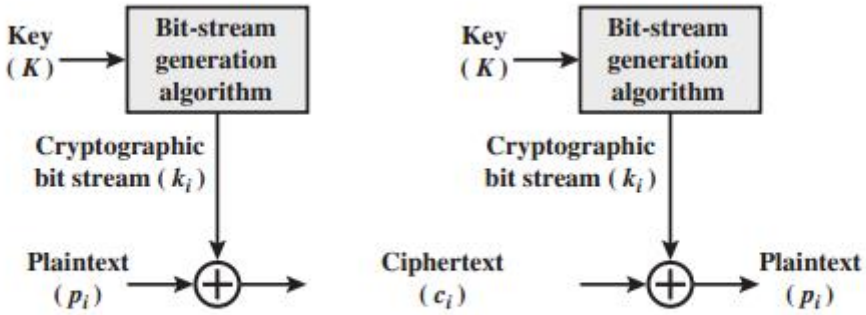
Chapter- 5

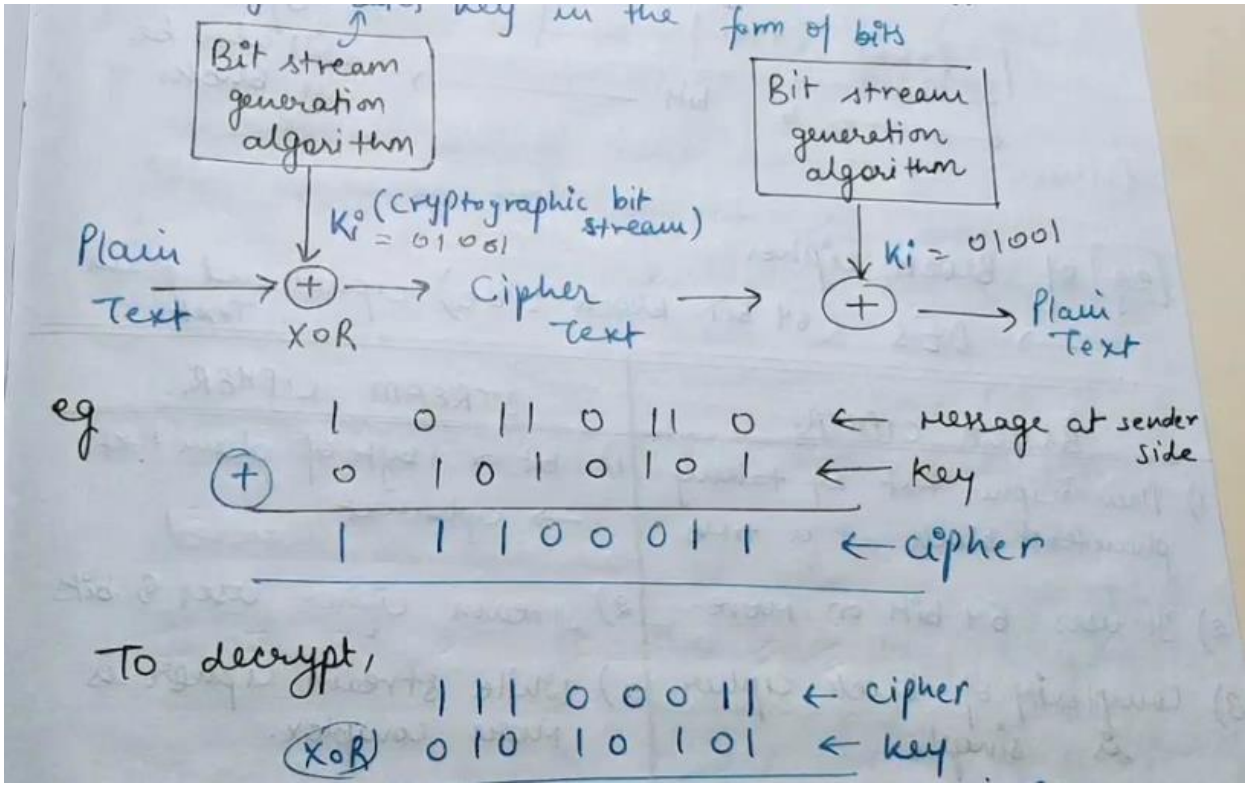
BLOCK CIPHERS AND THE DATA ENCRYPTIONSTANDARD

Stream Ciphers and Block Ciphers

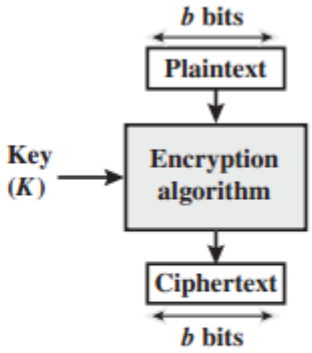
A **stream cipher** is one that encrypts a digital data stream one bit or one byte at a time. Examples of classical stream ciphers are the autokeyed Vigenère cipher and the Vernam cipher. In the ideal case, a one-time pad version of the Vernam cipher would be used.



(a) Stream cipher using algorithmic bit-stream generator

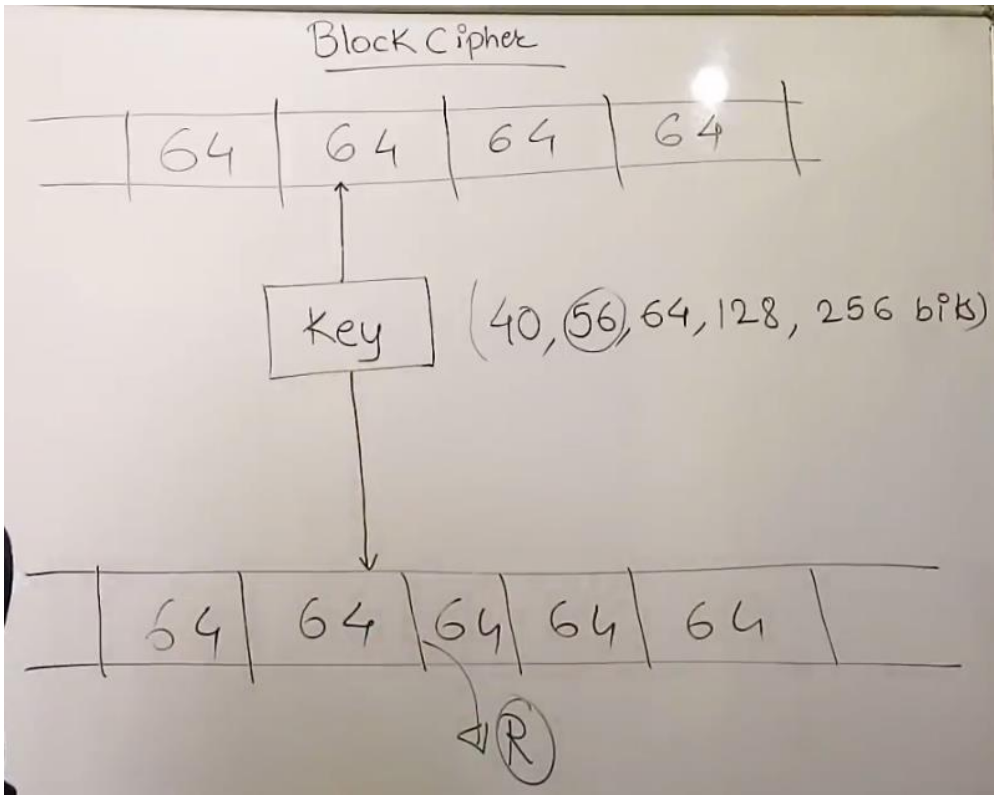


A **block cipher** is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a block size of 64 or 128 bits is used. As with a stream cipher, the two users share a symmetric encryption key (Figure 3.1b).



(b) Block cipher

Figure 3.1 Stream Cipher and Block Cipher



Difference between Stream and Block Cipher

Basis for comparison	Block cipher	Stream cipher
Basic	Converts the plain text by taking its block at a time.	Converts the text by taking one byte of the plain text at a time.
Complexity	Simple design	Complex comparatively
No of bits used	64 Bits or more	8 Bits
Confusion and Diffusion	Uses both confusion and diffusion	Relies on confusion only
Algorithm modes used	ECB (Electronic Code Book) CBC (Cipher Block Chaining)	CFB (Cipher Feedback) OFB (Output Feedback)
Reversibility	Reversing encrypted text is hard.	It uses XOR for the encryption which can be easily reversed to the plain text.
Implementation	Feistel Cipher	Vernam Cipher

FEISTEL CIPHER STRUCTURE

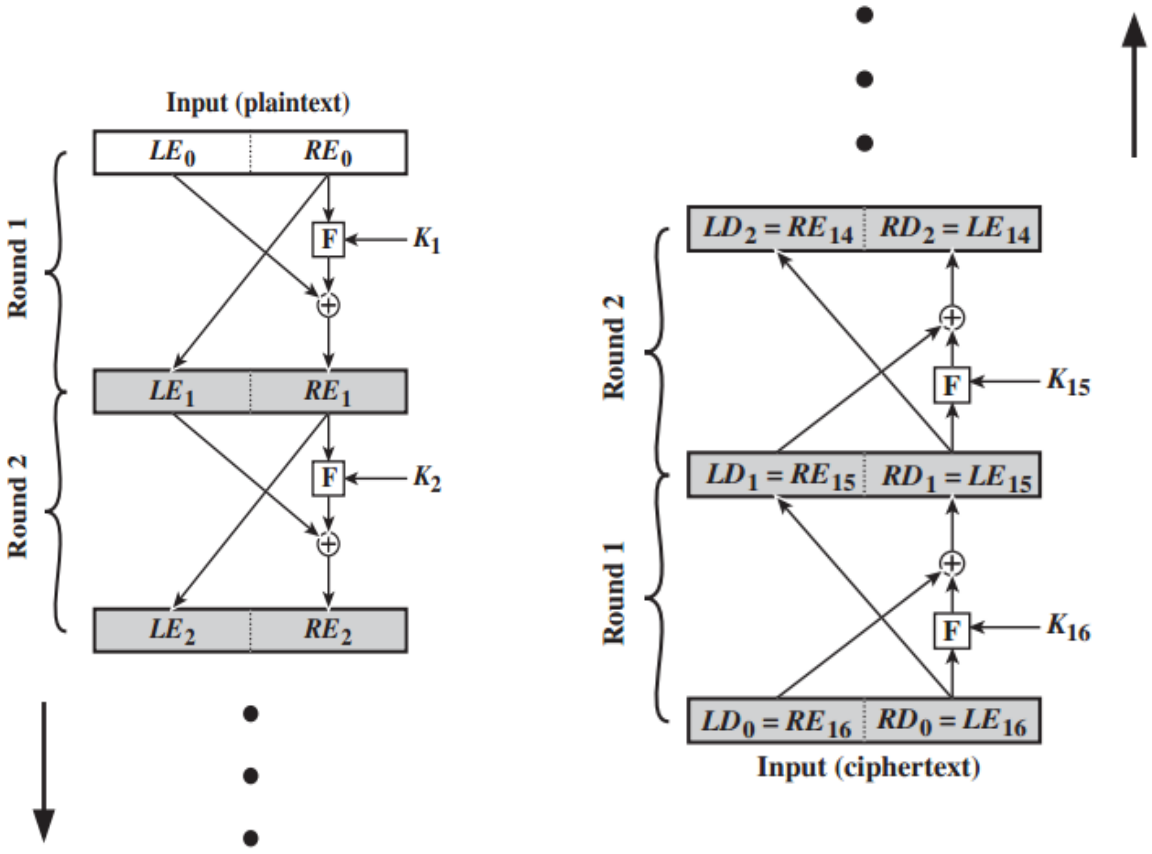
The left-hand side of depicts the structure proposed by Feistel. The inputs to the encryption algorithm are a plaintext block of length 2w bits and a key K . The plaintext block is divided into two halves, L_0 and R_0.

The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block. Each round i has as inputs L_{i-1} and R_{i-1} derived from the previous round, as well as a subkey K_i derived from the overall K. In general, the subkeys K_i are different from K and from each other. In Figure 3.3, 16 rounds are used, although any number of rounds could be implemented.

All rounds have the same structure. A **substitution** is performed on the left half of the data. This is done by applying a *round function* F to the right half of the data and then taking the exclusive-OR of the output of that function and the left half of the data. The round function has the same general structure for each round but is parameterized by the round subkey K_i. Another way to express this is to say that F is a function of right-half block of w bits and a subkey of y bits, which produces an output value of length w bits: $F(RE_i, K_{i+1})$. Following this substitution, a

permutation is performed that consists of the interchange of the two halves of the data.[6] This structure is a particular form of the substitution-permutation network (SPN) proposed by Shannon.

Fig Encryption and Decryption in Fiestel Cipher



The exact realization of a Feistel network depends on the choice of the following parameters and design features:

- **Block size:** Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed for a given algorithm. The greater security is achieved by greater diffusion. Traditionally, a block size of 64 bits has been considered a reasonable tradeoff and was nearly universal in block cipher design. However, the new AES uses a 128-bit block size.
- **Key size:** Larger key size means greater security but may decrease encryption/decryption speed. The greater security is achieved by greater resistance to brute-force attacks and greater confusion. Key sizes of 64 bits or less are now widely considered to be inadequate, and 128 bits has become a common size.
- **Number of rounds:** The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.
- **Subkey generation algorithm:** Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.
- **Round function F:** Again, greater complexity generally means greater resistance to cryptanalysis.

THE DATA ENCRYPTION STANDARD

The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46). The algorithm itself is referred to as the Data Encryption Algorithm (DEA).[7] For DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption.

DES Encryption

The overall scheme for DES encryption is illustrated in Figure 3.5. As with any encryption scheme, there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the plaintext must be 64 bits in length and the key is 56 bits in length.[8]

DES (DATA ENCRYPTION ALGORITHM)

Follows Feistel structure

→ Block Size – 64 bit Plain Text

→ No. of Rounds – 16 Rounds

→ Key Size – 64 bit

→ No. of Sub Keys – 16 Sub Keys

→ Sub Key Size – 48 bit Sub key

→ Cipher Text – 64 bit cipher Text

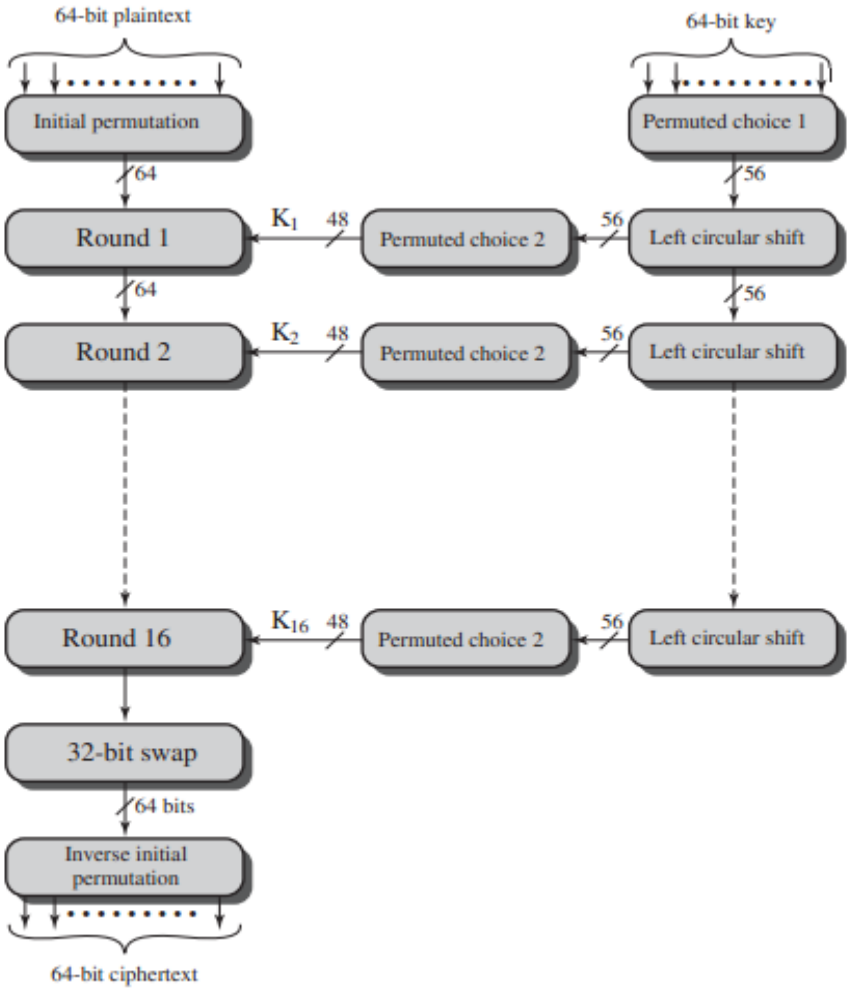


Figure 3.5 General Depiction of DES Encryption Algorithm

Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in three phases. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*. This is followed by a phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the **preoutput**. Finally, the preoutput is passed through a permutation [IP^{-1}] that is the inverse of the initial permutation function, to produce the 64-bit ciphertext. With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher, as shown in Figure 3.3.

The right-hand portion of Figure 3.5 shows the way in which the 56-bit key is used. Initially, the key is passed through a permutation function. Then, for each of the sixteen rounds, a *subkey* (K_i) is produced by the combination of a left

circular shift and a permutation. The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

6.2.1 Initial and Final Permutations

Figure 6.3 shows the initial and final permutations (P-boxes). Each of these permutations takes a 64-bit input and permutes them according to a predefined rule. We have shown only a few input ports and the corresponding output ports. These permutations are keyless straight permutations that are the inverse of each other. For example, in the initial permutation, the 58th bit in the input becomes the first bit in the output. Similarly, in the final permutation, the first bit in the input becomes the 58th bit in the output. In other words, if the rounds between these two permutations do not exist, the 58th bit entering the initial permutation is the same as the 58th bit leaving the final permutation.

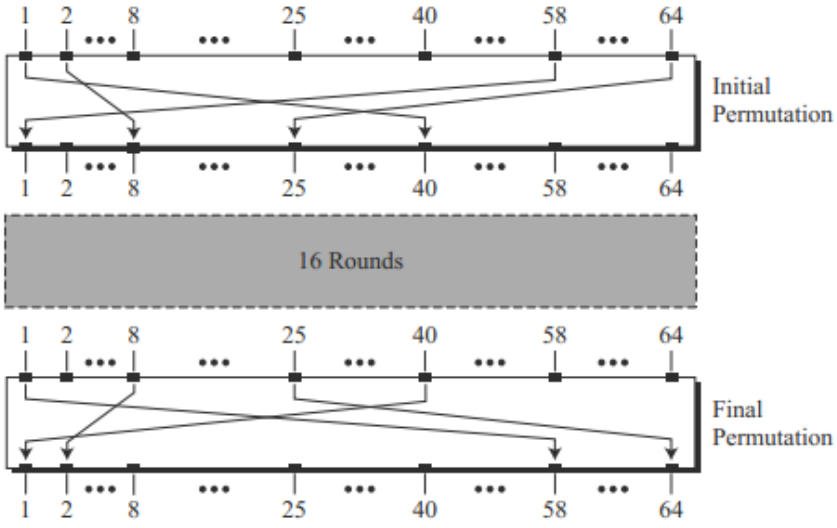


Fig. 6.3 *Initial and final permutation steps in DES*

The permutation rules for these P-boxes are shown in Table 6.1. Each side of the table can be thought of as a 64-element array. Note that, as with any permutation table we have discussed so far, the value of each element defines the input port number, and the order (index) of the element defines the output port number.

Table 6.1 *Initial and final permutation tables*

Initial Permutation	*Final Permutation*
58 50 42 34 26 18 10 02	40 08 48 16 56 24 64 32
60 52 44 36 28 20 12 04	39 07 47 15 55 23 63 31
62 54 46 38 30 22 14 06	38 06 46 14 54 22 62 30
64 56 48 40 32 24 16 08	37 05 45 13 53 21 61 29
57 49 41 33 25 17 09 01	36 04 44 12 52 20 60 28
59 51 43 35 27 19 11 03	35 03 43 11 51 19 59 27
61 53 45 37 29 21 13 05	34 02 42 10 50 18 58 26
63 55 47 39 31 23 15 07	33 01 41 09 49 17 57 25

These two permutations have no cryptography significance in DES. Both permutations are keyless and predetermined. The reason they are included in DES is not clear and has not been revealed by the DES designers. The guess is that DES was designed to be implemented in hardware (on chips) and that these two complex permutations may thwart a software simulation of the mechanism.

6.2.2 Rounds

DES uses 16 rounds. Each round of DES is a Feistel cipher, as shown in Fig. 6.4.

