3.4 DATA ENCRYPTION STANDARD (DES)

3.4.1 Background and History

The **Data Encryption Standard (DES)**, also called the Data Encryption Algorithm (DEA) by ANSI and DEA-1 by ISO, has been a cryptographic algorithm used for over two decades. Of late, DES has been found vulnerable against very powerful attacks, and therefore, the popularity of DES has been slightly on the decline. However, no book on security is complete without DES, as it has been a landmark in cryptographic algorithms. We shall also discuss DES at length to achieve two objectives: Firstly to learn about DES, but secondly and more importantly, to dissect and understand a real-life cryptographic algorithm. Using this philosophy, we shall then discuss some other cryptographic algorithms, but only at a conceptual level; because the in-depth discussion of DES would have already helped us understand in depth, how computer-based cryptographic algorithms work. DES is generally used in the ECB, CBC or the CFB mode.

The origins of DES go back to 1972, when in the US, the National Bureau of Standards (NBS), now known as the National Institute of Standards and Technology (NIST), embarked upon a project for protecting the data in computers and computer communications. They wanted to develop a single cryptographic algorithm. After two years, NBS realized that IBM's **Lucifer** could be considered a serious candidate, rather than developing a fresh algorithm from scratch. After a few discussions, in 1975, the NBS published the details of the algorithm. Towards the end of 1976, the US Federal Government decided to adopt this algorithm, and soon, it was renamed *Data Encryption Standard (DES)*. Soon, other bodies also recognized and adopted DES as a cryptographic algorithm.

3.4.2 How DES Works

1. Basic Principles

DES is a block cipher. It encrypts data in blocks of 64 bits each. That is, 64 bits of plain text goes as the input to DES, which produces 64 bits of cipher text. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits. The basic idea is shown in Fig. 3.18.

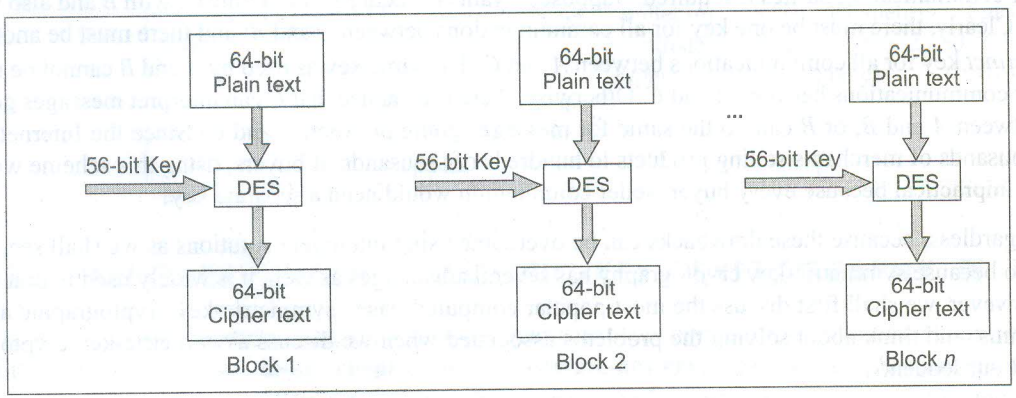

Fig. 3.18 Conceptual working of DES

We have mentioned that DES uses a 56-bit key. Actually, the initial key consists of 64 bits. However, before the DES process even starts, every eighth bit of the key is discarded to produce a 56-bit key. That is, bit positions 8, 16, 24, 32, 40, 48, 56 and 64 are discarded. This is shown in Fig. 3.19 with shaded bit positions indicating discarded bits. (Before discarding, these bits can be used for parity checking to ensure that the key does not contain any errors.)

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	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	57	58	59	60	61	62	63	64

Fig. 3.19 Discarding of every 8th bit of the original key (shaded bit positions are discarded)

Thus, the discarding of every 8th bit of the key produces a 56-bit key from the original 64-bit key, as shown in Fig. 3.20.

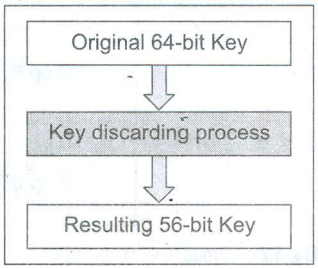

Fig. 3.20 Key-discarding process

Simplistically, DES is based on the two fundamental attributes of cryptography: substitution (also called confusion) and transposition (also called diffusion). DES consists of 16 steps, each of which is called a **round**. Each *round* performs the steps of substitution and transposition. Let us now discuss the broad-level steps in DES.

1. In the first step, the 64-bit plain-text block is handed over to an **Initial Permutation (IP)** function.
2. The initial permutation is performed on plain text.
3. Next, the Initial Permutation (IP) produces two halves of the permuted block; say Left Plain Text (LPT) and Right Plain Text (RPT).
4. Now, each of LPT and RPT go through 16 *rounds* of encryption process, each with its own key.
5. In the end, LPT and RPT are rejoined, and a **Final Permutation (FP)** is performed on the combined block.
6. The result of this process produces 64-bit cipher text.

This process is shown diagrammatically in Fig. 3.21.

