustive search, the cost is in DES operations. Regarding storage complexity, both linear and differential cryptanalysis require only negligible storage in the sense that known or chosen texts can be processed individually and discarded, but in a practical attack, storage for accumulated texts would be required if ciphertext was acquired prior to commencing the attack.

attack method	data complexity		storage complexity	processing complexity
	known	chosen		
exhaustive precomputation	—	1	2^{56}	1 (table lookup)
exhaustive search	1	—	negligible	2^{55}
linear cryptanalysis	2^{43} (85%)	—	for texts	2^{43}
	2^{38} (10%)	—	for texts	2^{50}
differential cryptanalysis	—	2^{47}	for texts	2^{47}
	2^{55}	—	for texts	2^{55}

Table 7.7: *DES strength against various attacks.*

7.93 Remark (*practicality of attack models*) To be meaningful, attack comparisons based on different models (e.g., Table 7.7) must appropriately weigh the feasibility of extracting (acquiring) enormous amounts of chosen (known) plaintexts, which is considerably more difficult to arrange than a comparable number of computing cycles on an adversary's own machine. Exhaustive search with one known plaintext-ciphertext pair (for ciphertext-only, see Example 7.28) and 2^{55} DES operations is significantly more feasible in practice (e.g., using highly parallelized custom hardware) than linear cryptanalysis (LC) requiring 2^{43} known pairs.

While exhaustive search, linear, and differential cryptanalysis allow recovery of a DES key and, therefore, the entire plaintext, the attacks of Note 7.8, which become feasible once about 2^{32} ciphertexts are available, may be more efficient if the goal is to recover only part of the text.

7.5 FEAL

The Fast Data Encipherment Algorithm (FEAL) is a family of algorithms which has played a critical role in the development and refinement of various advanced cryptanalytic techniques, including linear and differential cryptanalysis. FEAL-N maps 64-bit plaintext to 64-bit ciphertext blocks under a 64-bit secret key. It is an N-round Feistel cipher similar to DES (cf. Equations (7.4), (7.5)), but with a far simpler f-function, and augmented by initial and final stages which XOR the two data halves as well as XOR subkeys directly onto the data halves.

FEAL was designed for speed and simplicity, especially for software on 8-bit microprocessors (e.g., chipcards). It uses byte-oriented operations (8-bit addition mod 256, 2-bit left rotation, and XOR), avoids bit-permutations and table look-ups, and offers small code size. The initial commercially proposed version with 4 rounds (FEAL-4), positioned as a fast alternative to DES, was found to be considerably less secure than expected (see Table 7.10). FEAL-8 was similarly found to offer less security than planned. FEAL-16 or FEAL-32 may yet offer security comparable to DES, but throughput decreases as the number of rounds rises. Moreover, whereas the speed of DES implementations can be improved through very large lookup tables, this appears more difficult for FEAL.

Algorithm 7.94 specifies FEAL-8. The f-function $f(A, Y)$ maps an input pair of 32×16 bits to a 32-bit output. Within the f function, two byte-oriented data substitutions (S-boxes) S_0 and S_1 are each used twice; each maps a pair of 8-bit inputs to an 8-bit output