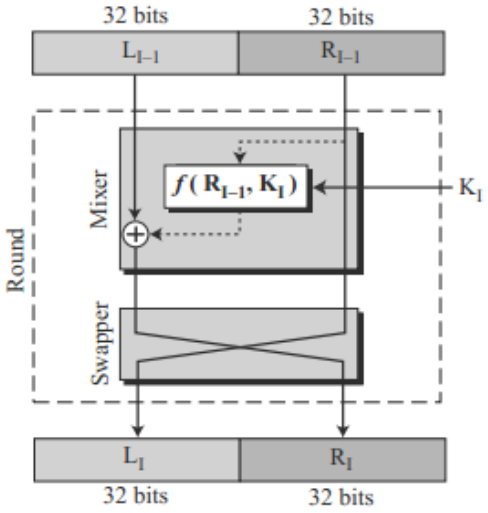
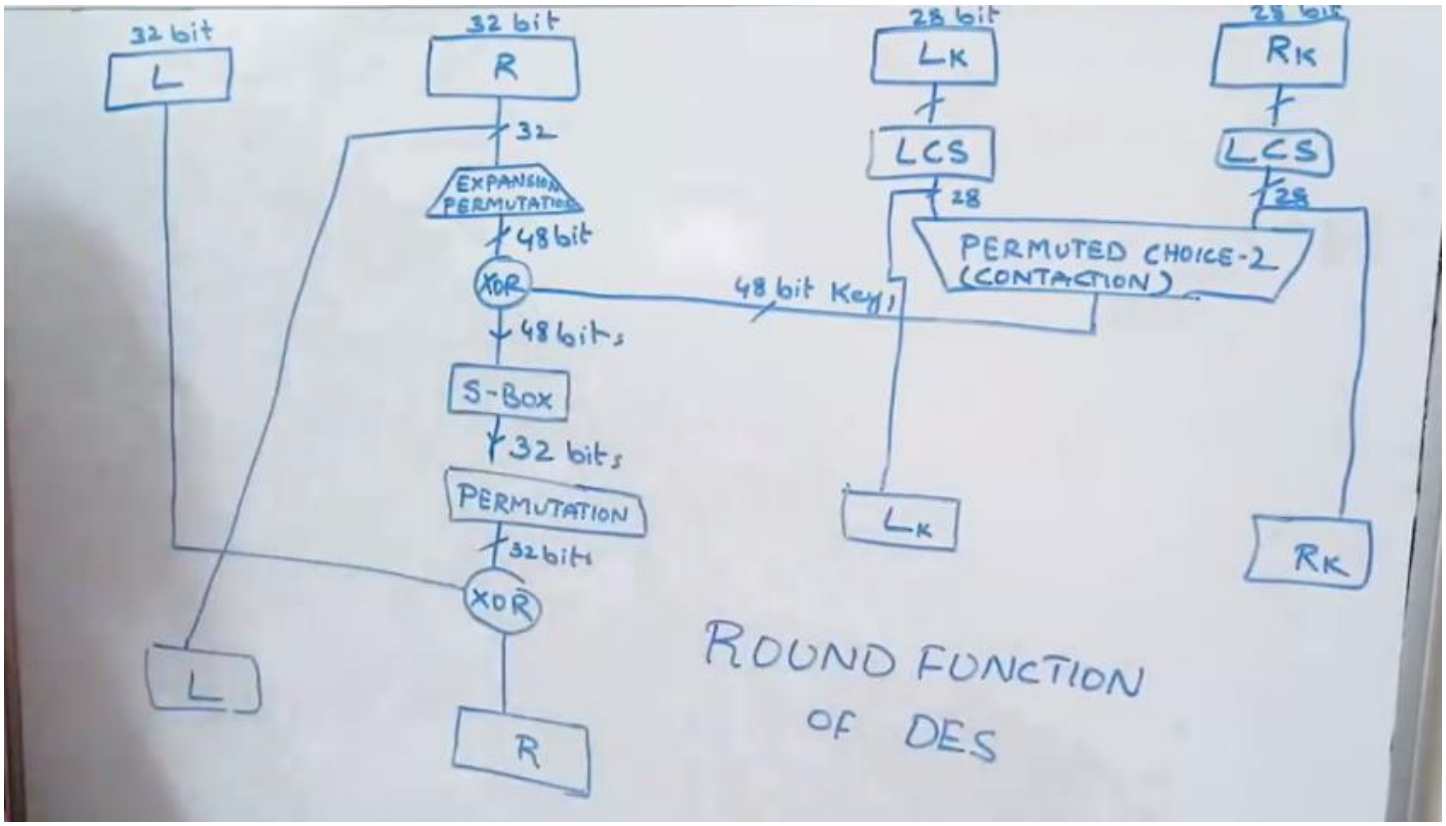


Fig. 6.4 *A round in DES (encryption site)*

The round takes L_{I-1} and R_{I-1} from previous round (or the initial permutation box) and creates L_I and R_I, which go to the next round (or final permutation box). As we discussed in Chapter 5, we can assume that each round has two cipher elements (mixer and swapper). Each of these elements is invertible. The swapper is obviously invertible. It swaps the left half of the text with the right half. The mixer is invertible because of the XOR operation. All noninvertible elements are collected inside the function $f(R_{I-1}, K_I)$.



DES Function

The heart of DES is the DES function. The DES function applies a 48-bit key to the rightmost 32 bits (R_{I-1}) to produce a 32-bit output. This function is made up of four sections: an expansion D-box, a whitener (that adds key), a group of S-boxes, and a straight D-box as shown in Fig. 6.5.

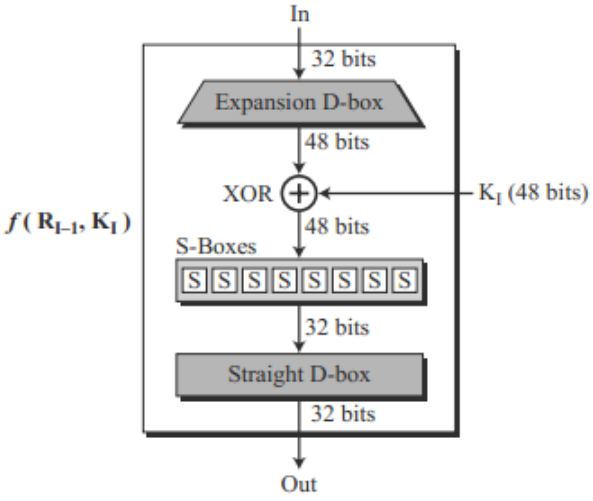


Fig. 6.5 *DES function*

Expansion D-box Since R_{I-1} is a 32-bit input and K_I is a 48-bit key, we first need to expand R_{I-1} to 48 bits. R_{I-1} is divided into 8 4-bit sections. Each 4-bit section is then expanded to 6 bits. This expansion permutation follows a predetermined rule. For each section, input bits 1, 2, 3, and 4 are copied to output bits 2, 3, 4, and 5, respectively. Output bit 1 comes from bit 4 of the previous section; output bit 6 comes from bit 1 of the next section. If sections 1 and 8 can be considered adjacent sections, the same rule applies to bits 1 and 32. Fig. 6.6 shows the input and output in the expansion permutation.

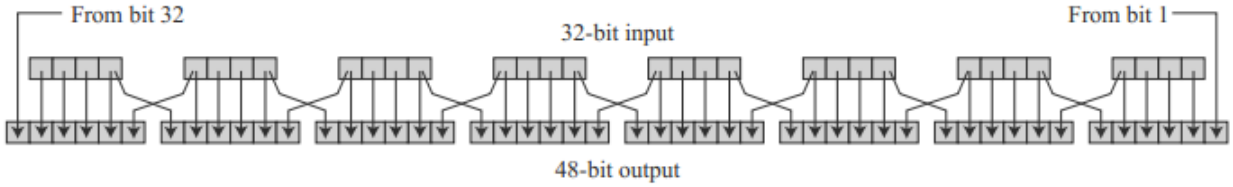


Fig. 6.6 *Expansion permutation*

Although the relationship between the input and output can be defined mathematically, DES uses Table 6.2 to define this D-box. Note that the number of output ports is 48, but the value range is only 1 to 32. Some of the inputs go to more than one output. For example, the value of input bit 5 becomes the value of output bits 6 and 8.

Table 6.2 *Expansion D-box table*

32	01	02	03	04	05
04	05	06	07	08	09
08	09	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	31	31	32	01

Whitener (XOR) After the expansion permutation, DES uses the XOR operation on the expanded right section and the round key. Note that both the right section and the key are 48-bits in length. Also note that the round key is used only in this operation.

S-Boxes The S-boxes do the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. See Fig. 6.7.

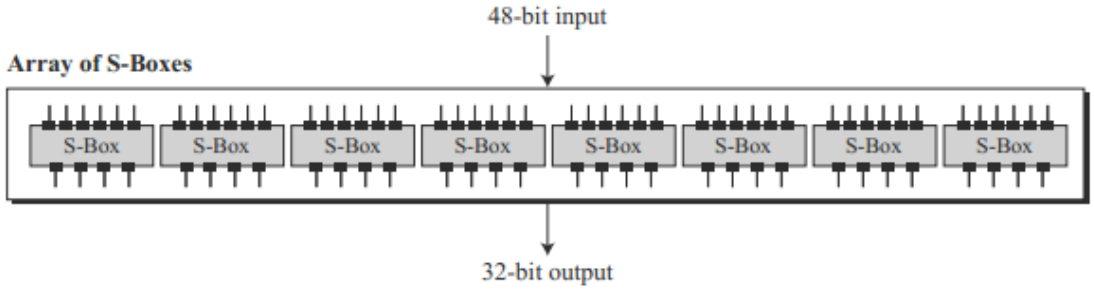


Fig. 6.7 *S-boxes*

The 48-bit data from the second operation is divided into eight 6-bit chunks, and each chunk is fed into a box. The result of each box is a 4-bit chunk; when these are combined the result is a 32-bit text. The substitution in each box follows a pre-determined rule based on a 4-row by 16-column table. The combination of bits 1 and 6 of the input defines one of four rows; the combination of bits 2 through 5 defines one of the sixteen columns as shown in Fig. 6.8. This will become clear in the examples.

Because each S-box has its own table, we need eight tables, as shown in Tables 6.3 to 6.10, to define the output of these boxes. The values of the inputs (row number and column number) and the values of the outputs are given as decimal numbers to save space. These need to be changed to binary.

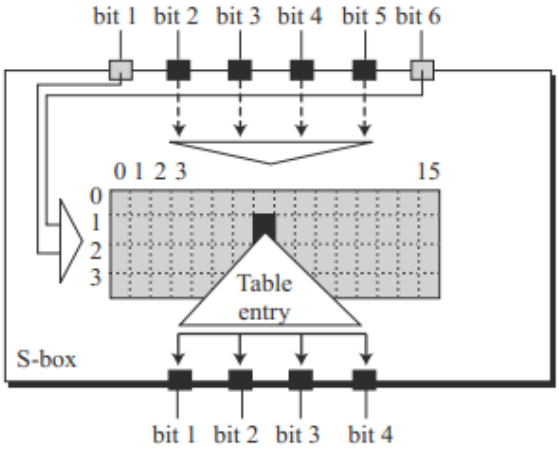


Fig. 6.8 *S-box rule*

Table 3.3 Definition of DES S-Boxes

S_1	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
S_2	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
S_3	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
S_4	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
S_5	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
S_6	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
S_7	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
S_8	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

Key Generation : The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key. However, the cipher key is normally given as a 64-bit key in which 8 extra bits are the parity bits, which are dropped before the actual key-generation process, as shown in Fig. 6.10.

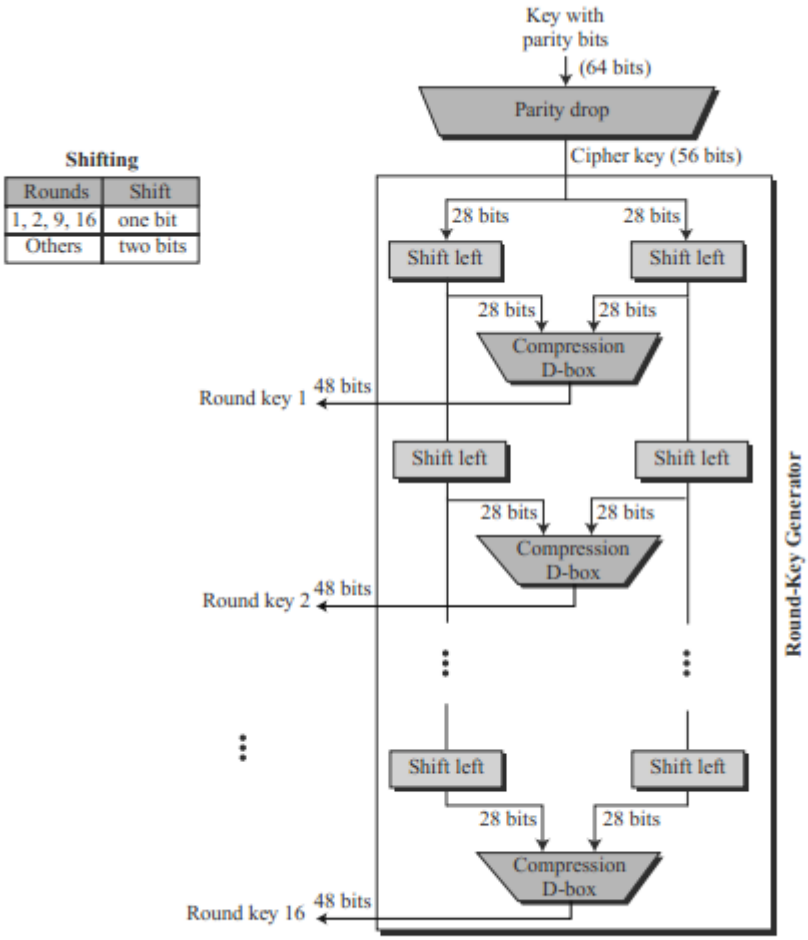


Fig. 6.10 *Key generation*

Parity Drop The preprocess before key expansion is a compression transposition step that we call **parity bit drop.** It drops the parity bits (bits 8, 16, 24, 32, …, 64) from the 64-bit key and permutes the rest of the bits according to Table 6.12. The remaining 56-bit value is the actual cipher key which is used to generate round keys. The parity drop step (a compression D-box) is shown in Table 6.12.

Table 6.12 *Parity-bit drop table*

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

Shift Left After the straight permutation, the key is divided into two 28-bit parts. Each part is shifted left (circular shift) one or two bits. In rounds 1, 2, 9, and 16, shifting is one bit; in the other rounds, it is two bits. The two parts are then combined to form a 56-bit part. Table 6.13 shows the number of shifts for each round.

Table 6.13 *Number of bit shifts*

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Compression D-box The compression D-box changes the 58 bits to 48 bits, which are used as a key for a round. The compression step is shown in Table 6.14.

Table 6.14 *Key-compression table*

14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Advantages:

1. its a 56 bit key. So there are 2^56 possibilities of keys which would take a decade to find the correct key using brute-force attack
2. Encryption and decryption takes the same algorithm. Only that the function need to be reversed and the key should be taken in opposite order. This is very convenient for software and hardware requirements.

Disadvantages:(I would say small weaknesses!)

1. Weak keys : the key that is selected on the rounds are a problem . During splitting of keys to two half and swapping them might throw up the same result if they have continuous 1's and 0's. Thins ends up in using the same key through out the 16-cycles
2. There can be same output from the S-Boxes on different inputs on permutation. These are called Semi weak keys.
3. If the message is encrypted with a particular key, and is taken 1's compliment of that encryption will be same as that of the encryption of the compliment message and compliment key.

Multiple encryption is a technique in which an encryption algorithm is used multiple times. In the first instance, plaintext is converted to ciphertext using the encryption algorithm. This ciphertext is then used as input and the algorithm is applied again. This process may be repeated through any number of stages.

Double DES and Triple DES

Double DES

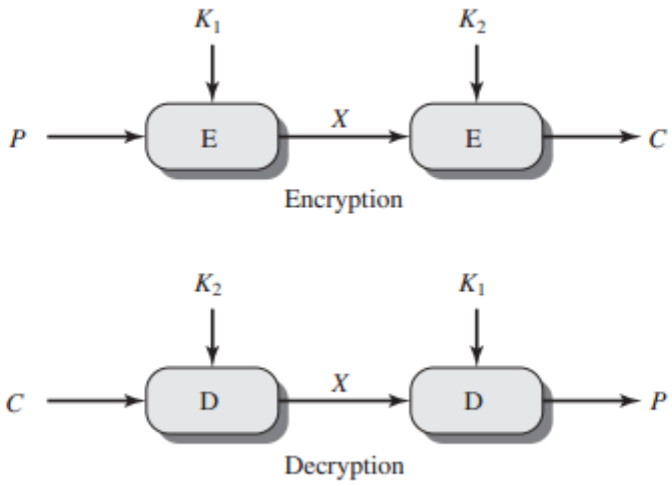
The simplest form of multiple encryption has two encryption stages and two keys (Figure 6.1a). Given a plaintext P and two encryption keys K_1 and K_2, ciphertext C is generated as

$$C = \mathrm{E}(K_2, \mathrm{E}(K_1, P))$$

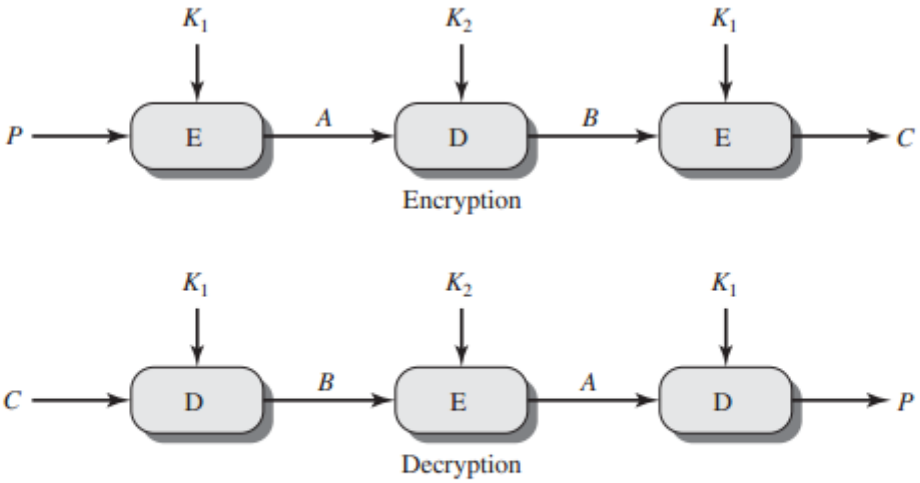
Decryption requires that the keys be applied in reverse order:

$$P = \mathrm{D}(K_1, \mathrm{D}(K_2, C))$$

For DES, this scheme apparently involves a key length of $56 \times 2 = 112$ bits, resulting in a dramatic increase in cryptographic strength. But we need to examine the algorithm more closely.

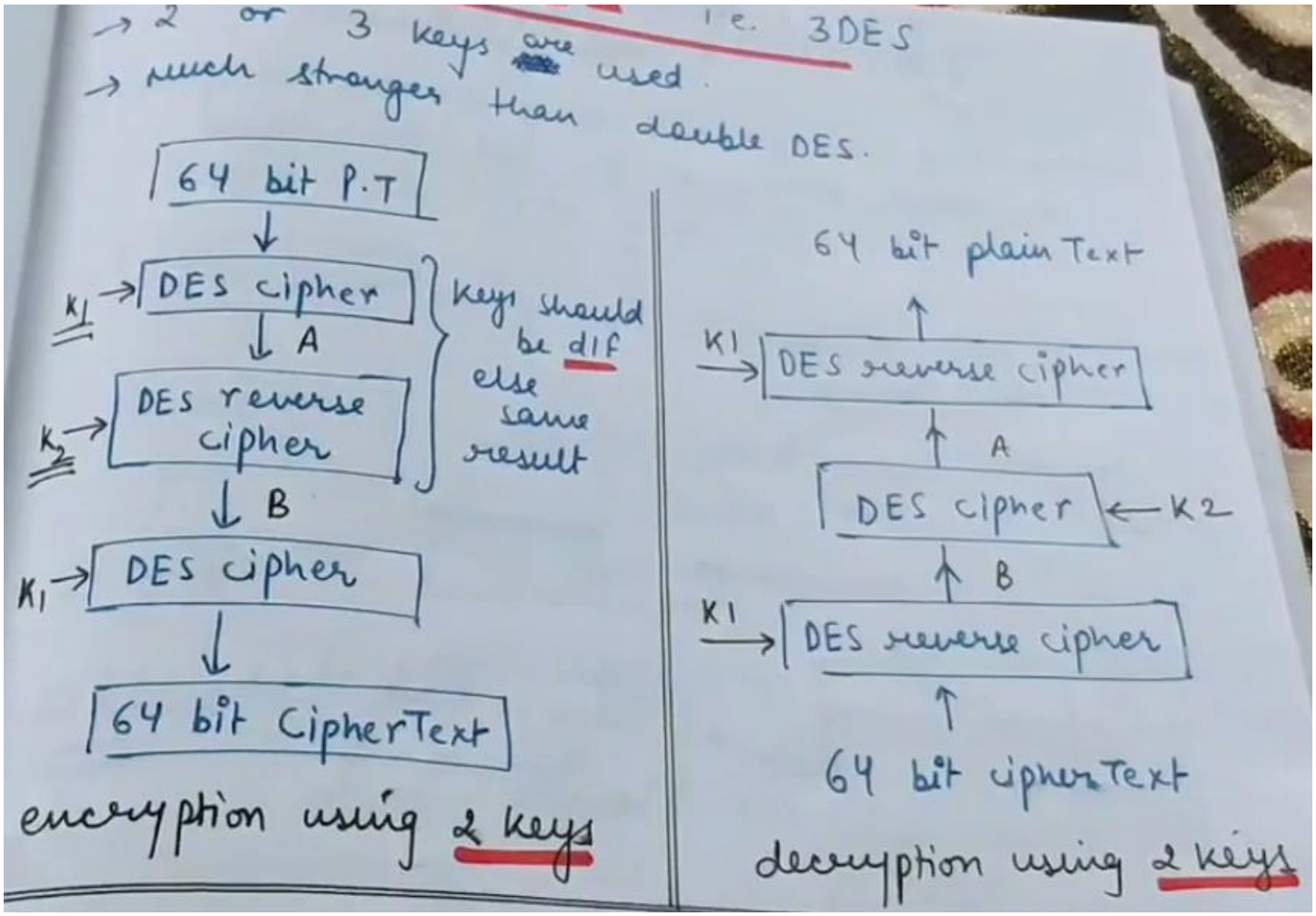


(a) Double encryption



(b) Triple encryption

Figure 6.1 Multiple Encryption



Triple DES with Two Keys

An obvious counter to the meet-in-the-middle attack is to use three stages of encryption with three different keys. This raises the cost of the meet-in-the-middle attack to 2^{112}, which is beyond what is practical now and far into the future. However, it has the drawback of requiring a key length of $56 \times 3 = 168$ bits, which may be somewhat unwieldy.