Let us now understand each of these processes in detail.

2. Initial Permutation (IP)

As we have noted, the Initial Permutation (IP) happens only once, and it happens before the first round. It suggests how the transposition in IP should proceed, as shown in Fig. 3.22. For example, it says that the IP replaces the first bit of the original plain-text block with the 58th bit of the original plain-text block, the second bit with the 50th bit of the original plain text block, and so on. This is nothing but jugglery of bit positions of the original plain-text block.

The complete transposition table used by IP is shown in Fig. 3.23. This table (and all others in this chapter) should be read from left to right, top to bottom. For instance, we have noted that 58 in the first position indicates that the contents of the 58th bit in the original plain-text block will overwrite the contents of the 1st bit position, during IP. Similarly, 1 is shown at the 40th position in the table, which

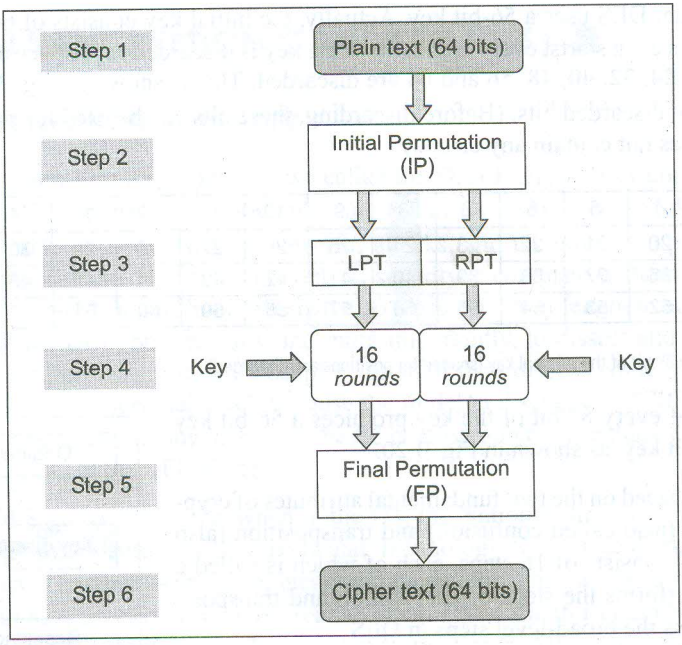

Fig. 3.21 Broad level steps in DES

Bit position in the plain-text block	To be overwritten with the contents of this bit position
1	58
2	50
3	42
..	..
64	7

Fig. 3.22 Idea of 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

Fig. 3.23 Initial Permutation (IP) table

means that the first bit will overwrite the 40th bit in the original plain-text block. The same rule applies for all other bit positions.

As we have noted, after IP is done, the resulting 64-bit permuted text block is divided into two half blocks. Each half block consists of 32 bits. We have called the left block as LPT and the right block as RPT. Now, 16 *rounds* are performed on these two blocks. We shall discuss this process now.

3. Rounds

Each of the 16 rounds, in turn, consists of the broad-level steps outlined in Fig. 3.24.

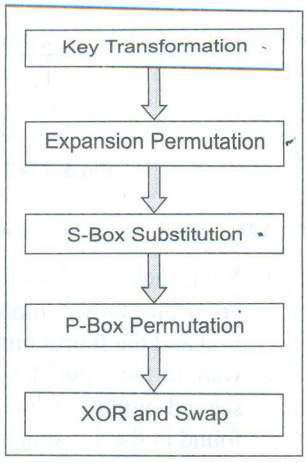

Fig. 3.24 Details of one *round* in DES

Let us discuss these details step-by-step.

Step 1: Key Transformation We have noted that the initial 64-bit key is transformed into a 56-bit key by discarding every 8th bit of the initial key. Thus, for each round, a 56-bit key is available. From this 56-bit key, a different 48-bit **subkey** is generated during each *round* using a process called **key transformation**. For this, the 56-bit key is divided into two halves, each of 28 bits. These halves are circularly shifted left by one or two positions, depending on the round. For example, if the *round* number is 1, 2, 9 or 16, the shift is done by only one position. For other rounds, the circular shift is done by two positions. The number of key bits shifted per round is shown in Fig. 3.25.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number of key bits shifted	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Fig. 3.25 Number of key bits shifted per *round*

After an appropriate shift, 48 of the 56 bits are selected. For selecting 48 of the 56 bits, the table shown in Fig. 3.26 is used. For instance, after the shift, bit number 14 moves into the first position, bit number 17 moves into the second position, and so on. If we observe the table carefully, we will realize that it contains only 48 bit positions. Bit number 18 is discarded (we will not find it in the table), like 7 others, to reduce the 56-bit key to a 48-bit key. Since the key-transformation process involves permutation as well as selection of a 48-bit subset of the original 56-bit key, it is called **compression permutation**.

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

Fig. 3.26 Compression permutation

Because of this compression permutation technique, a different subset of key bits is used in each round. That makes DES more difficult to crack.

Step 2: Expansion Permutation Recall that after initial permutation, we had two 32-bit plain text areas, called Left Plain Text (LPT) and Right Plain Text (RPT). During **expansion permutation**, the RPT is expanded from 32 bits to 48 bits. Besides increasing the bit size from 32 to 48, the bits are permuted as well, hence the name *expansion permutation*. This happens as follows:

1. The 32-bit RPT is divided into 8 blocks, with each block consisting of 4 bits. This is shown in Fig. 3.27.

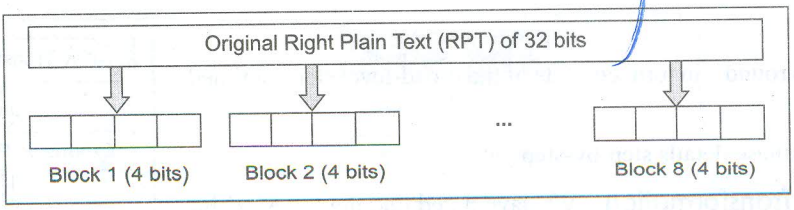

Fig. 3.27 Division of 32-bit RPT into eight 4-bit blocks

2. Next, each 4-bit block of the above step is then expanded to a corresponding 6-bit block. That is, per 4-bit block, 2 more bits are added. What are these two bits? They are actually the repeated first and the fourth bits of the 4-bit block. The second and the third bits are written down as they were in the input. This is shown in Fig. 3.28. Note that the first bit inputted is outputted to the second output position, and also repeats in output position 48. Similarly, the 32nd input bit is found in the 47th output position as well as in the first output position.

Clearly, this process results into expansion as well as permutation of the input bits while creating the output.

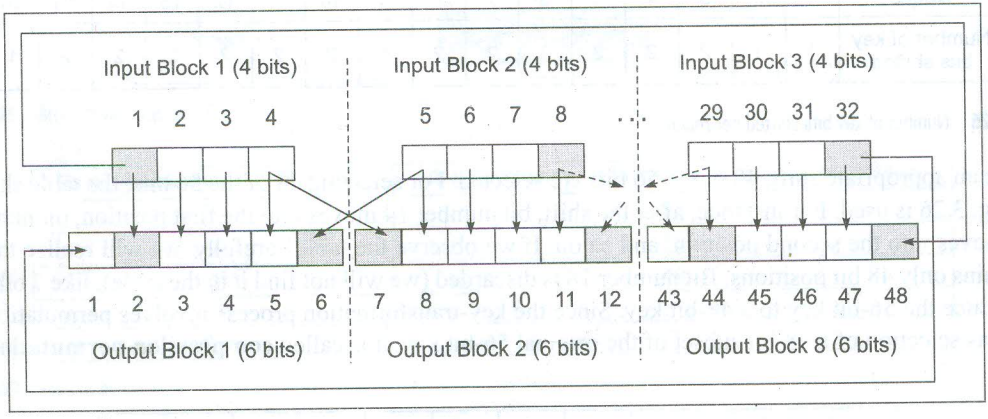

Fig. 3.28 RPT expansion permutation process

As we can see, the first input bit goes into the second and the 48th output positions. The second input bit goes into the third output position, and so on. Consequently, we will observe that the expansion permutation has actually used the table shown in Fig. 3.29.

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

Fig. 3.29 RPT expansion permutation table

As we have seen, firstly, the *key-transformation* process compresses the 56-bit key to 48 bits. Then, the *expansion permutation* process expands the 32-bit RPT to 48 bits. Now, the 48-bit key is XORed

with the 48-bit RPT, and the resulting output is given to the next step, which is the **S-box substitution** (which we shall discuss in the next section) as shown in Fig. 3.30.

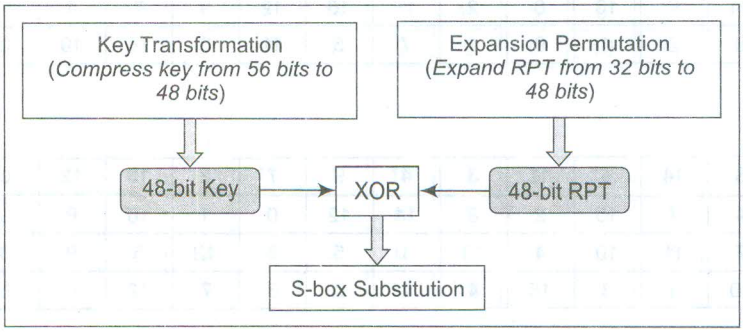

Fig. 3.30 Way to S-box substitution

Step 3: S-box Substitution **S-box substitution** is a process that accepts the 48-bit input from the XOR operation involving the compressed key and expanded RPT, and produces a 32-bit output using the substitution technique. The substitution is performed by eight **substitution boxes** (also called as **S-boxes**). Each of the eight S-boxes has a 6-bit input and a 4-bit output. The 48-bit input block is divided into 8 sub-blocks (each containing 6 bits), and each such sub-block is given to an S-box. The S-box transforms the 6-bit input into a 4-bit output, as shown in Fig. 3.31.