As an alternative, Tuchman proposed a triple encryption method that uses only two keys [TUCH79]. The function follows an encrypt-decrypt-encrypt (EDE) sequence (Figure 6.1b):

$$C = E(K_1, D(K_2, E(K_1, P)))$$

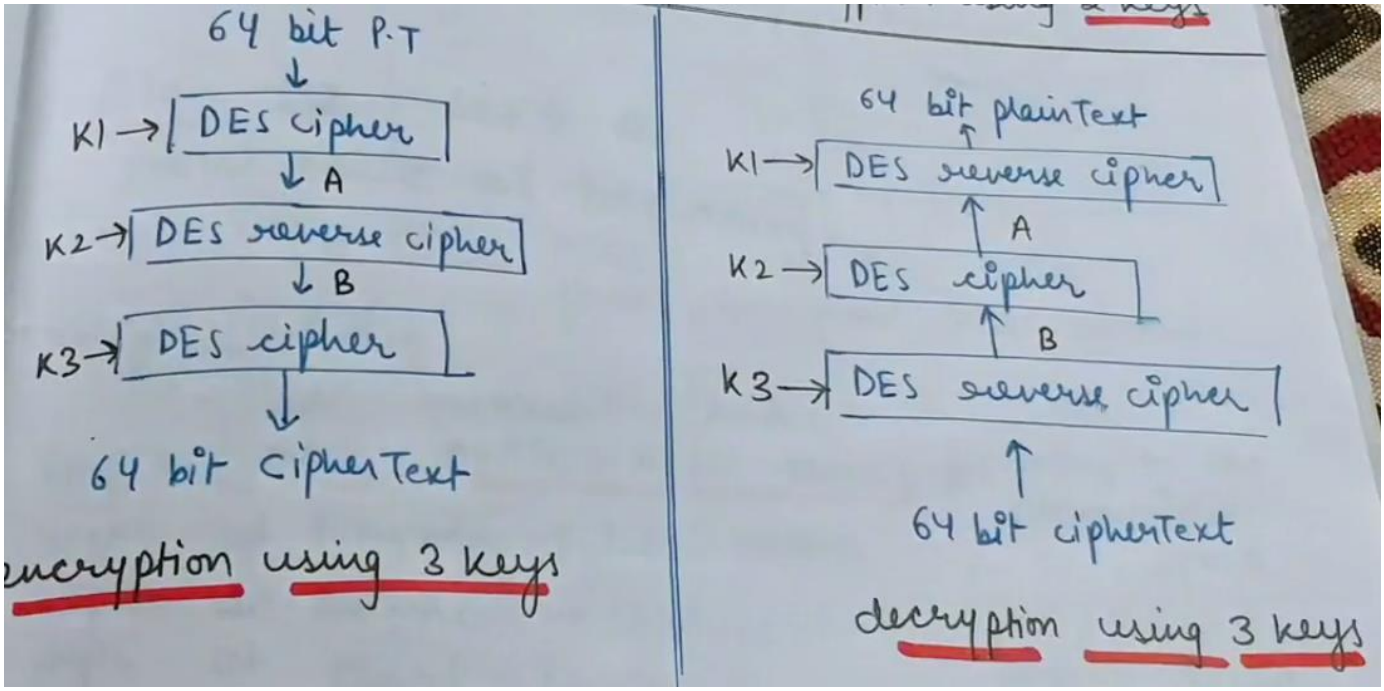
$$P = D(K_1, E(K_2, D(K_1, C)))$$

There is no cryptographic significance to the use of decryption for the second stage. Its only advantage is that it allows users of 3DES to decrypt data encrypted by users of the older single DES:

$$C = E(K_1, D(K_1, E(K_1, P))) = E(K_1, P)$$

$$P = D(K_1, E(K_1, D(K_1, C))) = D(K_1, C)$$

3DES with two keys is a relatively popular alternative to DES and has been adopted for use in the key management standards ANS X9.17 and ISO 8732.[1]



Modes of Block Cipher

Table 6.1 Block Cipher Modes of Operation

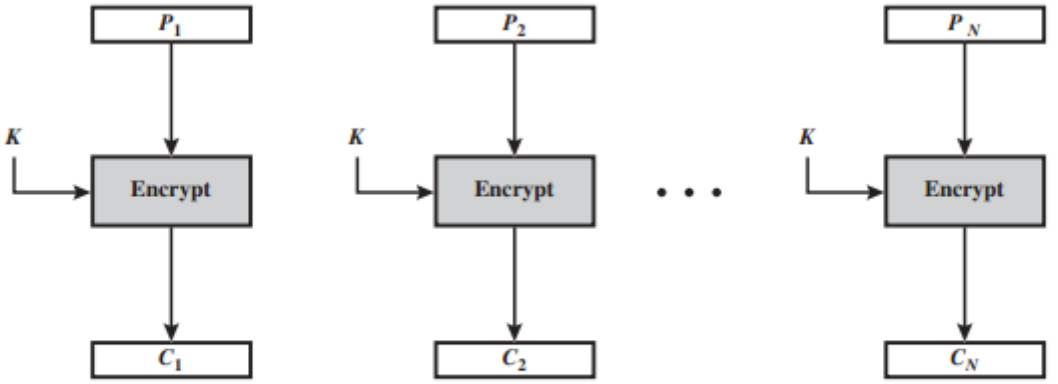
Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of 64 plaintext bits is encoded independently using the same key.	• Secure transmission of single values (e.g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of plaintext and the preceding 64 bits of ciphertext.	• General-purpose block-oriented transmission • Authentication
Cipher Feedback (CFB)	Input is processed s bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	• General-purpose stream-oriented transmission • Authentication
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding encryption output, and full blocks are used.	• Stream-oriented transmission over noisy channel (e.g., satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	• General-purpose block-oriented transmission • Useful for high-speed requirements

ELECTRONIC CODE BOOK:

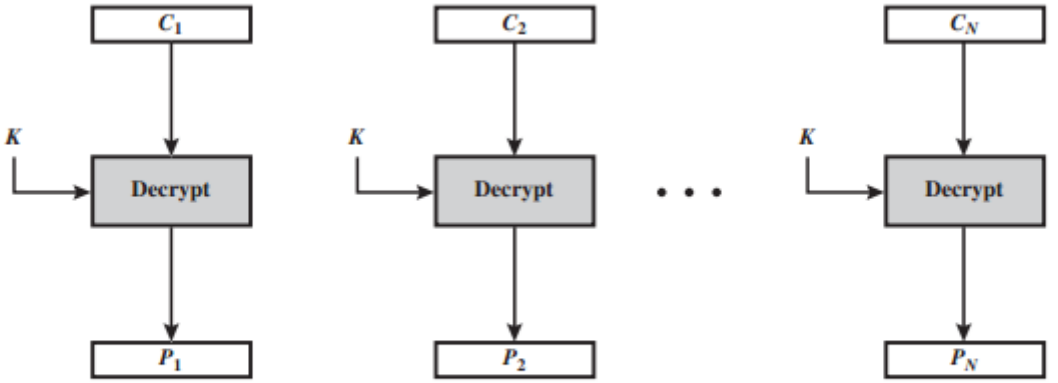
A block cipher takes a fixed-length block of text of length b bits and a key as input and produces a b-bit block of ciphertext. If the amount of plaintext to be encrypted is greater than b bits, then the block cipher can still be used by breaking the plaintext

The simplest mode is the **electronic codebook (ECB)** mode, in which plaintext is handled one block at a time and each block of plaintext is encrypted using the same key (Figure 6.3). The term *codebook* is used because, for a given key, there is a unique ciphertext for every b-bit block of plaintext. Therefore, we can imagine a gigantic codebook in which there is an entry for every possible b-bit plaintext pattern showing its corresponding ciphertext.

For a message longer than b bits, the procedure is simply to break the message into b-bit blocks, padding the last block if necessary. Decryption is performed one block at a time, always using the same key. In Figure 6.3, the plaintext (padded as necessary) consists of a sequence of b-bit blocks, $P_1, P_2, \ldots, P_N$; the



(a) Encryption



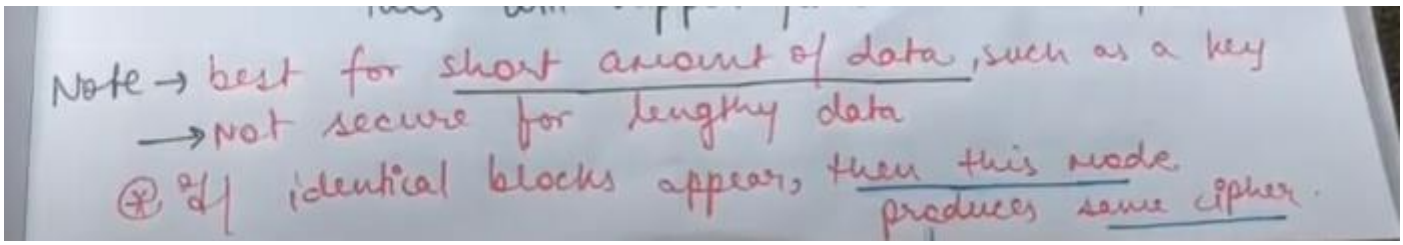
(b) Decryption

Figure 6.3 Electronic Codebook (ECB) Mode

corresponding sequence of ciphertext blocks is $C_1, C_2, \ldots, C_N$. We can define ECB mode as follows.

ECB	$C_j = E(K, P_j)$ $j = 1, \ldots, N$	$P_j = D(K, C_j)$ $j = 1, \ldots, N$

The ECB method is ideal for a short amount of data, such as an encryption key. Thus, if you want to transmit a DES or AES key securely, ECB is the appropriate mode to use.

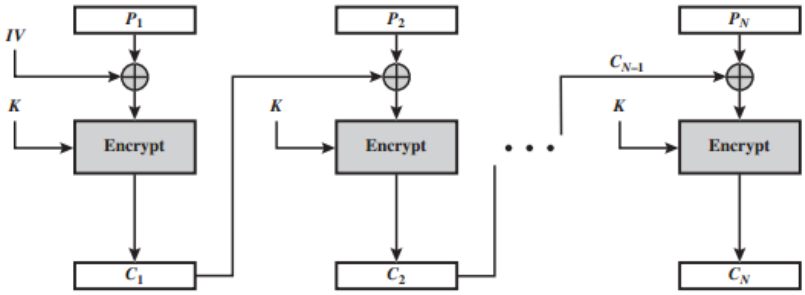


6.3 CIPHER BLOCK CHAINING MODE

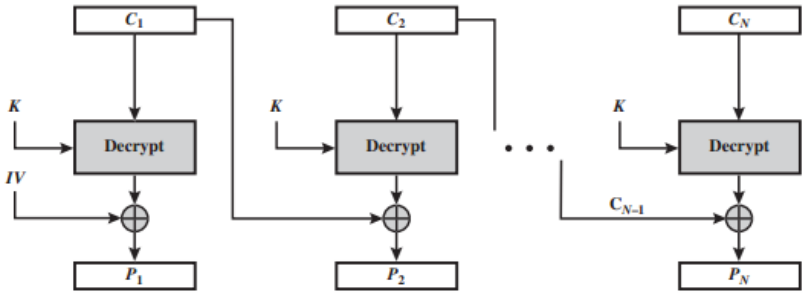
To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks. A simple way to satisfy this requirement is the **cipher block chaining** (**CBC**) mode (Figure 6.4). In this scheme, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block. In effect, we have chained together the processing of the sequence of plaintext blocks. The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating patterns of b bits are not exposed. As with the ECB mode, the CBC mode requires that the last block be padded to a full b bits if it is a partial block.

For decryption, each cipher block is passed through the decryption algorithm. The result is XORed with the preceding ciphertext block to produce the plaintext block. To see that this works, we can write

$$C_j = \mathrm{E}(K, [C_{j-1} \oplus P_j])$$



(a) Encryption



(b) Decryption

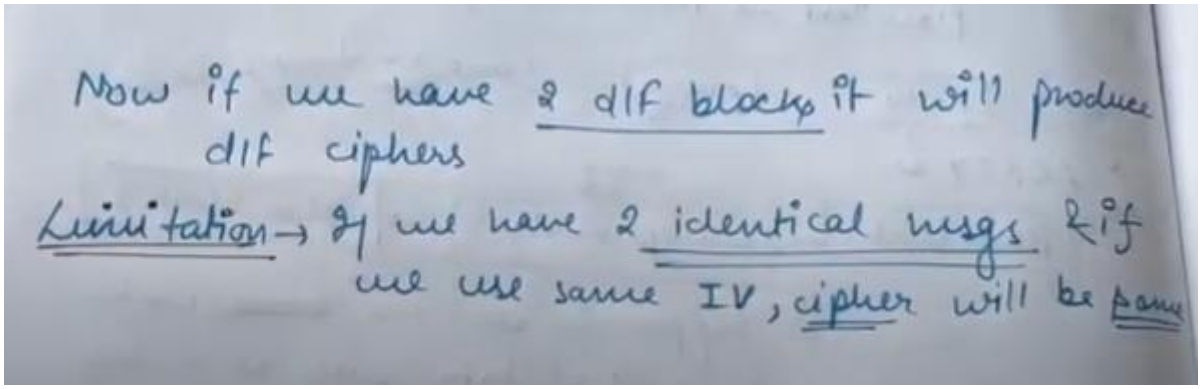
Figure 6.4 Cipher Block Chaining (CFB) Mode

Then

$$D(K, C_j) = D(K, E(K, [C_{j-1} \oplus P_j]))$$
$$D(K, C_j) = C_{j-1} \oplus P_j$$
$$C_{j-1} \oplus D(K, C_j) = C_{j-1} \oplus C_{j-1} \oplus P_j = P_j$$

To produce the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext. On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext. The IV is a data block that is that same size as the cipher block. We can define CBC mode as

CBC	$C_1 = E(K, [P_1 \oplus IV])$ $C_j = E(K, [P_j \oplus C_{j-1}])$ $j = 2, \ldots, N$	$P_1 = D(K, C_1) \oplus IV$ $P_j = D(K, C_j) \oplus C_{j-1}$ $j = 2, \ldots, N$



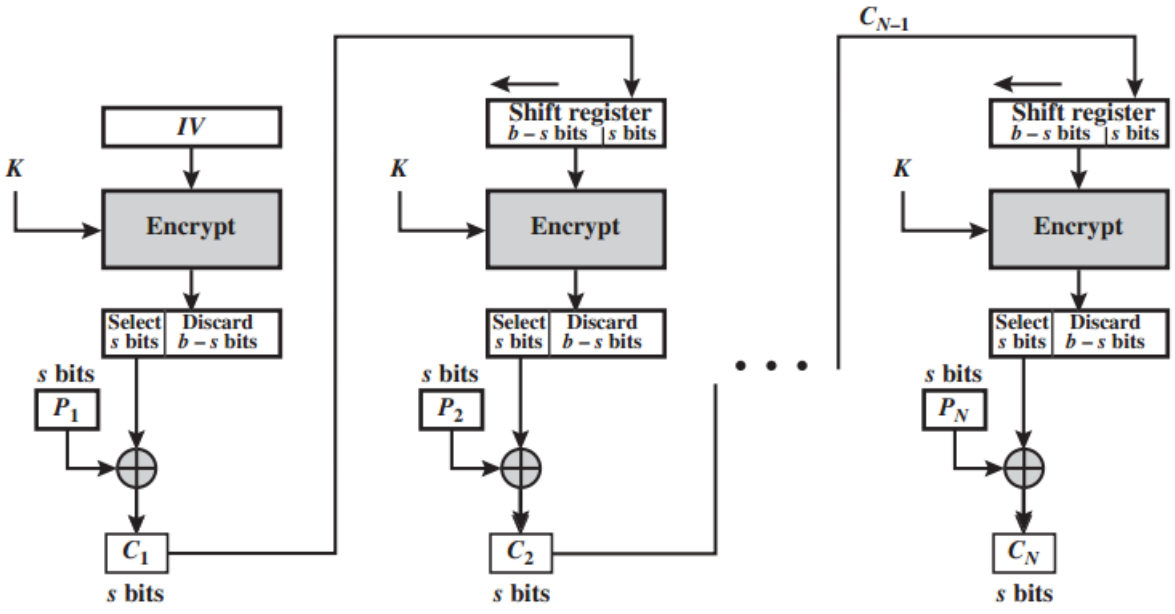
6.4 CIPHER FEEDBACK MODE

For AES, DES, or any block cipher, encryption is performed on a block of b bits. In the case of DES, $b = 64$ and in the case of AES, $b = 128$. However, it is possible to convert a block cipher into a stream cipher, using one of the three modes to be discussed in this and the next two sections: **cipher feedback** (CFB) mode, **output feedback** (OFB) mode, and **counter** (CTR) mode. A stream cipher eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time. Thus, if a character stream is being transmitted, each character can be encrypted and transmitted immediately using a character-oriented stream cipher.

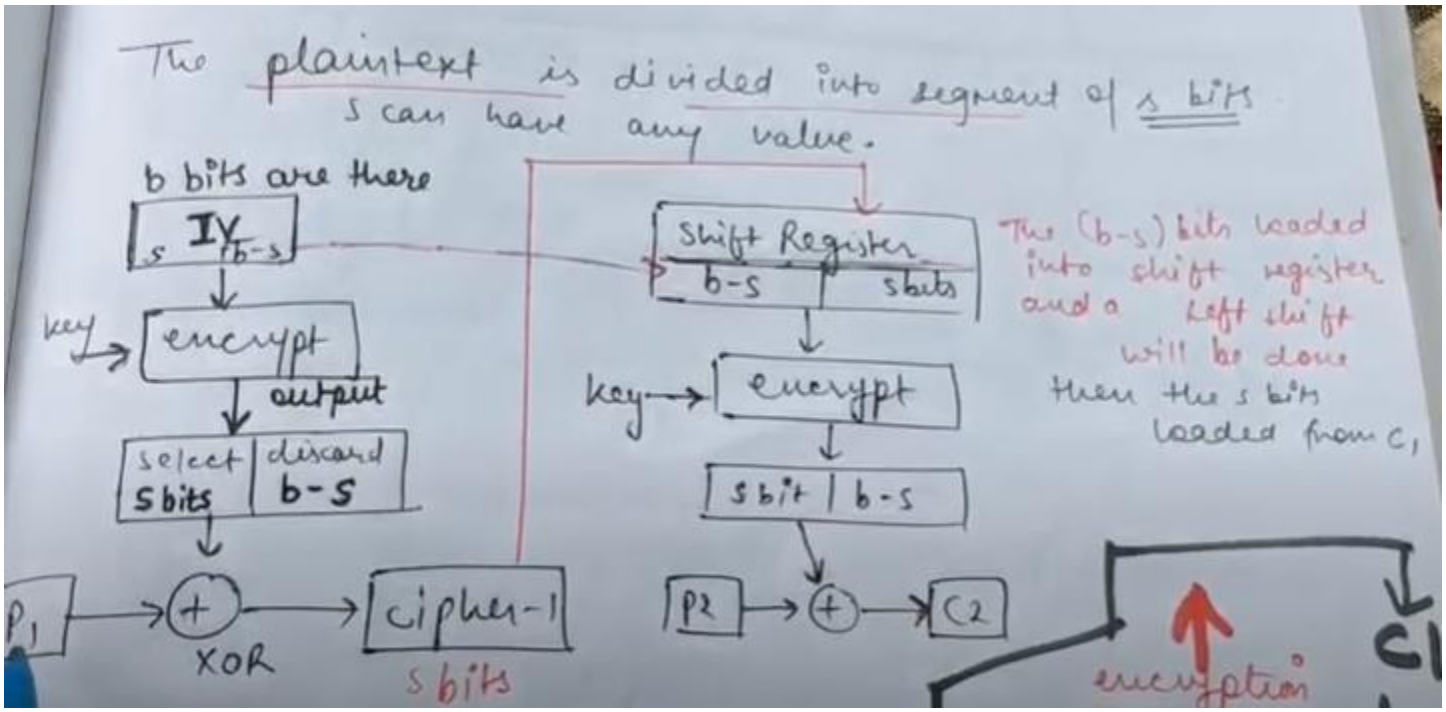
One desirable property of a stream cipher is that the ciphertext be of the same length as the plaintext. Thus, if 8-bit characters are being transmitted, each character should be encrypted to produce a ciphertext output of 8 bits. If more than 8 bits are produced, transmission capacity is wasted.

Figure 6.5 depicts the CFB scheme. In the figure, it is assumed that the unit of transmission is s bits; a common value is $s = 8$. As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext. In this case, rather than blocks of b bits, the plaintext is divided into *segments* of s bits.

First, consider encryption. The input to the encryption function is a b-bit shift register that is initially set to some initialization vector (IV). The leftmost (most significant) s bits of the output of the encryption function are XORed with the first segment of plaintext P_1 to produce the first unit of ciphertext C_1, which is then transmitted. In addition, the contents of the shift register are shifted left by s bits, and C_1 is placed in the rightmost (least significant) s bits of the shift register. This process continues until all plaintext units have been encrypted.



(a) Encryption



For decryption, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit. Note that it is the *encryption* function that is used, not the decryption function. This is easily explained. Let $\text{MSB}_s(X)$ be defined as the most significant s bits of X. Then

$$C_1 = P_1 \oplus \text{MSB}_s[\text{E}(K, \text{IV})]$$

Therefore, by rearranging terms:

$$P_1 = C_1 \oplus \text{MSB}_s[\text{E}(K, \text{IV})]$$

The same reasoning holds for subsequent steps in the process.

We can define CFB mode as follows.