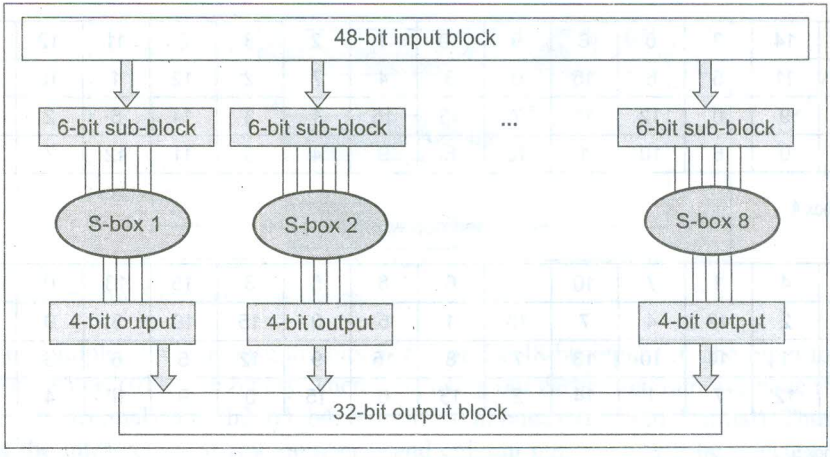

Fig. 3.31 S-box substitution

What is the logic used by S-box substitution for selecting only four of the six bits? We can conceptually think of every S-box as a table that has 4 rows (numbered 0 to 3) and 16 columns (numbered 0 to 15). Thus, we have 8 such tables, one for each S-box. At the intersection of every row and column, a 4-bit number (which will be the 4-bit output for that S-box) is present. This is shown in Fig. 3.32.

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Fig. 3.32(a) S-box 1

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Fig. 3.32(b) S-box 2

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Fig. 3.32(c) S-box 3

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Fig. 3.32(d) S-box 4

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Fig. 3.32(e) S-box 5

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Fig. 3.32(f) S-box 6

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Fig. 3.32(g) S-box 7

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Fig. 3.32(h) S-box 8

The 6-bit input indicates which row and column, and therefore, which intersection is to be selected, and thus, determines the 4-bit output. How is it done? Let us assume that the six bits of an S-box are indicated by b1, b2, b3, b4, b5, and b6. Now, bits b1 and b6 are combined to form a two-bit number. Two bits can store any decimal number between 0 (binary 00) and 3 (binary 11). This specifies the row number. The remaining four bits b2, b3, b4, b5 make up a four-bit number, which specifies the column number between decimal 0 (binary 0000) and 15 (binary 1111). Thus, the 6-bit input automatically selects the row number and column number for the selection of the output. This is shown in Fig. 3.33.

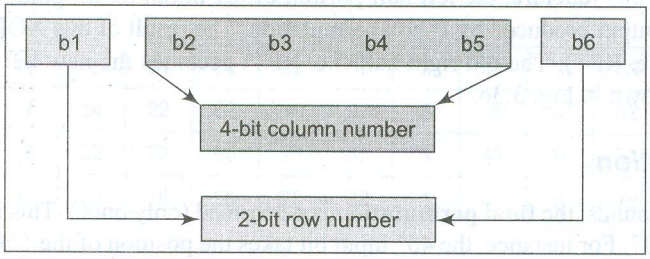

Fig. 3.33 Selecting an entry in an S-box based on the 6-bit input

Let us take an example now. Suppose the bits 5 to 8 of the 48-bit input (i.e. the input to the second S-box) contain a value 101101 in binary. Therefore, using our earlier diagram, we have (b1, b6) = 11 in binary (i.e. 3 in decimal), and (b2, b3, b4, b5) = 0110 in binary (i.e. 6 in decimal). Thus, the output of S-box 2 at the intersection of row number 3 and column number 6 will be selected, which is 4. (Remember to count rows and columns from 0, not 1). This is shown in Fig. 3.34.

The output of each S-box is then combined to form a 32-bit block, which is given to the last stage of a *round*, the P-box permutation, as discussed next.

Step 4: P-box Permutation The output of S-box consists of 32 bits. These 32 bits are permuted using a **P-box**. This straightforward permutation mechanism involves simple permutation (i.e. replacement of each bit with another bit, as specified in the P-box table, without any expansion or compression). This is called **P-box permutation**. The P-box is shown in Fig. 3.35. For example, a 16 in the first block indicates that the bit at position 16 of the original input moves to the bit at position 1 in the

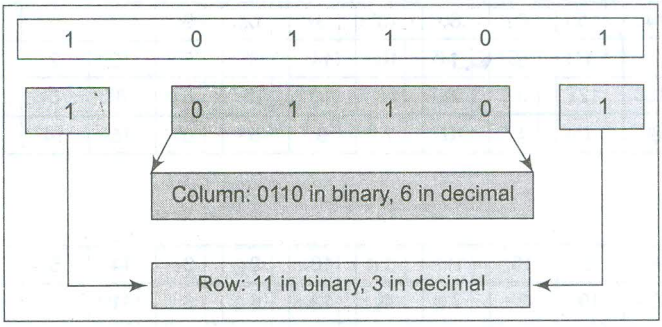

Fig. 3.34 Example of selection of S-box output based on the input

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

Fig. 3.35 P-box permutation

output, and a 10 in the block number 16 indicates that the bit at the position 10 of the original input moves to bit at the position 16 in the output.

Step 5: XOR and Swap Note that we have been performing all these operations only on the 32-bit right half portion of the 64-bit original plain text (i.e. on the RPT). The left half portion (i.e. LPT) was untouched so far. At this juncture, the left half portion of the initial 64-bit plain text block (i.e. LPT) is XORed with the output produced by P-box permutation. The result of this XOR operation becomes the new right half (i.e. RPT). The old right half (i.e. RPT) becomes the new left half, in a process of swapping. This is shown in Fig. 3.36.

4. Final Permutation

At the end of the 16 rounds, the **final permutation** is performed (only once). This is a simple transposition based on Fig. 3.37. For instance, the 40th input bit takes the position of the 1st output bit, and so on.

The output of the final permutation is the 64-bit encrypted block.

5. DES Decryption

From the above discussion of DES, we might get a feeling that it is an extremely complicated encryption scheme, and therefore, the decryption using DES would employ a completely different approach. To most people's surprise, the same algorithm used for encryption in DES also works for decryption! The values of the various tables and the operations as well as their sequence are so carefully chosen that the algorithm is reversible. The only difference between the encryption and the decryption process is the reversal of key portions. If the original key K was divided into *K1, K2, K3, ..., K16* for the 16 encryption *rounds*, then for decryption, the key should be used as *K16, K15, K14, ..., K1*.

6. Analyzing DES

(a) Use of S-boxes The tables used for substitution, i.e. the S-boxes, in DES are kept secret by IBM. IBM maintains that it took them over 17 person years to come up with the internal design of the

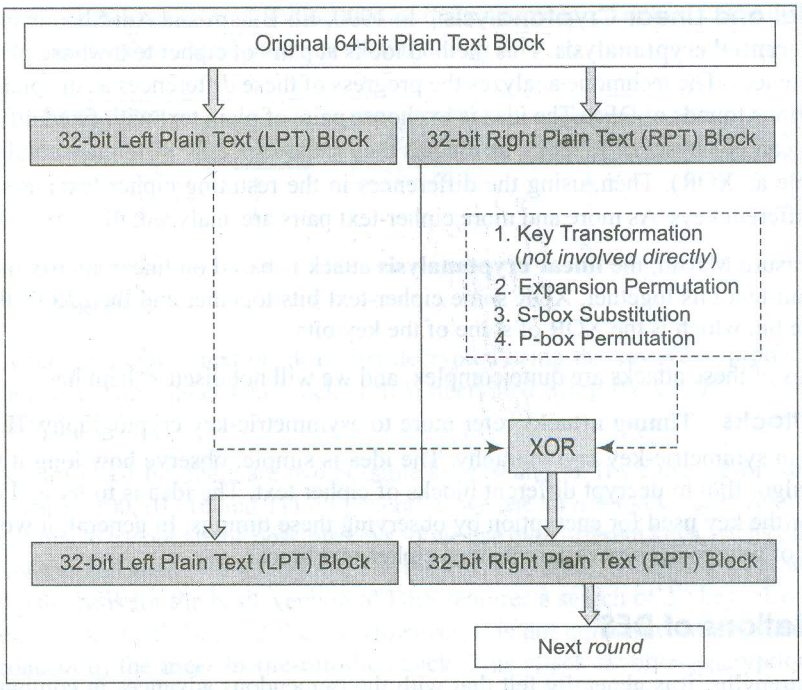

Fig. 3.36 XOR and swap

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

Fig. 3.37 Final permutation

S-boxes. Over the years, suspicion has grown that there is some vulnerability in this aspect of DES, intentional (so that the government agencies could secretly open encrypted messages) or otherwise. Several studies keep appearing, which suggest that there is some scope for attacks on DES via the S-boxes. However, no concrete example has emerged till date.

(b) Key Length We have mentioned earlier that any cryptographic system has two important aspects: the cryptographic algorithm and the key. The inner workings of the DES algorithm (which we have discussed earlier) are completely known to the general public. Therefore, the strength of DES lies only in the other aspect—its key, which must be secret.

As we know, DES uses 56-bit keys. (Interestingly, the original proposal was to make use of 112-bit keys.) Thus, there are 2^{56} possible keys (which is roughly equal to 7.2×10^{16} keys). Thus, it seems that a brute-force attack on DES is impractical. Even if we assume that to obtain the correct key, only half of the possible keys (i.e. the half of the **key space**) needs to be examined and tried out, a single computer performing one DES encryption per microsecond would require more than 1,000 years to break DES.

(c) Differential and Linear Cryptanalysis In 1990, Eli Biham and Adi Shamir introduced the concept of **differential cryptanalysis**. This method looks at pairs of cipher text whose plain texts have particular differences. The technique analyzes the progress of these differences as the plain texts travel through the various rounds of DES. The idea is to choose pairs of plain text with fixed differences. The two plain texts can be chosen at random, as long as they satisfy specific difference conditions (which can be as simple as XOR). Then, using the differences in the resulting cipher texts, assign different likelihood to different keys. As more and more cipher-text pairs are analyzed, the correct key emerges.