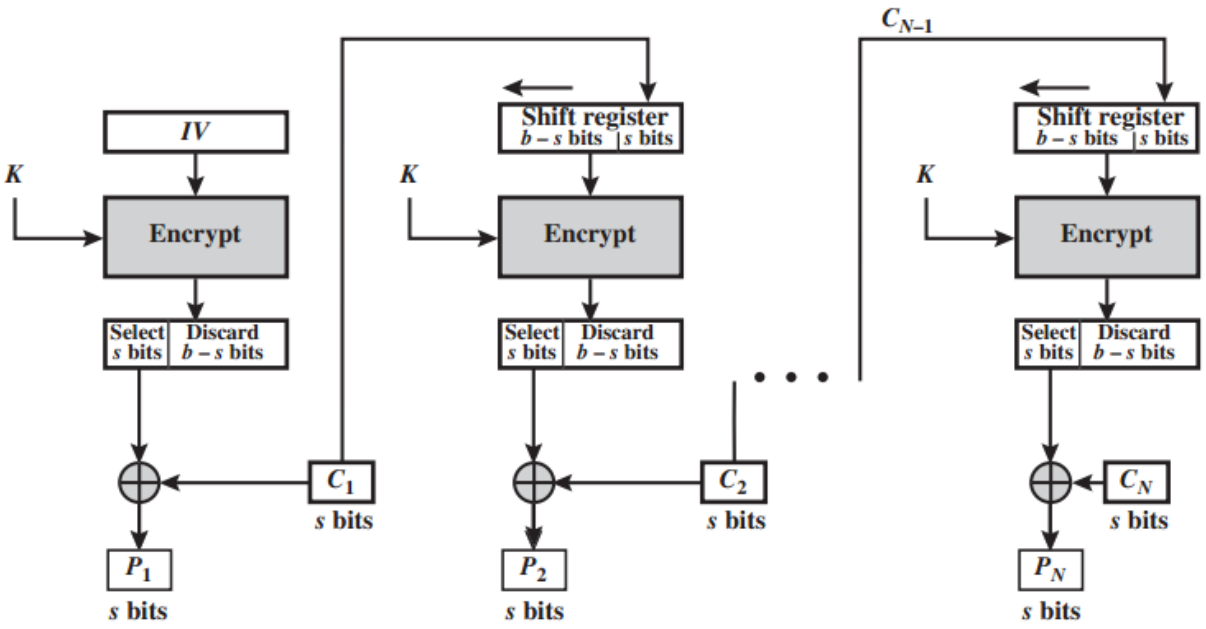


Figure 6.5 *s*-bit Cipher Feedback (CFB) Mode

CFB	$I_1 = IV$ $I_j = \text{LSB}_{b-s}(I_{j-1}) \parallel C_{j-1} \quad j = 2, \ldots, N$ $O_j = \text{E}(K, I_j) \quad j = 1, \ldots, N$ $C_j = P_j \oplus \text{MSB}_s(O_j) \quad j = 1, \ldots, N$	$I_1 = IV$ $I_j = \text{LSB}_{b-s}(I_{j-1}) \parallel C_{j-1} \quad j = 2, \ldots, N$ $O_j = \text{E}(K, I_j) \quad j = 1, \ldots, N$ $P_j = C_j \oplus \text{MSB}_s(O_j) \quad j = 1, \ldots, N$

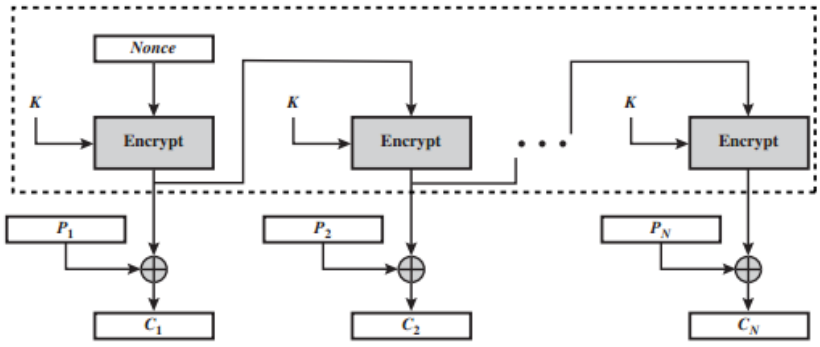
6.5 OUTPUT FEEDBACK MODE

The **output feedback** (OFB) mode is similar in structure to that of CFB. As can be seen in Figure 6.6, it is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB, the ciphertext unit is fed back to the shift register. The other difference is that the OFB mode operates on full blocks of plaintext and ciphertext, not on an *s*-bit subset. Encryption can be expressed as

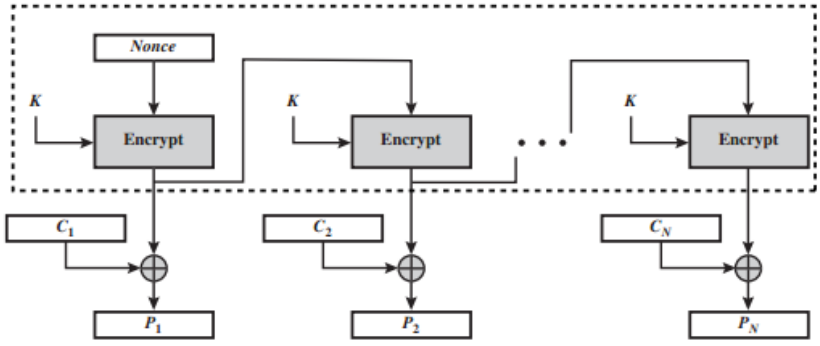
$$C_j = P_j \oplus \text{E}(K, [C_{j-i} \oplus P_{j-1}])$$

By rearranging terms, we can demonstrate that decryption works.

$$P_j = C_j \oplus \text{E}(K, [C_{j-1} \oplus P_{j-1}])$$

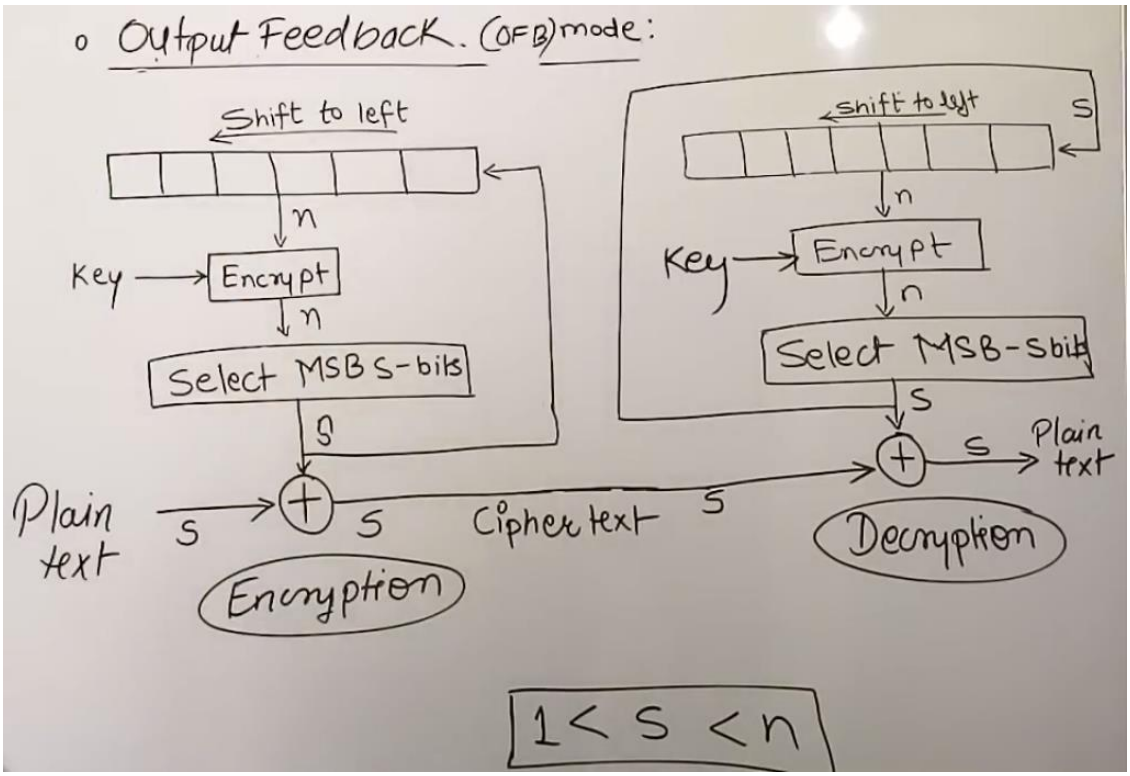


(a) Encryption



(b) Decryption

Figure 6.6 Output Feedback (OFB) Mode

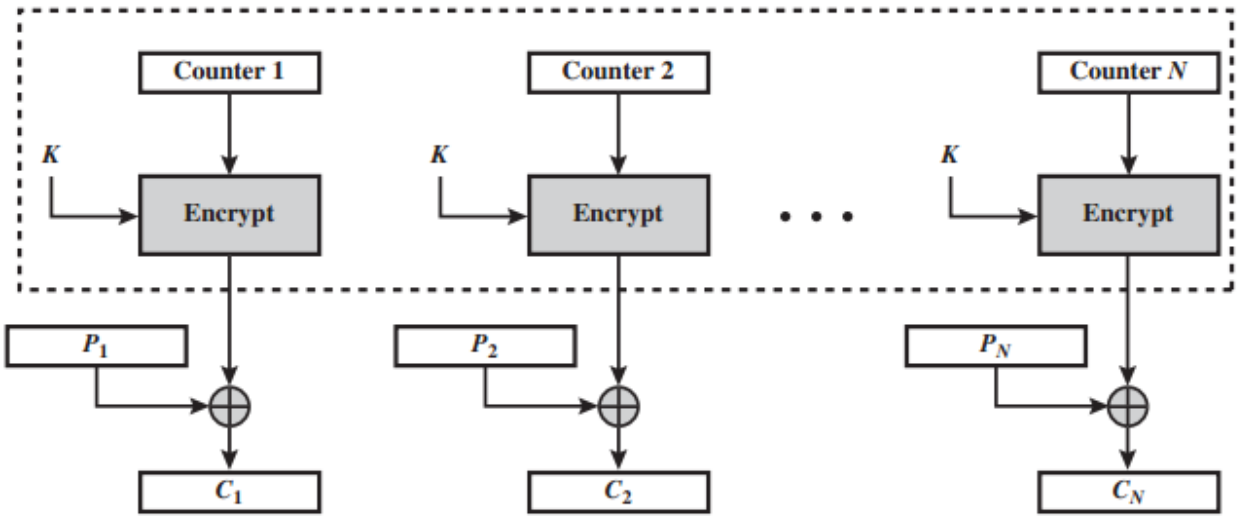


6.6 COUNTER MODE

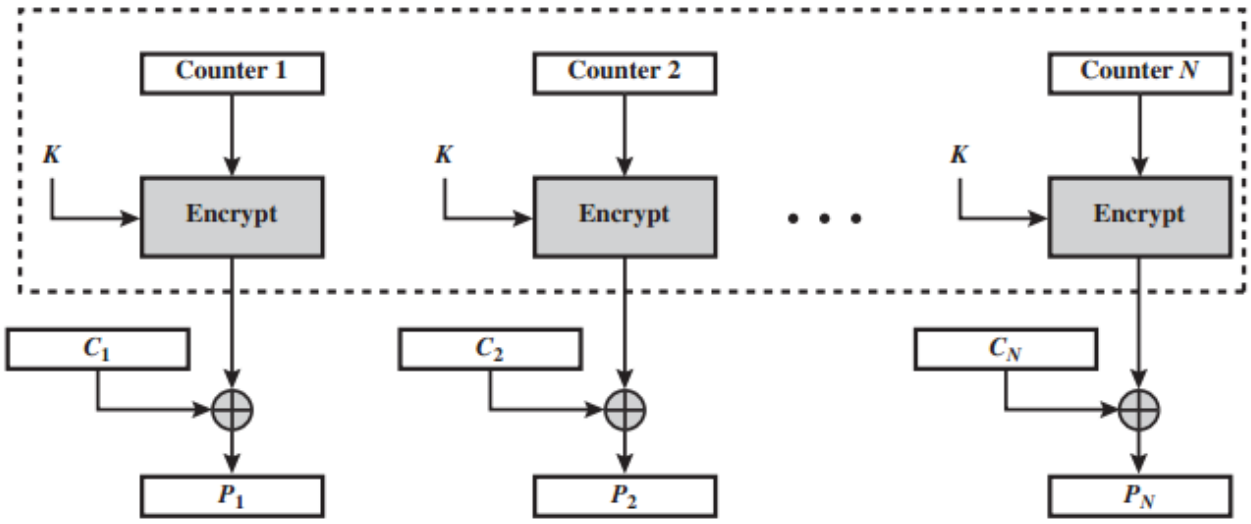
Although interest in the **counter** (CTR) mode has increased recently with applications to ATM (asynchronous transfer mode) network security and IP sec (IP security), this mode was proposed early on (e.g., [DIFF79]).

Figure 6.7 depicts the CTR mode. A counter equal to the plaintext block size is used. The only requirement stated in SP 800-38A is that the counter value must be different for each plaintext block that is encrypted. Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo 2^b, where b is the block size). For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining. For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block. Thus, the initial counter value must be made available for decryption. Given a sequence of counters $T_1, T_2, \ldots, T_N$, we can define CTR mode as follows.

CTR	$C_j = P_j \oplus \mathrm{E}(K, T_j) \quad j = 1, \ldots, N-1$ $C_N^* = P_N^* \oplus \mathrm{MSB}_u[\mathrm{E}(K, T_N)]$	$P_j = C_j \oplus \mathrm{E}(K, T_j) \quad j = 1, \ldots, N-1$ $P_N^* = C_N^* \oplus \mathrm{MSB}_u[\mathrm{E}(K, T_N)]$



(a) Encryption



(b) Decryption

Figure 6.7 Counter (CTR) Mode

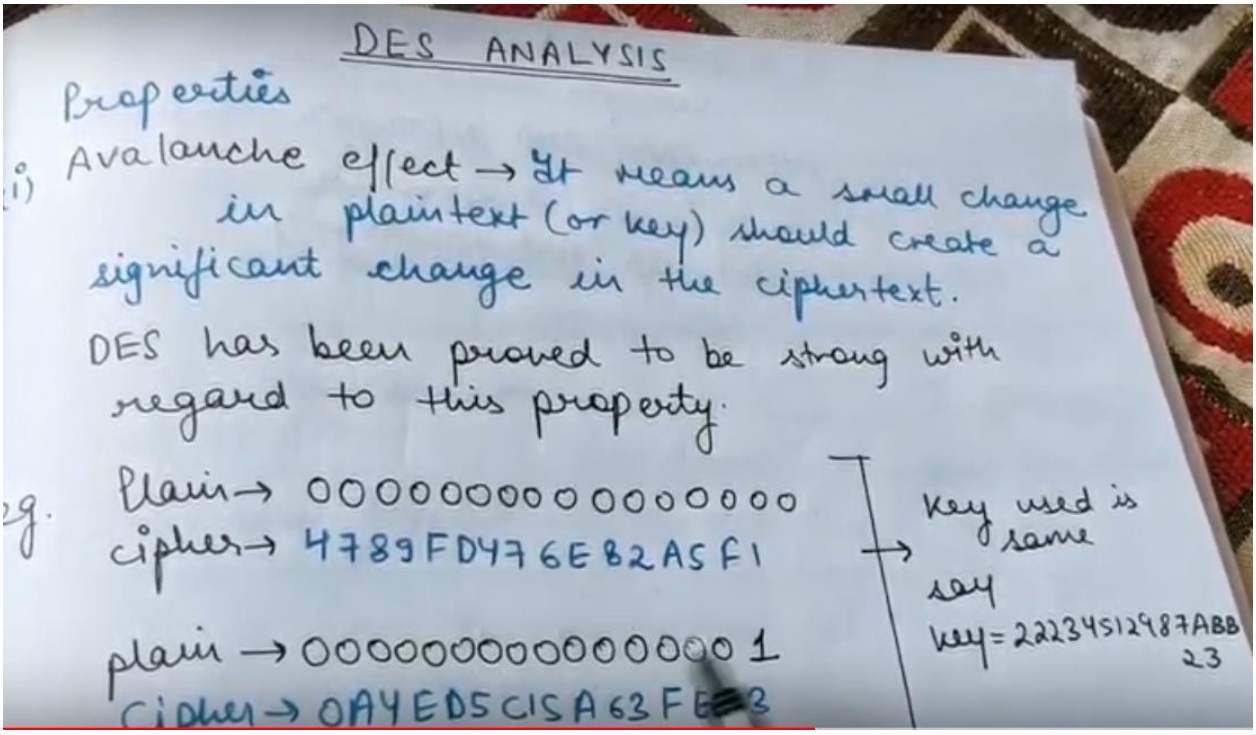
different for each plaintext block that is encrypted. Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo 2^b, where b is the block size). For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining. For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block. Thus, the initial counter value must be made available for decryption. Given a sequence of counters $T_1, T_2, \ldots, T_N$, we can define CTR mode as follows.

CTR	$C_j = P_j \oplus \mathrm{E}(K, T_j) \quad j = 1, \ldots, N-1$ $C_N^* = P_N^* \oplus \mathrm{MSB}_u[\mathrm{E}(K, T_N)]$	$P_j = C_j \oplus \mathrm{E}(K, T_j) \quad j = 1, \ldots, N-1$ $P_N^* = C_N^* \oplus \mathrm{MSB}_u[\mathrm{E}(K, T_N)]$

[LIPM00] lists the following advantages of CTR mode.

- **Hardware efficiency:** Unlike the three chaining modes, encryption (or decryption) in CTR mode can be done in parallel on multiple blocks of plaintext or ciphertext. For the chaining modes, the algorithm must complete the computation on one block before beginning on the next block. This limits the maximum throughput of the algorithm to the reciprocal of the time for one execution of block encryption or decryption. In CTR mode, the throughput is only limited by the amount of parallelism that is achieved.

- **Software efficiency:** Similarly, because of the opportunities for parallel execution in CTR mode, processors that support parallel features, such as aggressive pipelining, multiple instruction dispatch per clock cycle, a large number of registers, and SIMD instructions, can be effectively utilized.
- **Preprocessing:** The execution of the underlying encryption algorithm does not depend on input of the plaintext or ciphertext. Therefore, if sufficient memory is available and security is maintained, preprocessing can be used to prepare the output of the encryption boxes that feed into the XOR functions, as in Figure 6.7. When the plaintext or ciphertext input is presented, then the only computation is a series of XORs. Such a strategy greatly enhances throughput.
- **Random access:** The *i*th block of plaintext or ciphertext can be processed in random-access fashion. With the chaining modes, block C_i cannot be computed until the $i - 1$ prior block are computed. There may be applications in which a ciphertext is stored and it is desired to decrypt just one block; for such applications, the random access feature is attractive.
- **Provable security:** It can be shown that CTR is at least as secure as the other modes discussed in this section.
- **Simplicity:** Unlike ECB and CBC modes, CTR mode requires only the implementation of the encryption algorithm and not the decryption algorithm. This matters most when the decryption algorithm differs substantially from the encryption algorithm, as it does for AES. In addition, the decryption key scheduling need not be implemented.



Although, the two plaintexts differ only in 1 bit, ciphertext block differs a lot/significantly.

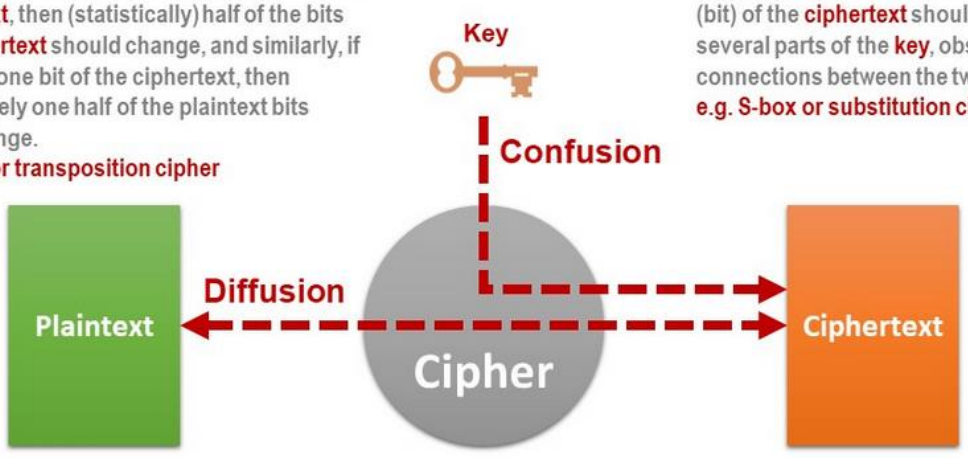
2. Completeness effect → It means that each bit of the ciphertext needs to depend on many bits on the plaintext.

The confusion and diffusion produced by D-boxes and S-boxes in DES, show a very strong completeness effect.

Confusion and Diffusion

Diffusion means that if we change a single bit of the **plaintext**, then (statistically) half of the bits in the **ciphertext** should change, and similarly, if we change one bit of the ciphertext, then approximately one half of the plaintext bits should change.
e.g. P-box or transposition cipher

Confusion means that each binary digit (bit) of the **ciphertext** should depend on several parts of the **key**, obscuring the connections between the two.
e.g. S-box or substitution cipher



Shannon's Theory of confusion and Diffusion

1) The terms confusion and diffusion were introduced by Claude Shannon.

2) Shannon's concern was to prevent cryptanalysis, based on statistical analysis. The reason is as follows:

Assume attacker has some knowledge of the statistical characteristics of the plaintext (eg in a msg, the frequency distribution of the various letters may be known.
If these statistics are in any way reflected in the ciphertext, the cryptanalyst i.e attacker may be able to deduce the encryption key.

Thus Shannon suggested 2 methods for frustating the attackers:

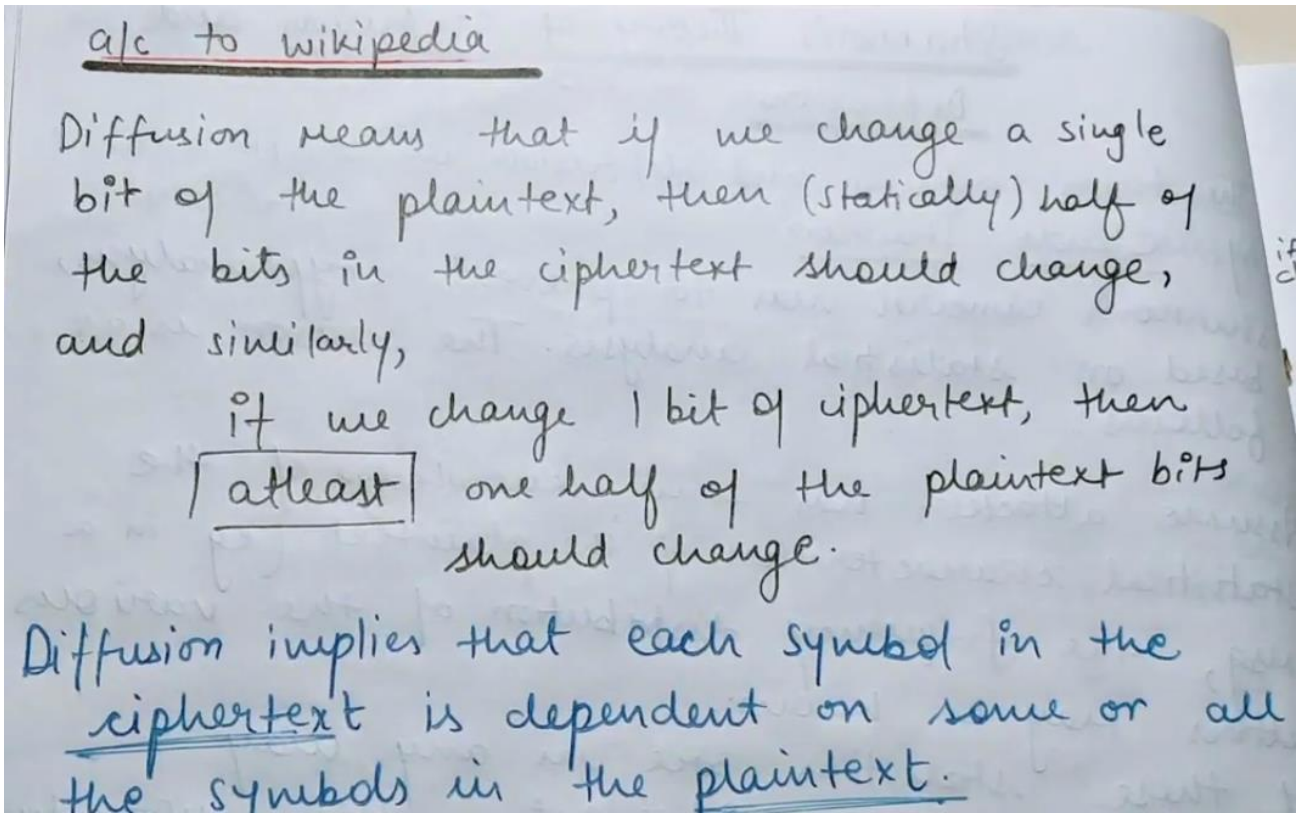
1) confusion
2) Diffusion

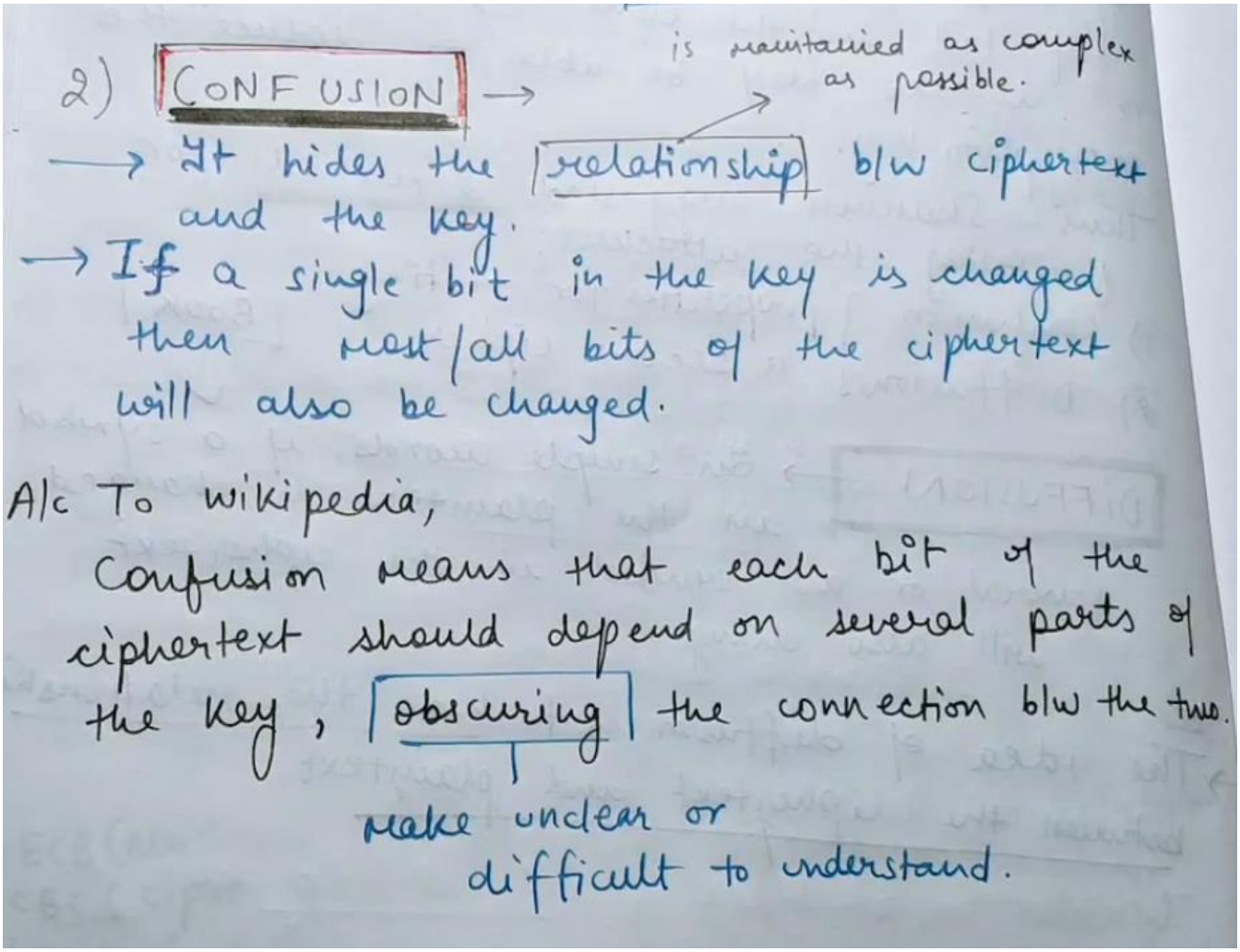
} properties for creating a secure cipher

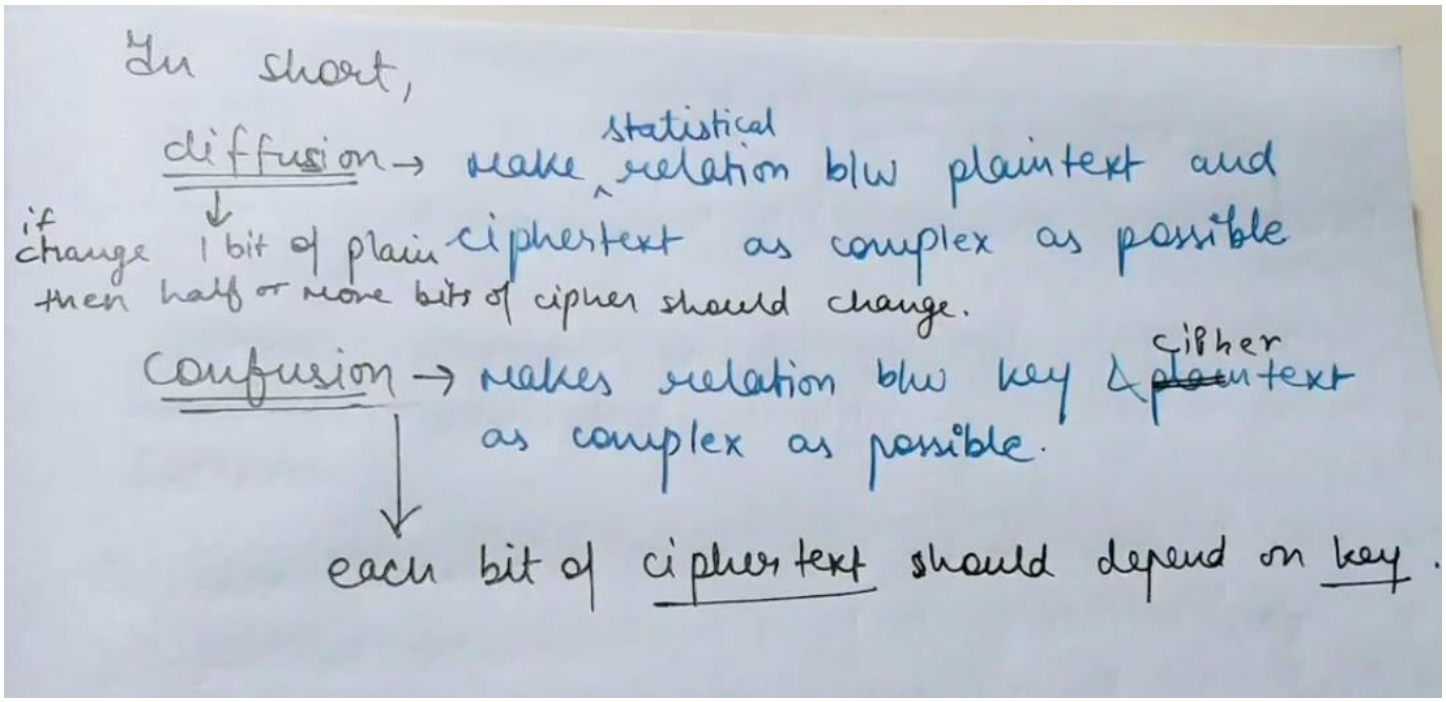
DIFFUSION → In simple words, if a symbol in the plaintext is changed, several or all symbols in the ciphertext will also change.

BOOK

The idea of diffusion is to hide the relationship between the ciphertext and plaintext.


a/c to wikipedia
Diffusion means that if we change a single bit of the plaintext, then (statically) half of the bits in the ciphertext should change, and similarly,
if we change 1 bit of ciphertext, then atleast one half of the plaintext bits should change.
Diffusion implies that each symbol in the ciphertext is dependent on some or all the symbols in the plaintext.


2) CONFUSION →
is maintained as complex as possible.
→ It hides the relationship b/w ciphertext and the key.
→ If a single bit in the key is changed then most/all bits of the ciphertext will also be changed.
A/c To wikipedia,
Confusion means that each bit of the ciphertext should depend on several parts of the key, obscuring the connection b/w the two.
make unclear or difficult to understand.



S.NO	Confusion	Diffusion
1.	Confusion is a cryptographic technique which is used to create faint cipher texts.	While diffusion is used to create cryptic plain texts.
2.	This technique is possible through substitution algorithm.	While it is possible through transportation algorithm.
3.	In confusion, if one bit within the secret's modified, most or all bits within the cipher text also will be modified.	While in diffusion, if one image within the plain text is modified, many or all image within the cipher text also will be modified
4.	In confusion, vagueness is increased in resultant.	While in diffusion, redundancy is increased in resultant.
5.	Both stream cipher and block cipher uses confusion.	Only block cipher uses diffusion.
6.	The relation between the cipher text and the key is masked by confusion.	While The relation between the cipher text and the plain text is masked by diffusion.

Chapter -6

Advanced Encryption Standard

5.2 AES STRUCTURE

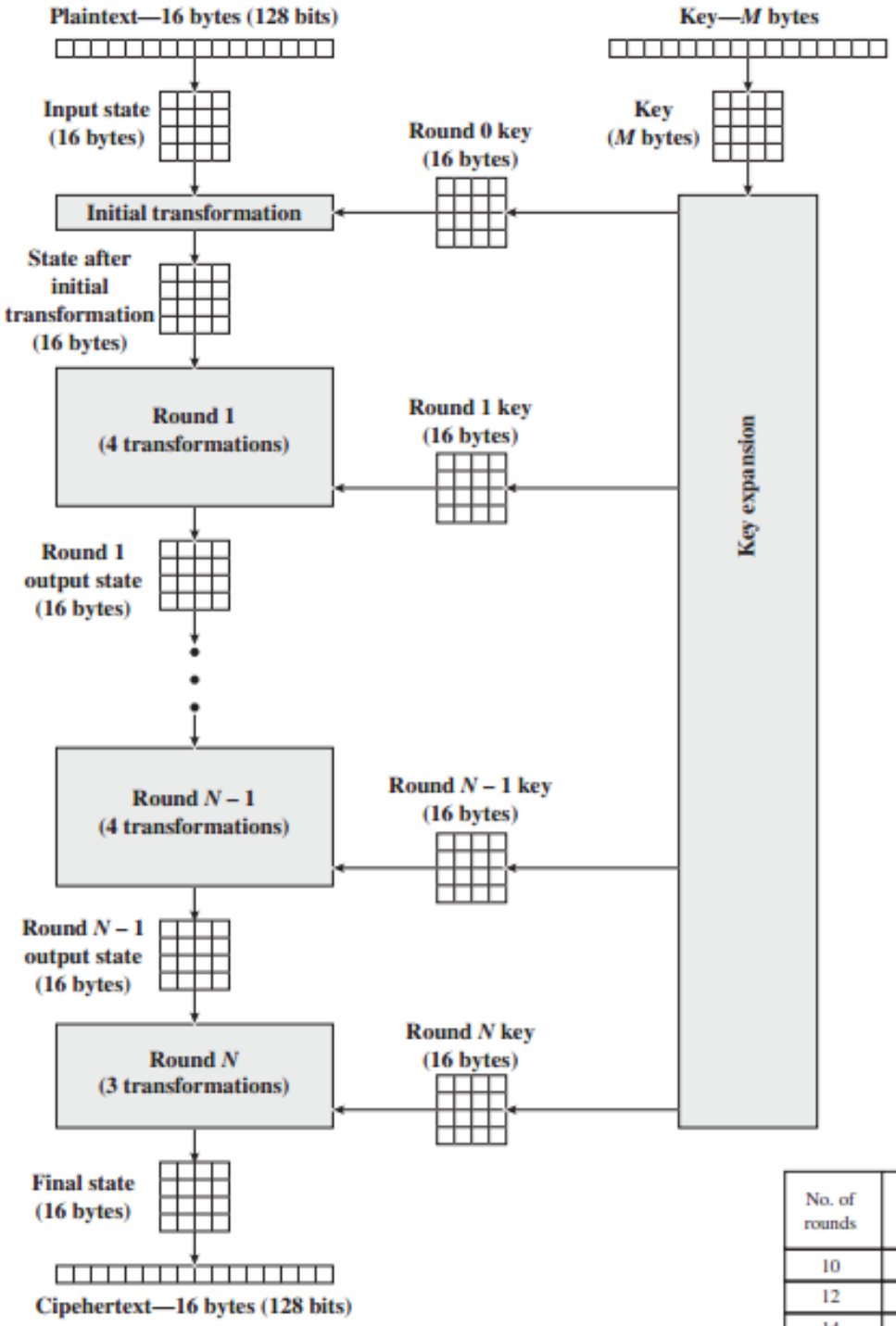
General Structure

Figure 5.1 shows the overall structure of the AES encryption process. The cipher takes a plaintext block size of 128 bits, or 16 bytes. The key length can be 16, 24, or 32 bytes (128, 192, or 256 bits). The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length.

The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a 4×4 square matrix of bytes. This block is copied into the **State** array, which is modified at each stage of encryption or decryption. After the final stage, **State** is copied to an output matrix. These operations are depicted in Figure 5.2a. Similarly, the key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words. Figure 5.2b shows the expansion for the 128-bit key. Each word is four bytes, and the total key schedule is 44 words for the 128-bit key. Note that the ordering of bytes within a matrix is by column. So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the **in** matrix, the second four bytes occupy the second column, and so on. Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the **w** matrix.

The cipher consists of N rounds, where the number of rounds depends on the key length: 10 rounds for a 16-byte key, 12 rounds for a 24-byte key, and 14 rounds for a 32-byte key (Table 5.1). The first $N - 1$ rounds consist of four distinct transformation functions: SubBytes, ShiftRows, MixColumns, and AddRoundKey, which are described subsequently. The final round contains only three transformations, and there is a initial single transformation (AddRoundKey) before the first round,

which can be considered Round 0. Each transformation takes one or more 4×4 matrices as input and produces a 4×4 matrix as output. Figure 5.1 shows that the output of each round is a 4×4 matrix, with the output of the final round being the ciphertext. Also, the key expansion function generates $N + 1$ round keys, each of which is a distinct 4×4 matrix. Each round key serve as one of the inputs to the AddRoundKey transformation in each round.



No. of rounds	Key Length (bytes)
10	16
12	24
14	32

Figure 5.1 AES Encryption Process

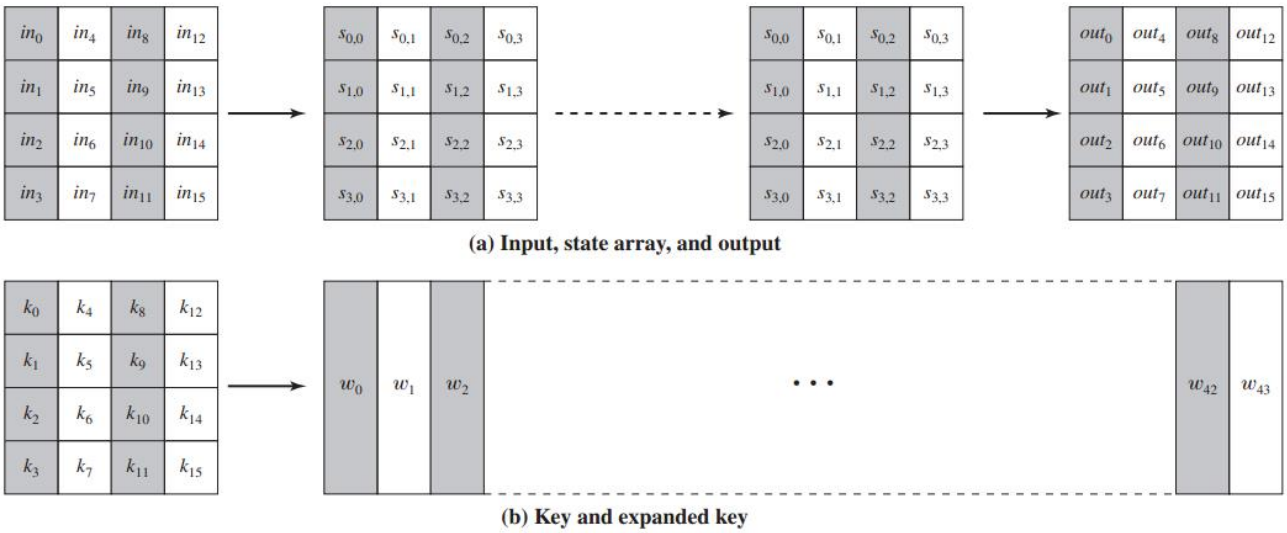


Figure 5.2 AES Data Structures

Table 5.1 AES Parameters

Key Size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext Block Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of Rounds	10	12	14
Round Key Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded Key Size (words/bytes)	44/176	52/208	60/240

Detailed Structure

Figure 5.3 shows the AES cipher in more detail, indicating the sequence of transformations in each round and showing the corresponding decryption function. As was done in Chapter 3, we show encryption proceeding down the page and decryption proceeding up the page.

Before delving into details, we can make several comments about the overall AES structure.

1. One noteworthy feature of this structure is that it is not a Feistel structure. Recall that, in the classic Feistel structure, half of the data block is used to modify the other half of the data block and then the halves are swapped. AES instead processes the entire data block as a single matrix during each round using substitutions and permutation.
2. The key that is provided as input is expanded into an array of forty-four 32-bit words, $\mathbf{w}[i]$. Four distinct words (128 bits) serve as a round key for each round; these are indicated in Figure 5.3.