Invented by Mitsuru Matsui, the **linear cryptanalysis** attack is based on linear approximations. If we XOR some plain-text bits together, XOR some cipher-text bits together and then XOR the result, we will get a single bit, which is the XOR of some of the key bits.

The descriptions of these attacks are quite complex, and we will not discuss them here.

(d) Timing Attacks **Timing attacks** refer more to asymmetric-key cryptography. However, they can also apply to symmetric-key cryptography. The idea is simple: observe how long it takes for the cryptographic algorithm to decrypt different blocks of cipher text. The idea is to try and obtain either the plain text or the key used for encryption by observing these timings. In general, it would take different amounts of time to decrypt different sized cipher-text blocks.

3.4.3 Variations of DES

In spite of its strengths, it is generally felt that with the tremendous advances in computer hardware (higher processing speeds of gigahertz, higher memory availability at cheap prices, parallel processing capabilities, etc.), DES is susceptible to possible attacks. However, because DES is already proven to be a very competent algorithm, it would be wise to reuse DES by making it stronger by some means, rather than writing a new cryptographic algorithm. Writing a new algorithm is not easy, more so because it has to be tested sufficiently so as to be proved as a strong algorithm. Consequently, two main variations of DES have emerged, which are **double DES** and **triple DES**. Let us discuss them now.

1. Double DES

Double DES is quite simple to understand. Essentially, it does twice what DES normally does only once. Double DES uses two keys, say *K1* and *K2*. It first performs DES on the original plain text using *K1* to get the encrypted text. It again performs DES on the encrypted text, but this time with the other key, i.e. *K2*. The final output is the encryption of encrypted text (i.e. the original plain text encrypted twice with two different keys). This is shown in Fig. 3.38.

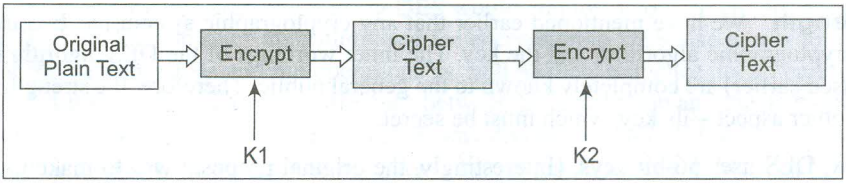

Fig. 3.38 Double DES encryption

Of course, there is no reason why double encryption cannot be applied to other cryptographic algorithms as well. However, in the case of DES, it is already quite popular, and therefore, we have dis-

cussed this in the context of DES. It should also be quite simple to imagine that the decryption process would work in exactly the reverse order, as shown in Fig. 3.39.

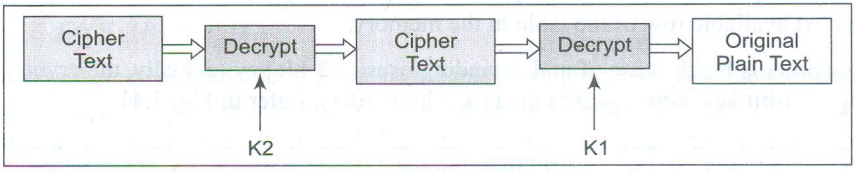

Fig. 3.39 Double DES decryption

The doubly encrypted cipher-text block is first decrypted using the key *K2* to produce the singly encrypted cipher text. This cipher-text block is then decrypted using the key *K1* to obtain the original plain-text block.

If we use a key of just 1 bit, there are two possible keys (0 and 1). If we use a 2-bit key, there are four possible key values (00, 01, 10 and 11). In general, if we use an n-bit key, the cryptanalyst has to perform 2^n operations to try out all the possible keys. If we use two different keys, each consisting of n bits, the cryptanalyst would need 2^{2n} attempts to crack the key. Therefore, on the face of it, we may think that since the cryptanalysis for the basic version of DES requires a search of 2^{56} keys, Double DES would require a key search of (2^{2*56}), i.e. 2^{112} keys. However, it is not quite true. Merkle and Hellman introduced the concept of the **meet-in-the-middle** attack. This attack involves encryption from one end, decryption from the other, and matching the results in the middle, hence the name *meet-in-the-middle* attack. Let us understand how it works.

Suppose that the cryptanalyst knows two basic pieces of information: P (a plain-text block), and C (the corresponding final cipher-text block) for a message. We know that the relations shown in Fig. 3.40 are true for P and C, if we are using double DES. The mathematical equivalents of these are also shown. The result of the first encryption is called T, and is denoted as $T = E_{K1}(P)$ [i.e. encrypt the block P with key $K1$]. After this encrypted block is encrypted with another key $K2$, we denote the result as $C = E_{K2}(E_{K1}(P))$ [i.e. encrypt the already encrypted block T, with a different key $K2$, and call the final cipher text as C].