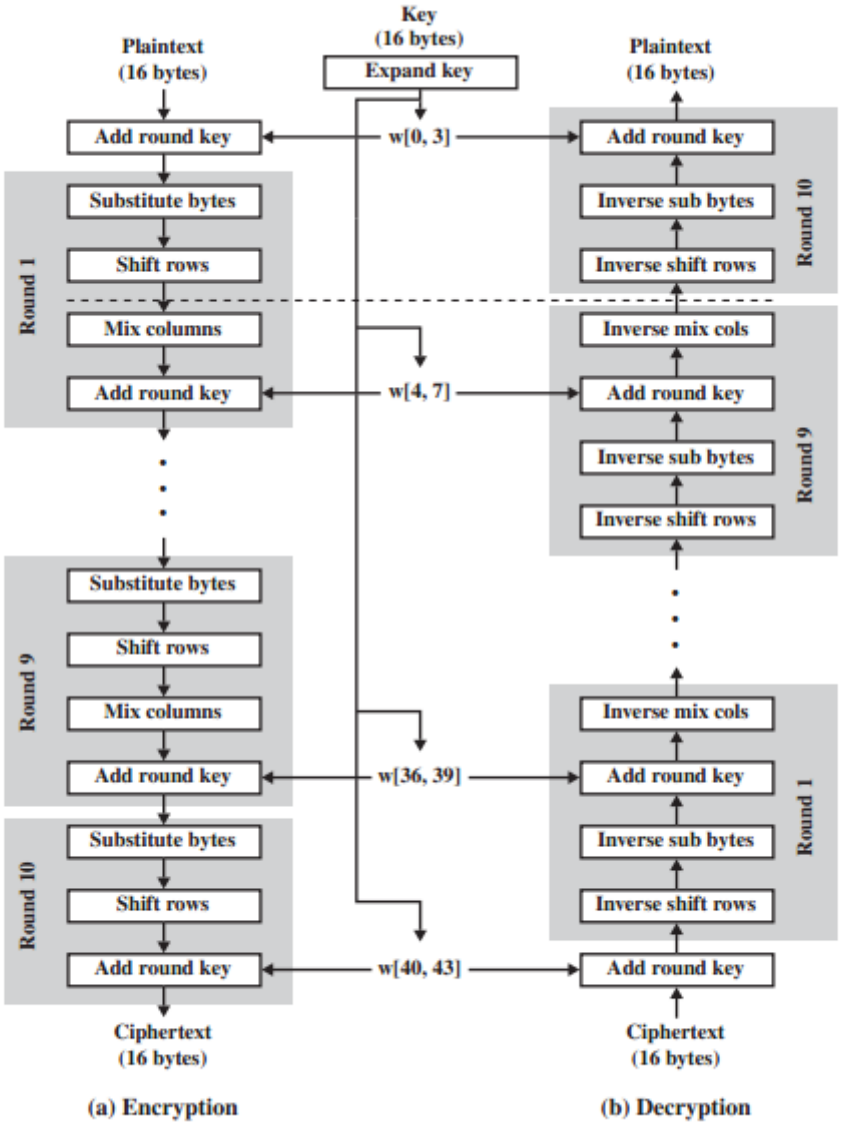


Figure 5.3 AES Encryption and Decryption

3. Four different stages are used, one of permutation and three of substitution:
   - **Substitute bytes:** Uses an S-box to perform a byte-by-byte substitution of the block
   - **ShiftRows:** A simple permutation
   - **MixColumns:** A substitution that makes use of arithmetic over GF(2^8)
   - **AddRoundKey:** A simple bitwise XOR of the current block with a portion of the expanded key
4. The structure is quite simple. For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. Figure 5.4 depicts the structure of a full encryption round.

5. Only the AddRoundKey stage makes use of the key. For this reason, the cipher begins and ends with an AddRoundKey stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.
6. The AddRoundKey stage is, in effect, a form of Vernam cipher and by itself would not be formidable. The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key. We can view the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure.
7. Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block, using the result that $A \oplus B \oplus B = A$.
8. As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not

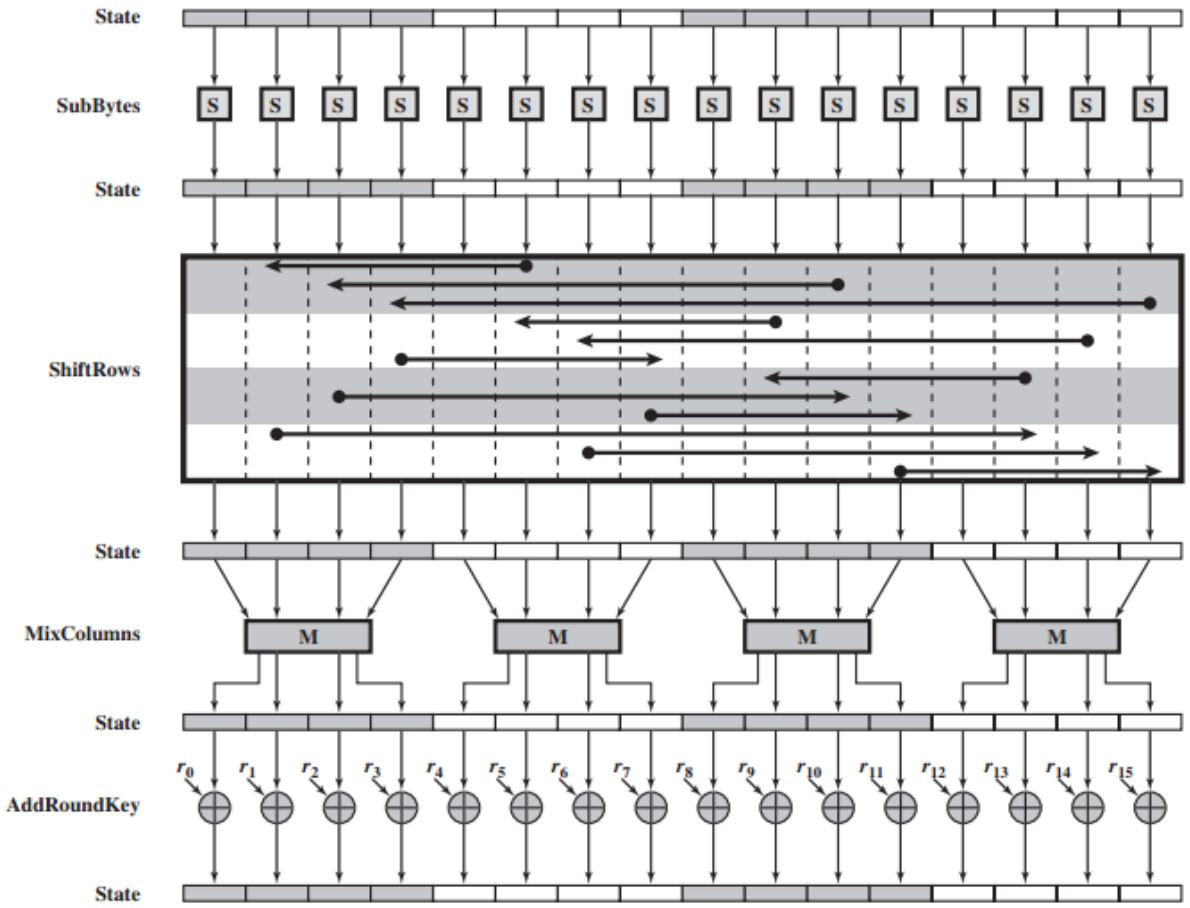


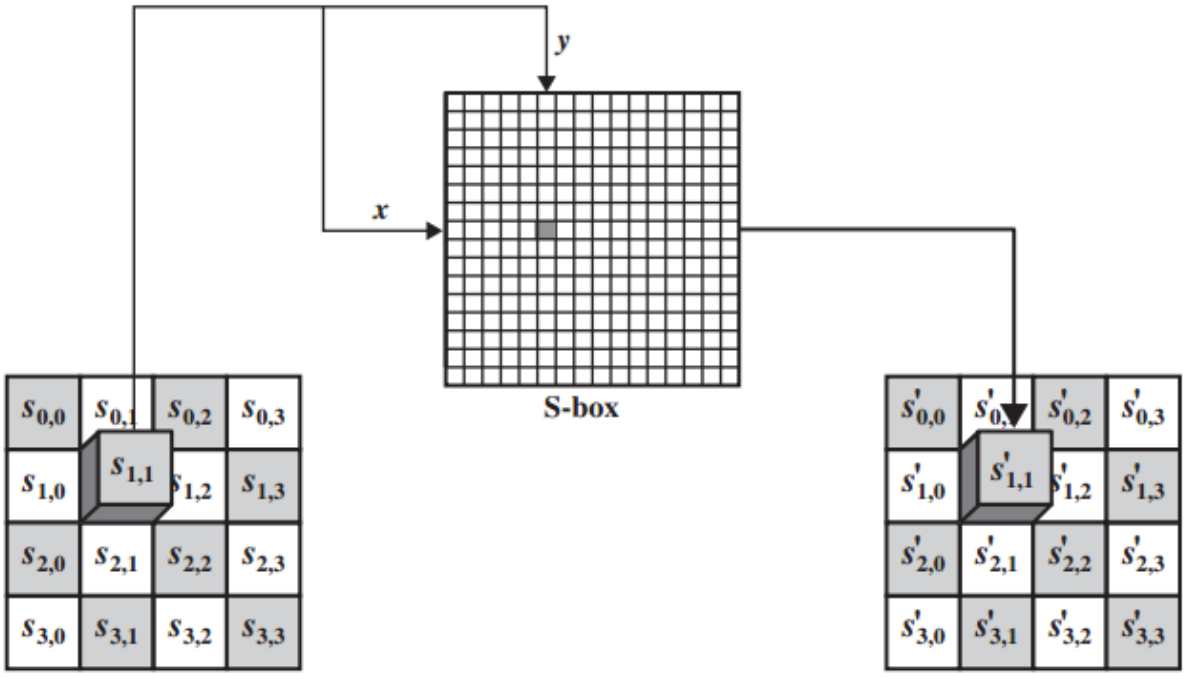
Figure 5.4 AES Encryption Round

identical to the encryption algorithm. This is a consequence of the particular structure of AES.

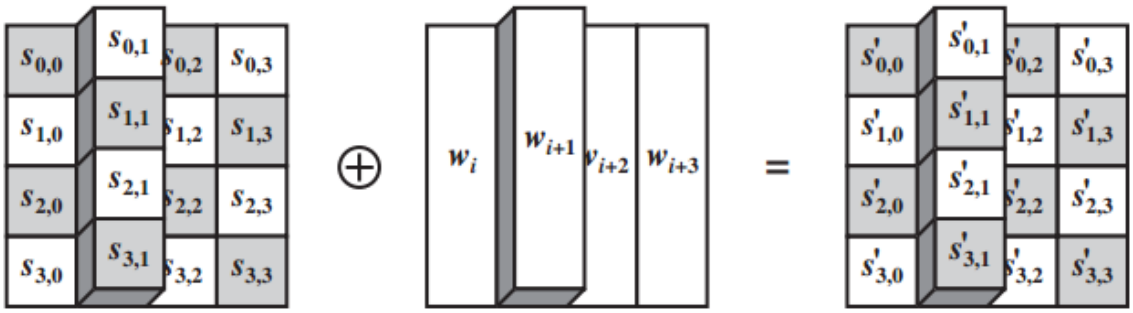
9. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. Figure 5.3 lays out encryption and decryption going in opposite vertical directions. At each horizontal point (e.g., the dashed line in the figure), **State** is the same for both encryption and decryption.
10. The final round of both encryption and decryption consists of only three stages. Again, this is a consequence of the particular structure of AES and is required to make the cipher reversible.

Substitute Bytes Transformation

Forward and Inverse Transformations The **forward substitute byte transformation**, called SubBytes, is a simple table lookup (Figure 5.5a). AES defines a 16 × 16 matrix of byte values, called an S-box (Table 5.2a), that contains a permutation of all possible 256 8-bit values. Each individual byte of **State** is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value. For example, the hexadecimal value[3] {95} references row 9, column 5



(a) Substitute byte transformation



(b) Add round key transformation

Figure 5.5 AES Byte-Level Operations

Table 5.2 AES S-Boxes

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(a) S-box

		y														
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

(b) Inverse S-box

of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.

Here is an example of the SubBytes transformation:

EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

ShiftRows Transformation

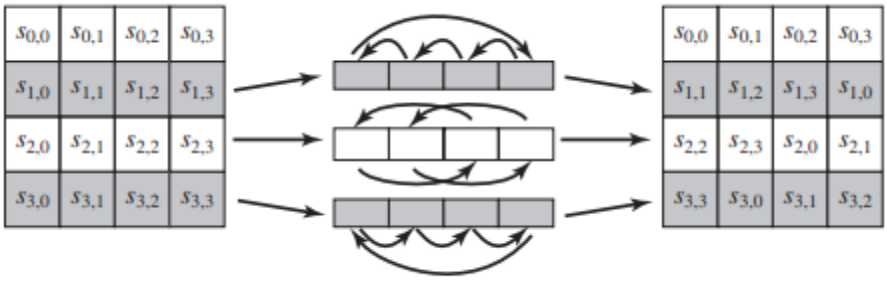
Forward and Inverse Transformations The **forward shift row transformation**, called ShiftRows, is depicted in Figure 5.7a. The first row of **State** is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed. The following is an example of ShiftRows.

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

The **inverse shift row transformation**, called InvShiftRows, performs the circular shifts in the opposite direction for each of the last three rows, with a 1-byte circular right shift for the second row, and so on.



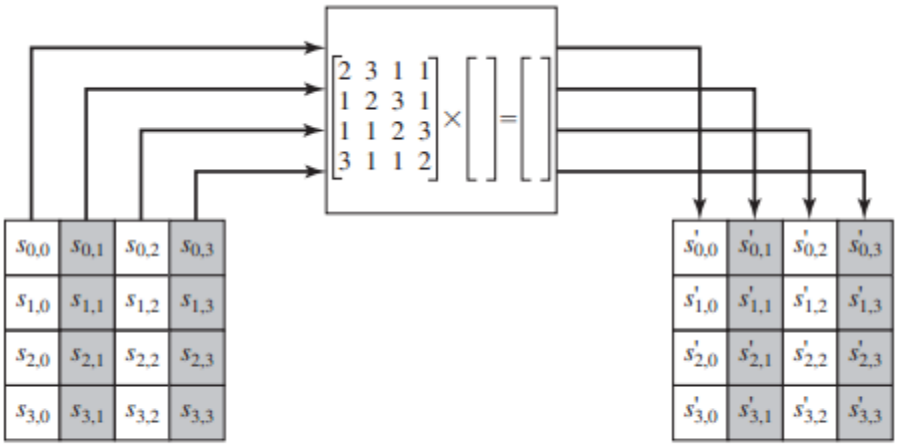
(a) Shift row transformation

MixColumns Transformation

Forward and Inverse Transformations The **forward mix column transformation**, called MixColumns, operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column. The transformation can be defined by the following matrix multiplication on **State** (Figure 5.7b):

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \quad (5.3)$$

Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications[5] are performed



(b) Mix column transformation

Figure 5.7 AES Row and Column Operations

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \quad (5.3)$$

in GF(2^8). The MixColumns transformation on a single column of **State** can be expressed as

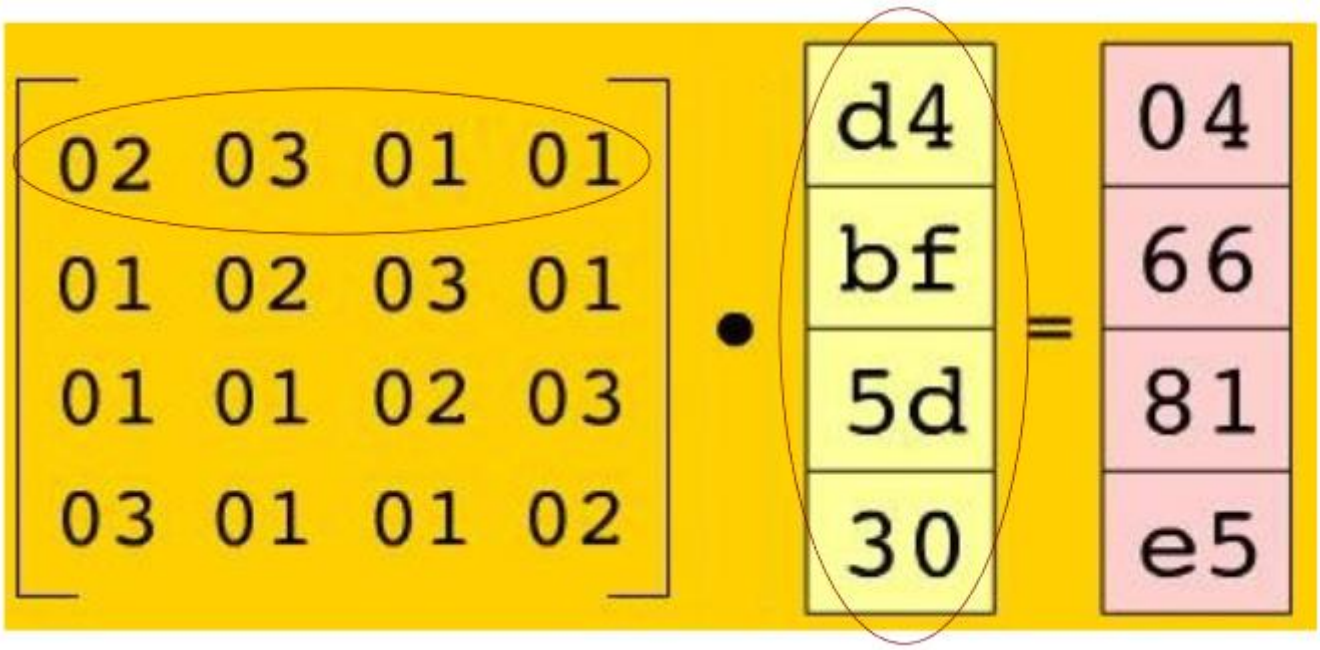
$$\begin{aligned} s'_{0,j} &= (2 \bullet s_{0,j}) \oplus (3 \bullet s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus (2 \bullet s_{1,j}) \oplus (3 \bullet s_{2,j}) \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \bullet s_{2,j}) \oplus (3 \bullet s_{3,j}) \\ s'_{3,j} &= (3 \bullet s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \bullet s_{3,j}) \end{aligned} \quad (5.4)$$

The following is an example of MixColumns:

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC



Now we are pretty much ready to calculate the answers. Like I mention early, we will multiply the rows with the column. Let's take the first row of the first matrix and multiply them with our a's values.

To get the r_0 value, the formula goes like this:

$$r_0 = \{02.d4\} + \{03.bf\} + \{01.5d\} + \{01.30\}$$

Wow. Does it not seems easy to obtain the answer? Yes, it LOOKS easy. But when it comes to calculating, apparently it isn't anymore. We will go into the steps one at a time.

1. {02.d4}

We will start with converting d4 to binary. Remember d4 is a byte so when using the Calculator program on the computer, change it to byte under Hex mode. (Qword is usable but I still prefer to change to byte just in case)

d4 = 1101 0100

Now d4 is exactly 8 bits which is good. In the case where you never get a 8 bits long characters such as 25 in Hex (converted: 100101), pad on with 0 in the front of the result until you get 8 characters of 1's and 0's. (25 ends up with 0010 0101)

Now another thing to remember, there is a rule established in the multiplication of the values as written in the book, Cryptography and Network Security[2], that multiplication of a value by x (ie. by 02) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with (0001 1011) if the leftmost bit of the original value (before the shift) is 1. We can implement this rule in our calculation.

{d4}.{02} = 1101 0100 << 1 (<< is left shift, 1 is the number of shift done, pad on with 0's)
= 1010 1000 XOR 0001 1011 (because the leftmost is a 1 before shift)
= 1011 0011 (ans)

Calculation:

1010 1000
0001 1011 (XOR)
1011 0011

Now we do the same for our next set of values, {03.bf}

2. {03.bf}

Similarly, we convert bf into binary:

bf = 1011 1111

In this case, we are multiplying 03 to bf. Maybe you are starting to wonder how we are going to multiply them, or some might just multiply them directly. For my case, I followed what was suggested in the book[2], we split 03 up in its binary form.

03 = 11
= 10 XOR 01

We are now able to calculate our result.