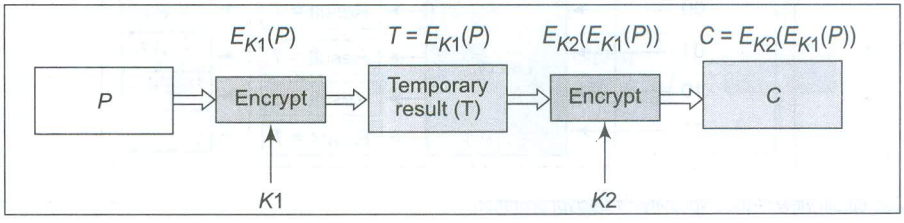

Fig. 3.40 Mathematical expression of double DES

Now, the aim of the cryptanalyst, who is armed with the knowledge of P and C, is to obtain the values of $K1$ and $K2$. What would the cryptanalyst do?

Step 1 For all possible values (2^{56}) of key $K1$, the cryptanalyst would use a large table in the memory of the computer, and perform the following two steps:

1. The cryptanalyst would encrypt the plain-text block P by performing the first encryption operation, i.e. $E_{K1}(P)$. That is, it will calculate T.
2. The cryptanalyst would store the output of the operation $E_{K1}(P)$, i.e. the temporary cipher text (T), in the next available row of the table in the memory.

We show this process for the ease of understanding using a 2-bit key (actually, the cryptanalyst has to do this using a 64-bit key, which makes the task a lot harder). Refer to Fig. 3.41.

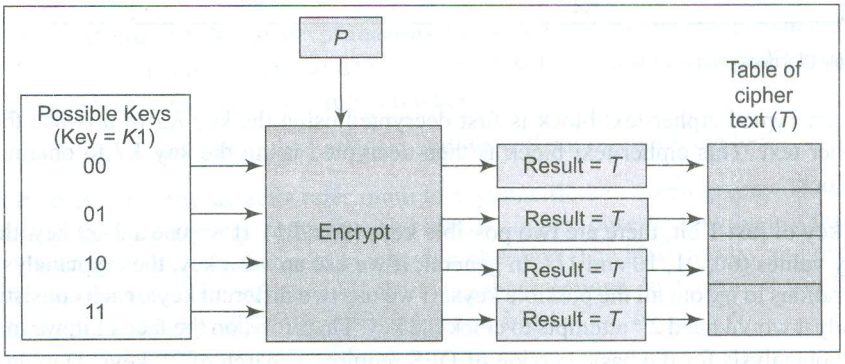

Fig. 3.41 Conceptual view of the cryptanalyst's *Encrypt* operation

Step 2 Thus, at the end of the above process, the cryptanalyst will have the table of cipher texts as shown in the figure. Next, the cryptanalyst will perform the reverse operation. That is, he/she will now decrypt the known cipher text C with all the possible values of $K2$ [i.e. perform $D_{K2}(C)$ for all possible values of $K2$]. In each case, the cryptanalyst will compare the resulting value with all the values in the table of cipher texts, which were computed earlier. This process (as before, for 2-bit key) is shown in Fig. 3.42.

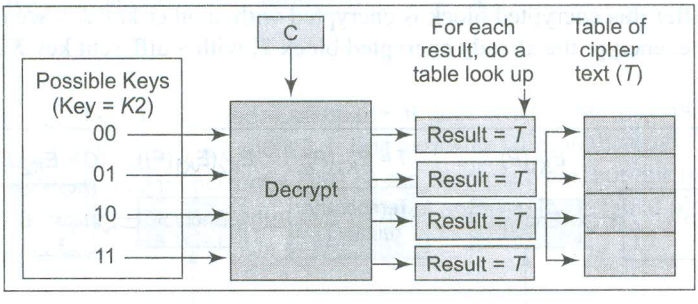

Fig. 3.42 Conceptual view of the cryptanalyst's *Decrypt* operation

To summarize:

- In the first step, the cryptanalyst was calculating the value of T from the left-hand side (i.e. encrypt P with $K1$ to find T). Thus, here $T = E_{K1}(P)$.
- In the second step, the cryptanalyst was finding the value of T from the right-hand side (i.e. decrypt C with $K2$ to find T). Thus, here $T = D_{K2}(C)$.

From the above two steps, we can actually conclude that the temporary result (T) can be obtained in two ways, either by encrypting P with $K1$, or by decrypting C with $K2$. This is because, we can write the following equations:

$$T = E_{K1}(P) = D_{K2}(C)$$

Now, if the cryptanalyst creates a table of $E_{K1}(P)$ (i.e. table of T) for all the possible values of $K1$, and then performs $D_{K2}(C)$ for all possible values of $K2$ (i.e. computes T), there is a chance that she gets the same T in both the operations. If the cryptanalyst is able to find the same T for both *encrypt with K1* and *decrypt with K2* operations, it means that the cryptanalyst knows not only P and C, but he/she has now also been able to find out the possible values of $K1$ and $K2$!

The cryptanalyst can now try this $K1$ and $K2$ pair on another known pair of P and C, and if he she is able to get the same T by performing the $E_{K1}(P)$ and $D_{K2}(C)$ operations, he/she can then try to use $K1$ and $K2$ for the remaining blocks of the message.

Clearly, this attack is possible, but requires a lot of memory. For an algorithm that uses 64-bit plain-text blocks and 56-bit keys, we would need 2^{56} 64-bit blocks to store the table of T in memory (there is no point in storing it on the disk, as it would be too slow, and defeat the very purpose of the attack). This is equivalent to 10^{17} bytes, which is too high for the next few generations of computers!

2. *Triple DES*

Although the *meet-in-the-middle* attack on double DES is not quite practical yet, in cryptography, it is always better to take the minimum possible chances. Consequently, double DES seemed inadequate, paving way for **triple DES**. As we can imagine, Triple DES is *DES three times*. It comes in two kinds: one that uses three keys, and the other that uses two keys. We will study both, one by one.

(a) Triple DES with Three Keys The idea of triple DES with three keys is illustrated in Fig. 3.43. As we can see, the plain-text block P is first encrypted with a key $K1$, then encrypted with a second key $K2$, and finally with a third key, $K3$, where $K1$, $K2$ and $K3$ are all different from each other.

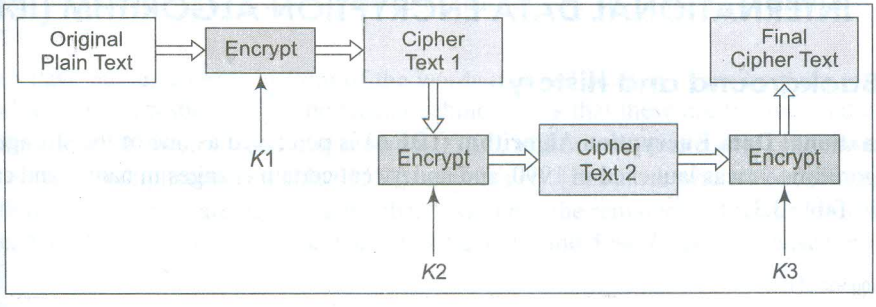

Fig. 3.43 Triple DES with three keys

Triple DES with three keys is used quite extensively in many products, including PGP and S/MIME. To decrypt the cipher text C and obtain the plain text P, we need to perform the operation $P = D_{K1}(D_{K2}(D_{K3}(C)))$.

(b) Triple DES with Two Keys Triple DES with three keys is highly secure. It can be denoted in the form of an equation as $C = E_{K3}(E_{K2}(E_{K1}(P)))$. However, triple DES with three keys also has the drawback of requiring $56 \times 3 = 168$ bits for the key, which can be slightly difficult to have in practical

situations. A workaround suggested by Tuchman uses just two keys for triple DES. Here, the algorithm works as follows:

1. Encrypt the plain text with key $K1$. Thus, we have $E_{K1}(P)$.
2. Decrypt the output of step 1 above with key $K2$. Thus, we have $D_{K2}(E_{K1}(P))$.
3. Finally, encrypt the output of step 2 again with key $K1$. Thus, we have $E_{K1}(D_{K2}(E_{K1}(P)))$.

This is shown in Fig. 3.44.

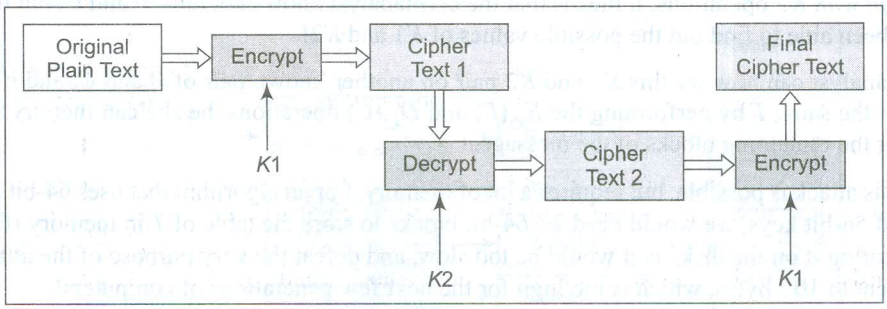

Fig. 3.44 Triple DES with two keys

To decrypt the cipher text C and obtain the original plain text P, we need to perform the operation $P = D_{K1}(E_{K2}(D_{K1}(C)))$.

There is no special meaning attached to the second step of decryption. Its only significance is that it allows triple DES to work with two, rather than three keys. This is also called **Encrypt-Decrypt-Encrypt (EDE)** mode. Triple DES with two keys is not susceptible to the *meet-in-the-middle* attack, unlike double DES as $K1$ and $K2$ alternate here.

3.5 INTERNATIONAL DATA ENCRYPTION ALGORITHM (IDEA)

3.5.1 Background and History

The **International Data Encryption Algorithm (IDEA)** is perceived as one of the strongest cryptographic algorithms. It was launched in 1990, and underwent certain changes in names and capabilities as shown in Table 3.3.

Table 3.3 Progress of IDEA

Year	Name	Description
1990	Proposed Encryption Standard (PES).	Developed by Xuejia Lai and James Massey at the Swiss Federal Institute of Technology.
1991	Improved Proposed Encryption Standard (IPES).	Improvements in the algorithm as a result of cryptanalysts finding some areas of weakness.
1992	International Data Encryption Algorithm (IDEA).	No major changes, simply renamed.