{03} . {bf} = {10 XOR 01} . {1011 1111}
= {1011 1111 . 10} XOR {1011 1111 . 01}
= {1011 1111 . 10} XOR {1011 1111}
(Because {1011 1111} x 1[in decimal] = 1011 1111)
= 0111 1110 XOR 0001 1011 XOR 1011 1111
= 1101 1010 (ans)

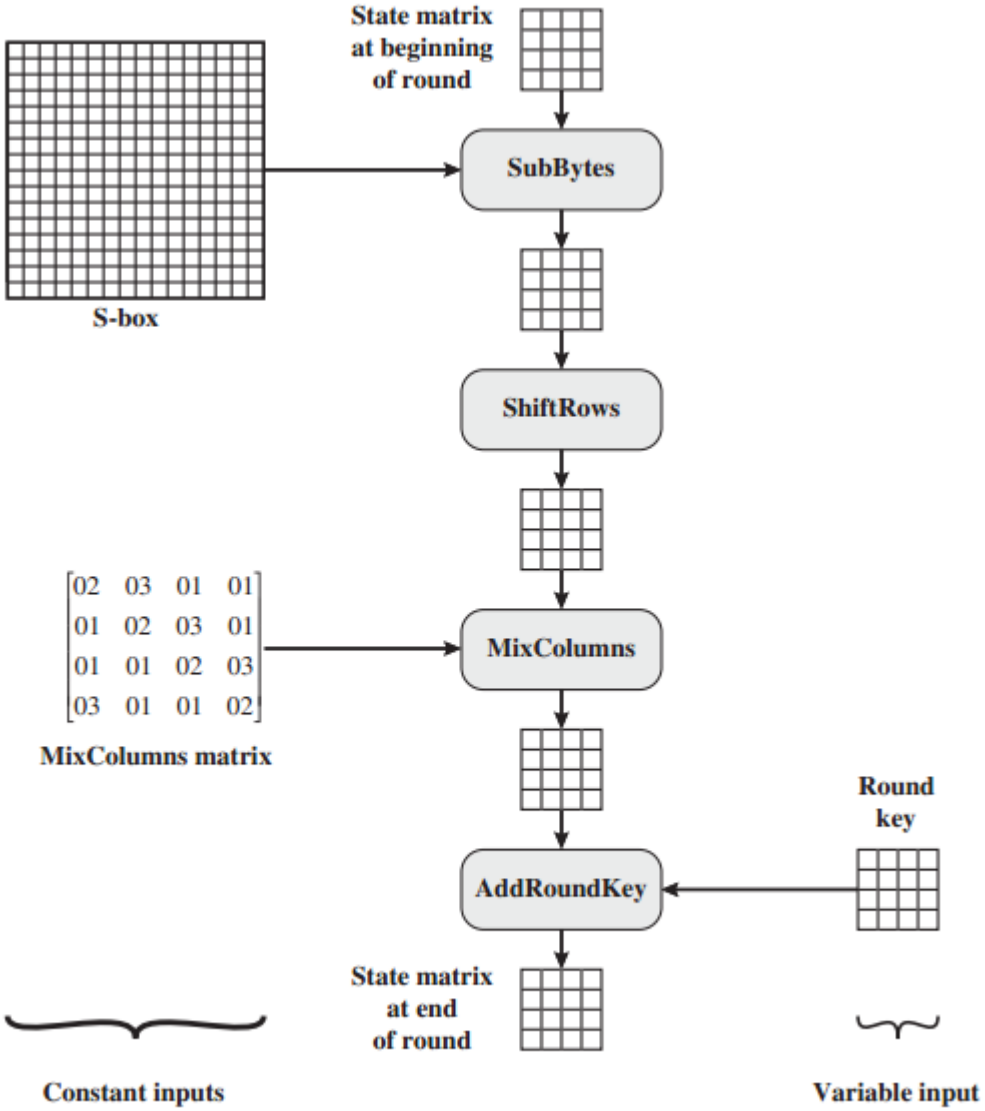


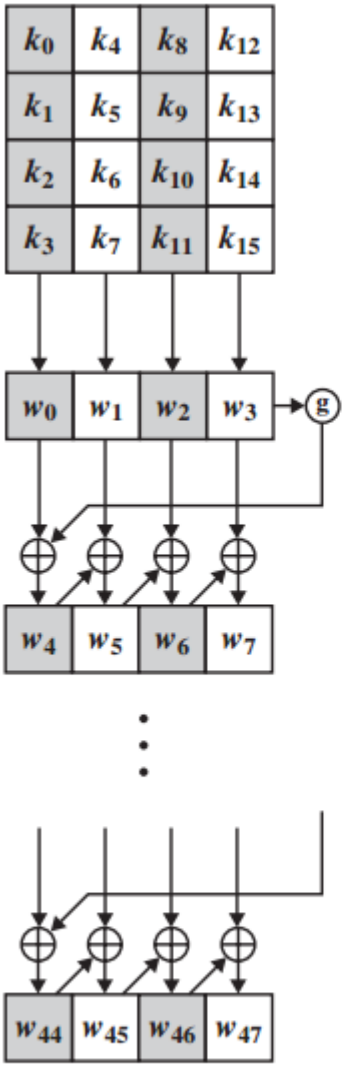
Figure 5.8 Inputs for Single AES Round

5.4 AES KEY EXPANSION

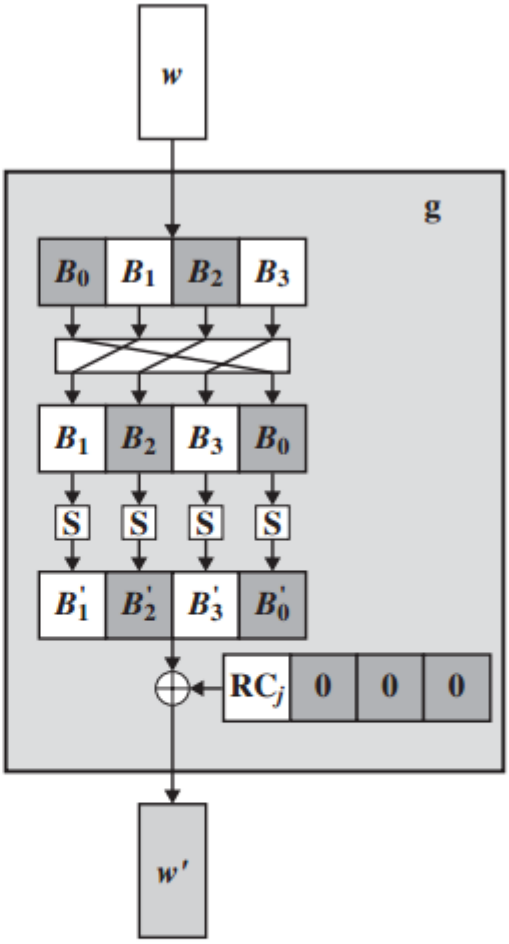
Key Expansion Algorithm

The AES key expansion algorithm takes as input a four-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a four-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher. The pseudocode on the next page describes the expansion.

The key is copied into the first four words of the expanded key. The remainder of the expanded key is filled in four words at a time. Each added word **w**[i] depends on the immediately preceding word, **w**[i − 1], and the word four positions back, w[i − 4]. In three out of four cases, a simple XOR is used. For a word whose position in the **w** array is a multiple of 4, a more complex function is used. Figure 5.9 illustrates the generation of the expanded key, using the symbol g to represent that complex function. The function g consists of the following subfunctions.



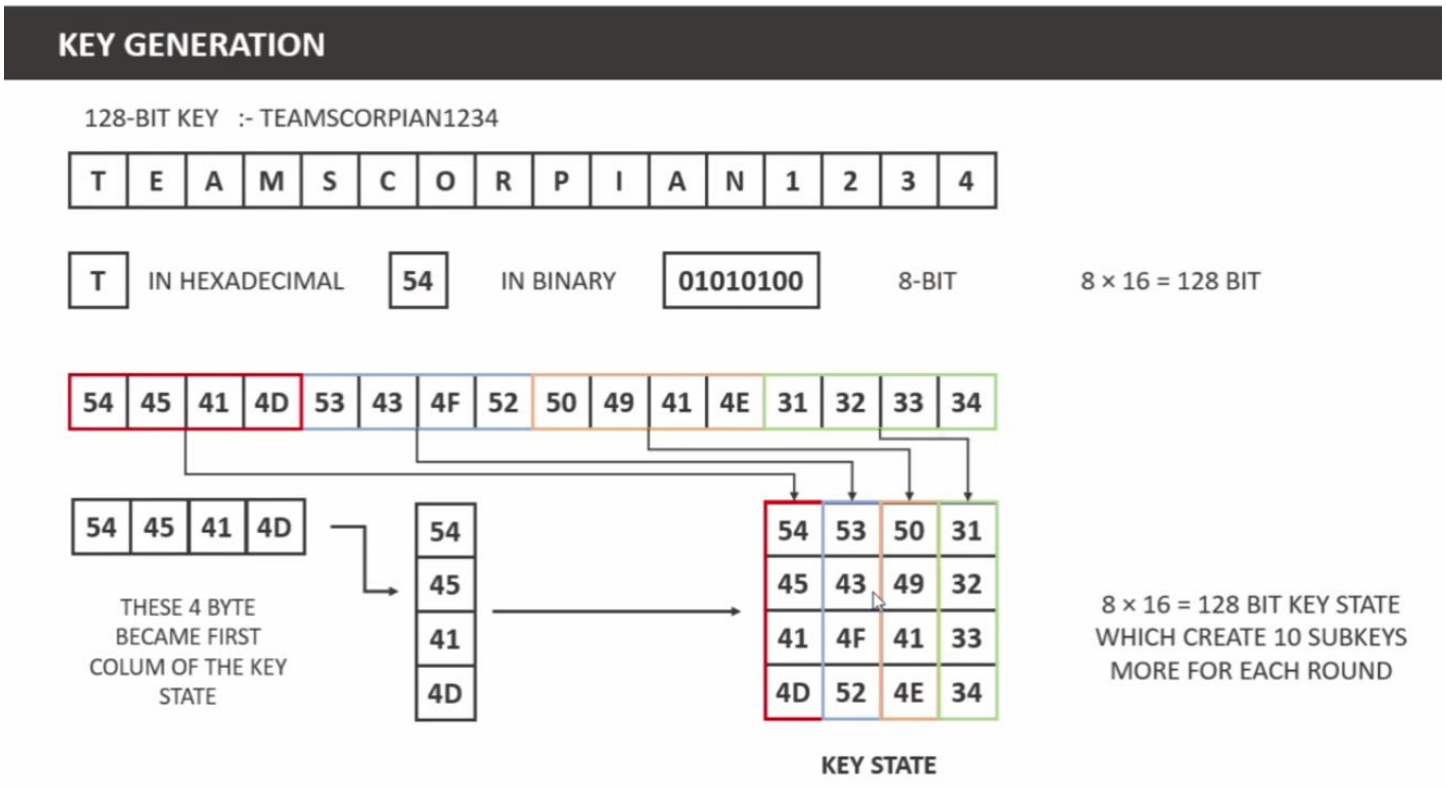
(a) Overall algorithm

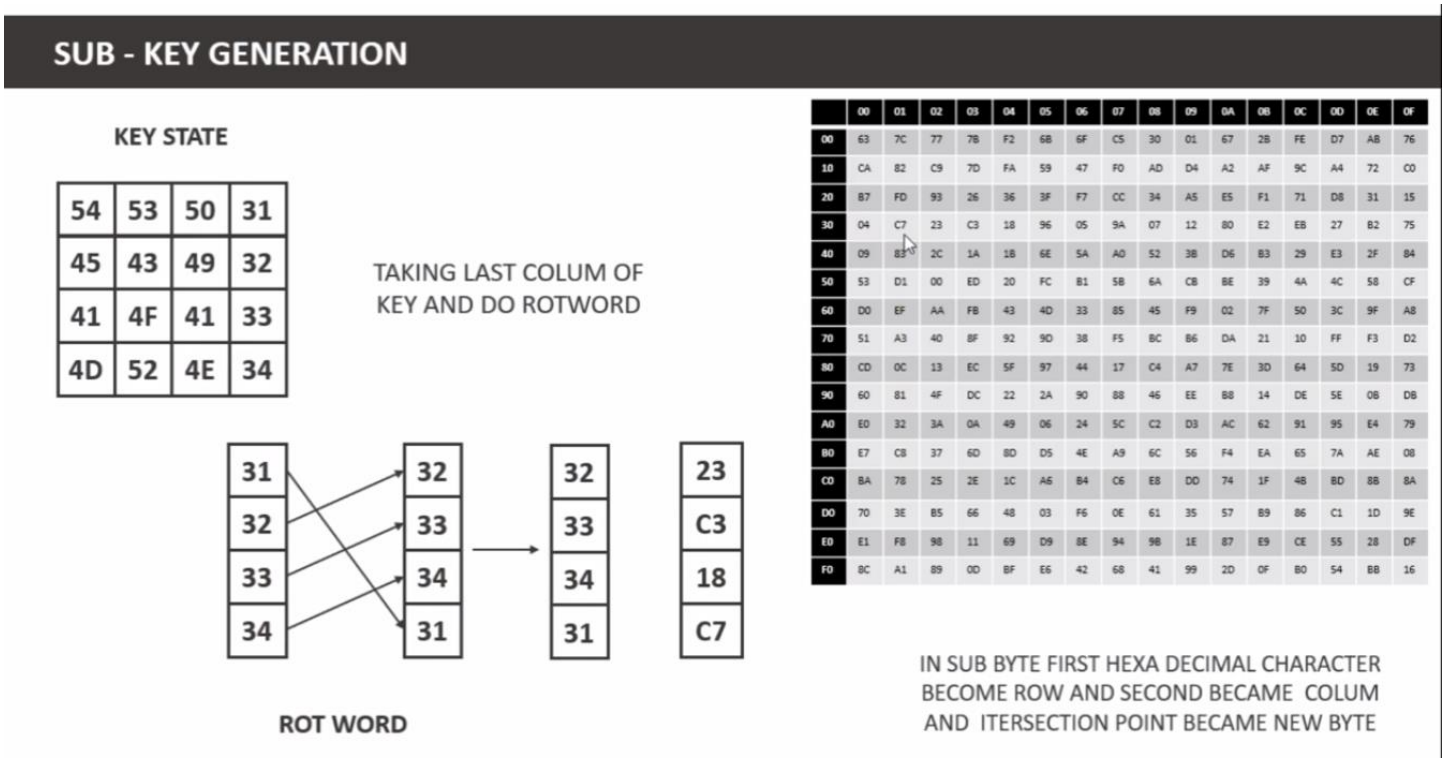


(b) Function g

Figure 5.9 AES Key Expansion

Example:





	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
10	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
20	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
30	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
40	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
50	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
60	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
70	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
80	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
90	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A0	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B0	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C0	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D0	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E0	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F0	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

SUB - KEY GENERATION

KEY STATE

54	53	50	31
45	43	49	32
41	4F	41	33
4D	52	4E	34

AFTER CALCULATING ROTWORD AND SUB BYTE OF LAST COLUM IN PREVIOUS SILDE WE GET, THIS COLUM

54		23		01		76
45		C3		00		86
41	XOR	18	XOR	00	=	59
4D		C7		00		8A

FIRST COLUM | AFTER SUB BYTE COLUM | RCON

RCON

01	02	04	08	10	20	40	80	1B	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

RCON IS A PRE DEFINED TABLE FOR KEY GENERATION IN AES

SUB - KEY GENERATION

KEY STATE

54	53	50	31
45	43	49	32
41	4F	41	33
4D	52	4E	34

AFTER CALCULATING ROTWORD AND SUB BYTE OF LAST COLUM IN PREVIOUS SILDE WE GET, THIS COLUM

XOR

54		23		01		76	
45		C3		00		86	
41	XOR	18	XOR	00	=	59	
4D		C7		00		8A	

FIRST COLUM | AFTER SUB | RCON

RCON

01	02	04	08	10	20	40	80	1B	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

RCON IS A PRE DEFINED TABLE FOR KEY GENERATION IN AES

SUB - KEY GENERATION

KEY STATE

54	53	50	31
45	43	49	32
41	4F	41	33
4D	52	4E	34

AFTER CALCULATING ROTWORD AND SUB BYTE OF LAST COLUM IN PREVIOUS SILDE WE GET, THIS COLUM

XOR

54		23		01		76	25
45		C3		00		86	C5
41	XOR	18	XOR	00	=	59	16
4D		C7		00		8A	D8

FIRST COLUM | AFTER SUB BYTE COLUM | RCON

RCON

01	02	04	08	10	20	40	80	1B	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

RCON IS A PRE DEFINED TABLE FOR KEY GENERATION IN AES

SUB - KEY GENERATION

KEY STATE

54	53	50	31
45	43	49	32
41	4F	41	33
4D	52	4E	34

AFTER CALCULATING ROTWORD AND SUB BYTE OF LAST COLUM IN PREVIOUS SILDE WE GET, THIS COLUM

XOR

54		23		01		76	25	75
45		C3		00		86	C5	8C
41	XOR	18	XOR	00	=	59	16	57
4D		C7		00		8A	D8	96

FIRST COLUM | AFTER SUB BYTE COLUM | RCON

RCON

01	02	04	08	10	20	40	80	1B	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

RCON IS A PRE DEFINED TABLE FOR KEY GENERATION IN AES

SUB - KEY GENERATION

KEY STATE

54	53	50	31
45	43	49	32
41	4F	41	33
4D	52	4E	34

AFTER CALCULATING ROTWORD AND SUB BYTE OF LAST COLUM IN PREVIOUS SILDE WE GET, THIS COLUM

RCON

01	02	04	08	10	20	40	80	1B	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

RCON IS A PRE DEFINED TABLE FOR KEY GENERATION IN AES

54		23		01		76	25	75	44
45		C3		00		86	C5	8C	BE
41	XOR	18	XOR	00	=	59	16	57	64
4D		C7		00		8A	D8	96	A2

FIRST COLUM | AFTER SUB BYTE COLUM | RCON | KEY 1

KEY STATE BECAME KEY 0 ,
KEY 1 WE GET IN THIS SLIDE
AND KEY 1 FURTHER CREATE KEY 2 AND SO ON
EVERY KEY USING DIFFERENT RCON COLUM FOR KEY GENERATION

SUB - KEYS

54	53	50	31	76	25	75	44	6F	4A	3F	7B	4A	00	3F	44	59	59	66	22	DA	83	E5	C7
45	43	49	32	86	C5	8C	BE	28	ED	61	DF	B6	5B	3A	E5	6F	34	0E	EB	86	B2	BC	57
41	4F	41	33	59	16	57	64	1A	0C	5B	3F	B8	B4	EF	D0	C8	7C	93	43	D2	AE	3D	7E
4D	52	4E	34	8A	D8	96	A2	B0	68	FE	5C	FA	92	6C	30	FE	6C	00	30	FA	96	96	A6

KEY 0 | KEY 1 | KEY 2 | KEY 3 | KEY 4 | KEY 5

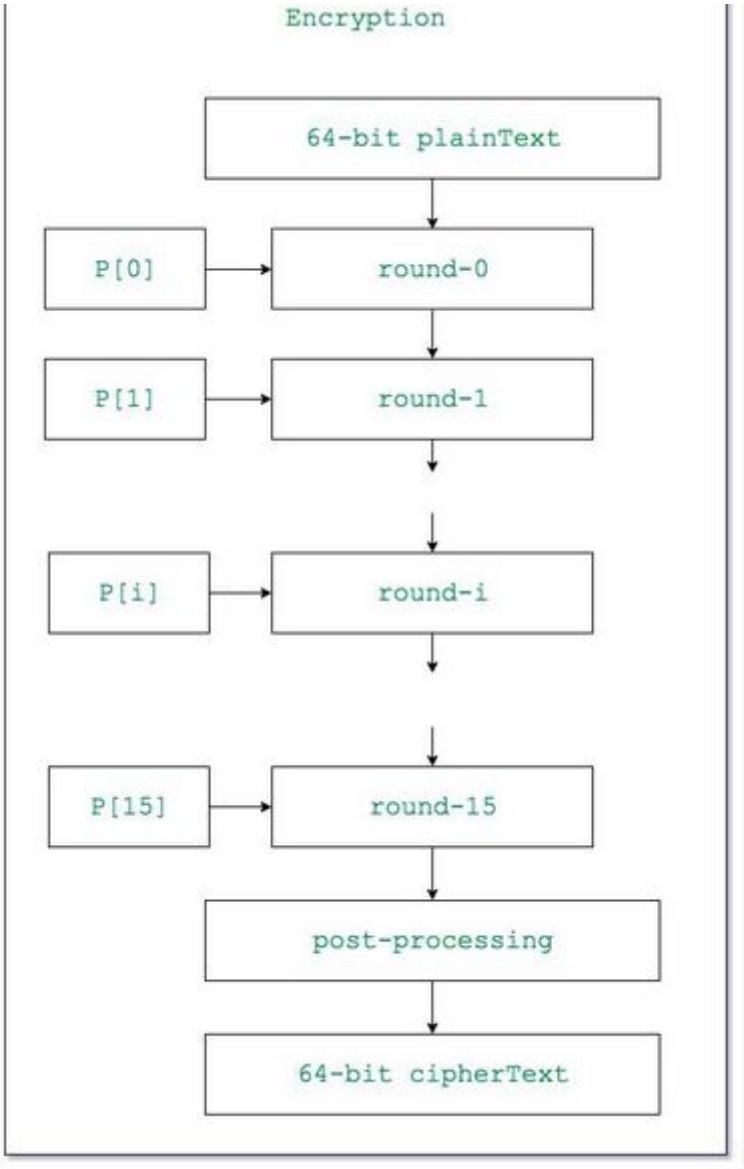
3C	BF	5A	9D	22	9D	C7	5A	1C	81	46	1C	9B	1A	5C	40	A4	BE	E2	A2
DD	6F	D3	84	82	ED	3E	BA	76	9B	A5	1F	B6	2D	88	97	3E	13	9B	0C
21	8F	B2	CC	6A	E5	57	9B	7E	9B	CC	57	25	BE	72	25	1A	A4	D6	F3
DE	48	DE	78	62	2A	F4	8C	06	2C	D8	54	26	0A	D2	86	62	68	BA	3C

KEY 6 | KEY 7 | KEY 8 | KEY 9 | KEY 10

Blowfish Algorithm

Blowfish is an encryption technique designed by **Bruce Schneier** in 1993 as an alternative to DES Encryption Technique. It is significantly faster than DES and provides a good encryption rate with no effective cryptanalysis technique found to date. It is one of the first, secure block cyphers not subject to any patents and hence freely available for anyone to use.

1. **blockSize**: 64-bits
2. **keySize**: 32-bits to 448-bits variable size
3. **number of subkeys**: 18 [P-array]
4. **number of rounds**: 16
5. **number of subsitution boxes**: 4 [each having 512 entries of 32-bits each]



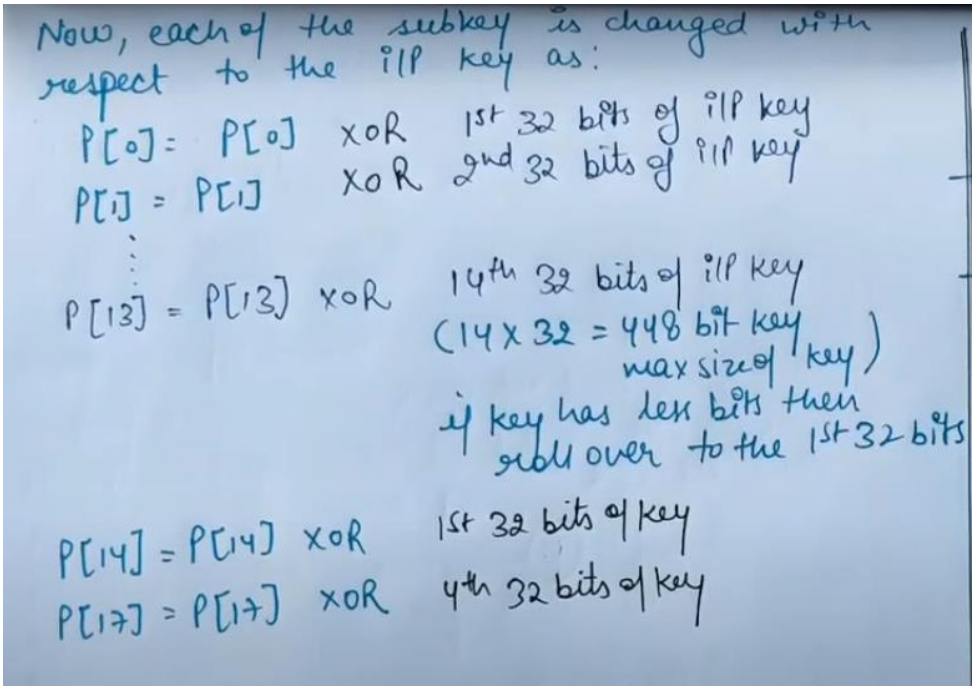
Step1: Generation of subkeys:

- 18 subkeys{P[0]...P[17]} are needed in both encryption aswell as decryption process and the same subkeys are used for both the processes.
- These 18 subkeys are stored in a P-array with each array element being a 32-bit entry.
- It is initialised with the digits of pi(?).
- The hexadecimal representation of each of the subkeys is given by:

```
P[0] = "243f6a88"
P[1] = "85a308d3"
.
.
.
P[17] = "8979fb1b"
```

32-bit hexadecimal representation of initial values of sub-keys

P[0]	: 243f6a88	P[9]	: 38d01377
P[1]	: 85a308d3	P[10]	: be5466cf
P[2]	: 13198a2e	P[11]	: 34e90c6c
P[3]	: 03707344	P[12]	: c0ac29b7
P[4]	: a4093822	P[13]	: c97c50dd
P[5]	: 299f31d0	P[14]	: 3f84d5b5
P[6]	: 082efa98	P[15]	: b5470917
P[7]	: ec4e6c89	P[16]	: 9216d5d9
P[8]	: 452821e6	P[17]	: 8979fb1b



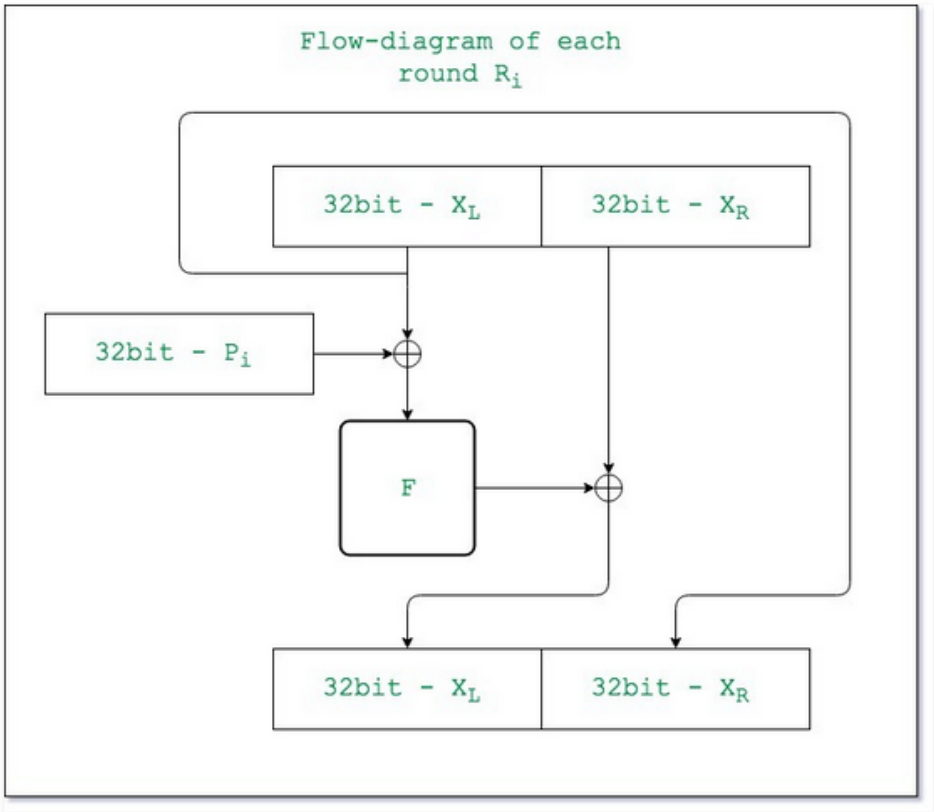
Step2: initialise Substitution Boxes:

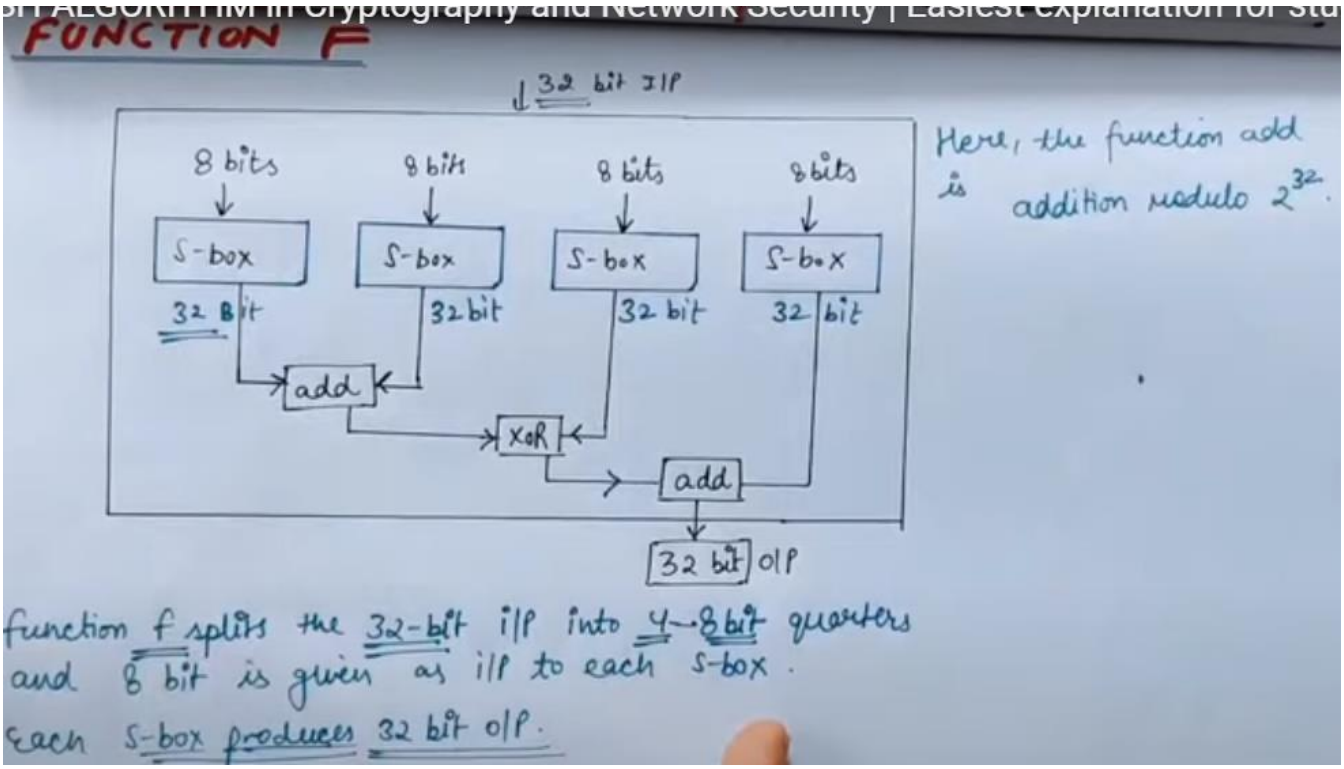
- 4 Substitution boxes(S-boxes) are needed{S[0]...S[4]} in both encryption aswell as decryption process with each S-box having 256 entries{S[i][0]...S[i][255], 0&lei&le4} where each entry is 32-bit.
- It is initialised with the digits of pi(?) after initialising the P-array. You may find the s-boxes in here!

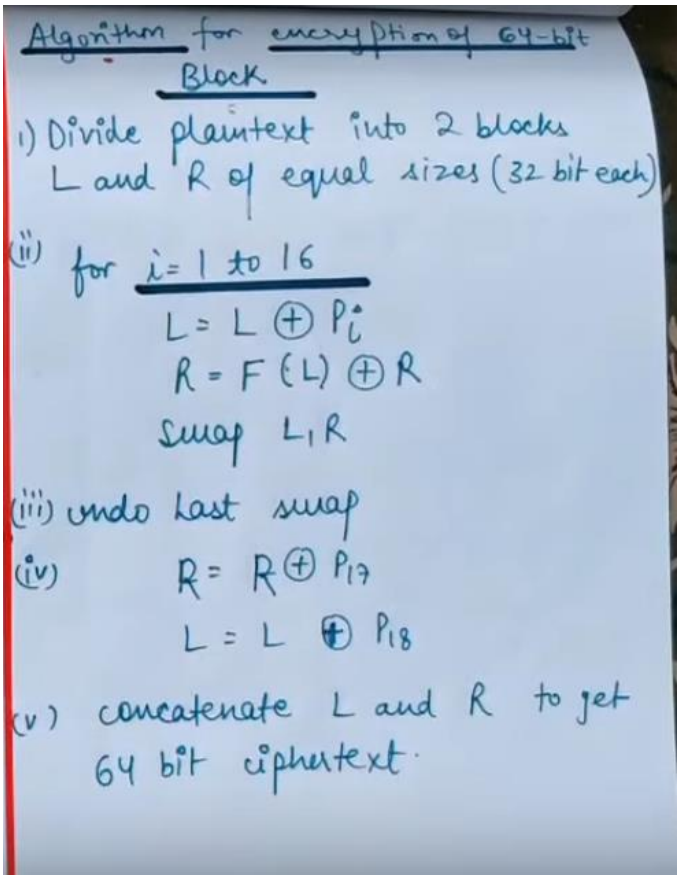
```
S-box 1
d1310ba6 98dfb5ac 2ffd72db d01adfb7 b8e1afed 6a267e96 ba7c9045 f12c7f99
24a19947 b3916cf7 0801f2e2 858efc16 636920d8 71574e69 a458fea3 f4933d7e
0d95748f 728eb658 718bcd58 82154aee 7b54a41d c25a59b5 9c30d539 2af26013
c5d1b023 286085f0 ca417918 b8db38ef 8e79dcb0 603a180e 6c9e0e8b b01e8a3e
d71577c1 bd314b27 78af2fda 55605c60 e65525f3 aa55ab94 57489862 63e81440
55ca396a 2aab10b6 b4cc5c34 1141e8ce a15486af 7c72e993 b3ee1411 636fbc2a
2ba9c55d 741831f6 ce5c3e16 9b87931e afd6ba33 6c24cf5c 7a325381 28958677
3b8f4898 6b4bb9af c4bfe81b 66282193 61d809cc fb21a991 487cac60 5dec8032
ef845d5d e98575b1 dc262302 eb651b88 23893e81 d396acc5 0f6d6ff3 83f44239
2e0b4482 a4842004 69c8f04a 9e1f9b5e 21c66842 f6e96c9a 670c9c61 abd388f0
6a51a0d2 d8542f68 960fa728 ab5133a3 6eef0b6c 137a3be4 ba3bf050 7efb2a98
a1f1651d 39af0176 66ca593e 82430e88 8cee8619 456f9fb4 7d84a5c3 3b8b5ebe
e06f75d8 85c12073 401a449f 56c16aa6 4ed3aa62 363f7706 1bfedf72 429b023d
37d0d724 d00a1248 db0fead3 49f1c09b 075372c9 80991b7b 25d479d8 f6e8def7
e3fe501a b6794c3b 976ce0bd 04c006ba c1a94fb6 409f60c4 5e5c9ec2 196a2463
68fb6faf 3e6c53b5 1339b2eb 3b52ec6f 6dfc511f 9b30952c cc814544 af5ebd09
bee3d004 de334afd 660f2807 192e4bb3 c0cba857 45c8740f d20b5f39 b9d3fbdb
5579c0bd 1a60320a d6a100c6 402c7279 679f25fe fb1fa3cc 8ea5e9f8 db3222f8
3c7516df fd616b15 2f501ec8 ad0552ab 323db5fa fd238760 53317b48 3e00df82
9e5c57bb ca6f8ca0 1a87562e df1769db d542a8f6 287effc3 ac6732c6 8c4f5573
695b27b0 bbca58c8 e1ffa35d b8f011a0 10fa3d98 fd2183b8 4afcb56c 2dd1d35b
9a53e479 b6f84565 d28e49bc 4bfb9790 e1ddf2da a4cb7e33 62fb1341 cee4c6e8
ef20cada 36774c01 d07e9efe 2bf11fb4 95dbda4d ae909198 eaad8e71 6b93d5a0
d08ed1d0 afc725e0 8e3c5b2f 8e7594b7 8ff6e2fb f2122b64 8888b812 900df01c
4fad5ea0 688fc31c d1cff191 b3a8c1ad 2f2f2218 be0e1777 ea752dfe 8b021fa1
e5a0cc0f b56f74e8 18acf3d6 ce89e299 b4a84fe0 fd13e0b7 7cc43b81 d2ada8d9
165fa266 80957705 93cc7314 211a1477 e6ad2065 77b5fa86 c75442f5 fb9d35cf
ebcdaf0c 7b3e89a0 d6411bd3 ae1e7e49 00250e2d 2071b35e 226800bb 57b8e0af
2464369b f009b91e 5563911d 59dfa6aa 78c14389 d95a537f 207d5ba2 02e5b9c5
83260376 6295cfa9 11c81968 4e734a41 b3472dca 7b14a94a 1b510052 9a532915
d60f573f bc9bc6e4 2b60a476 81e67400 08ba6fb5 571be91f f296ec6b 2a0dd915
b6636521 e7b9f9b6 ff34052e c5855664 53b02d5d a99f8fa1 08ba4799 6e85076a
```

Step3: Encryption:

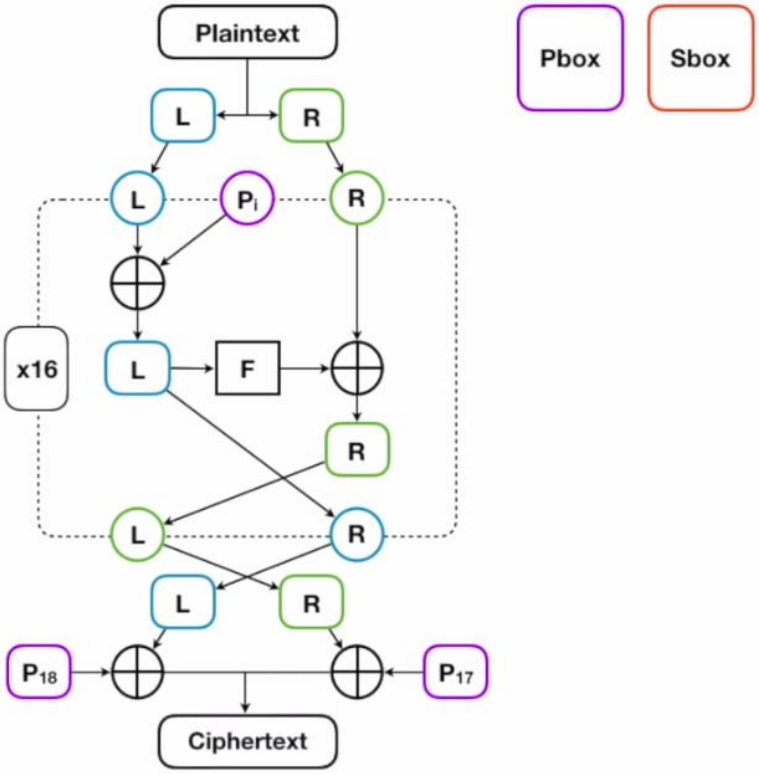
- The encryption function consists of two parts:
  a. **Rounds:** The encryption consists of **16** rounds with each round(Ri) taking inputs the plainText(P.T.) from previous round and corresponding subkey(Pi). The description of each round is as follows:







Overall Structure



RC5- ALgorithm

RC5 is a symmetric Block cipher.

RC5 is symmetric key encryption algorithm developed by **Ronald Rivest**

RC5 is a **block cipher** and addresses two word blocks at a time

Features of RC5 are

- Requires less memory
- Fast : Since uses primitive operation such as addition XOR and shift
- It allows variable no of rounds and variable length of key bits
- Suitable for modern processors, smart cards as well as devices with less memory

RC5 Working - Basic Principle

- Word size (PT block size) in bits: RC5 **encrypts two word blocks at a time** and they are of **16, 32, 64 bits**
- No. of rounds: **0 to 255**
- No. of 8 bit bytes (octets) in the key are **0-255**
- i.e. PT block size can be of 32,64 or 128 bit (since 2-word blocks are used)
- Output resulting from the RC5 i.e. CT has same size as the input plaintext
- RC5 allows variable values in three parameters i.e. PT length, No. of rounds , Key length so at any instance RC5 algo is denoted as RC5- w/r/b or $\mathbf{RC5_{wrb}}$

PARAMETER	POSSIBLE VALUE
block/word size (bits)	16, 32, 64
Number of Rounds	0 – 255
Key Size (bytes)	0 – 255

RC5-w/r/b

where -
w=word size in bits,
r=number of rounds and
b=key size in bytes.

RC5 Working - Basic Principle..

Where -

w- Word size

r - No. of rounds

b - No. of **8 bit bytes** in key

RC5-32/16/16 means

RC5- with block size of **64 bit** as it **uses two word blocks** at a time

RC5- with **16 rounds**

RC5 – with **16 bytes (i.e. 128 bits) in key** values

Rivest suggested **RC5 – 32/12/16** as minimum safety version

RC5 – Operation Principle

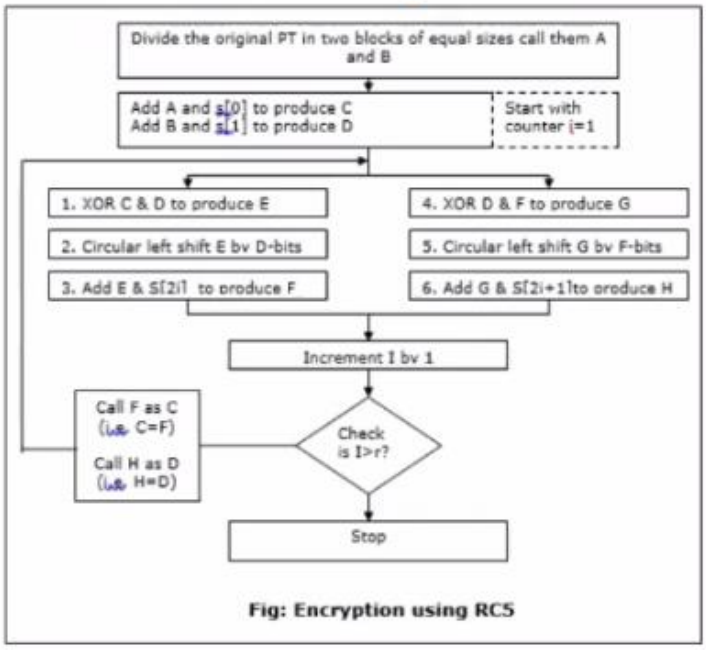


Fig: Encryption using RC5

As shown in above fig. **first two steps are of one-time initial operation.**

Input PT is **divided in to 32 bit blocks A and B**

First two sub keys **s[0] and s[1]** are added to A and B and produces C & D

Now the rounds will begin and in each round there are following operations

- Bitwise XOR
- Left Circular Shift
- Addition with the next sub key for both C and D

As shown in fig. the output of one block is feedback to the input of next block which makes whole logic complicated for cryptanalyst to decipher

Step by Step Algorithmic Details

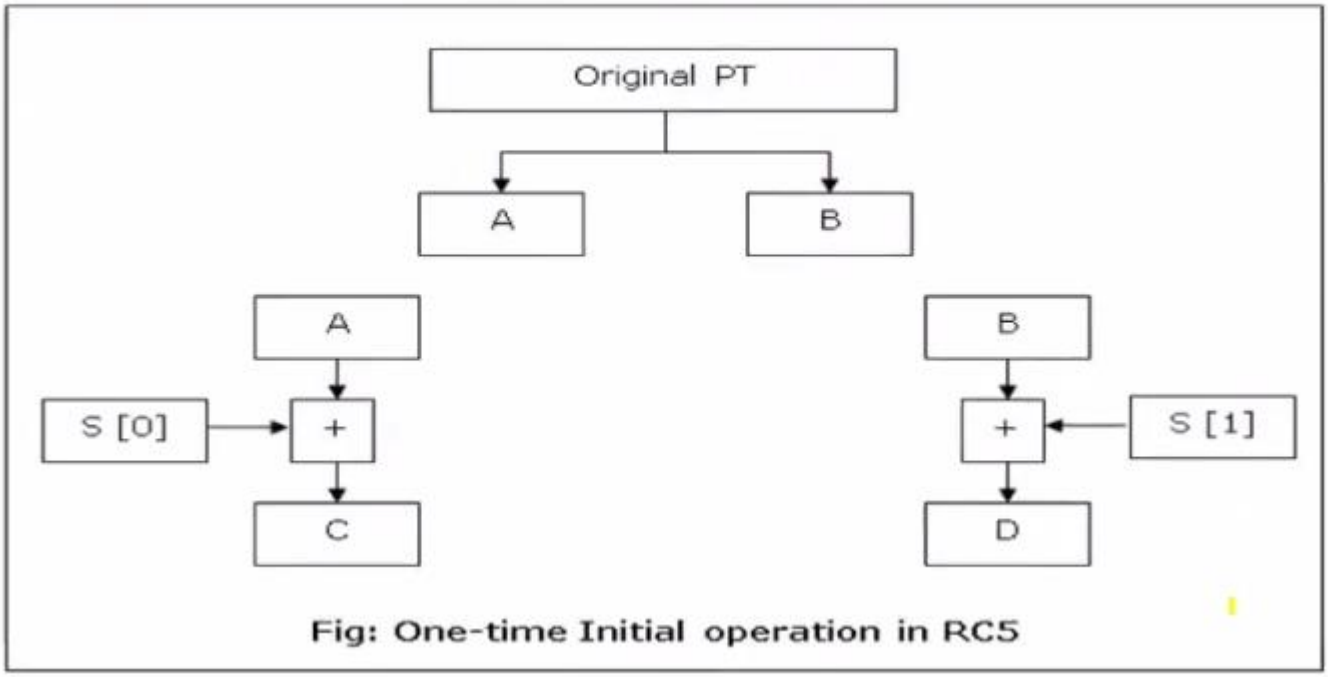
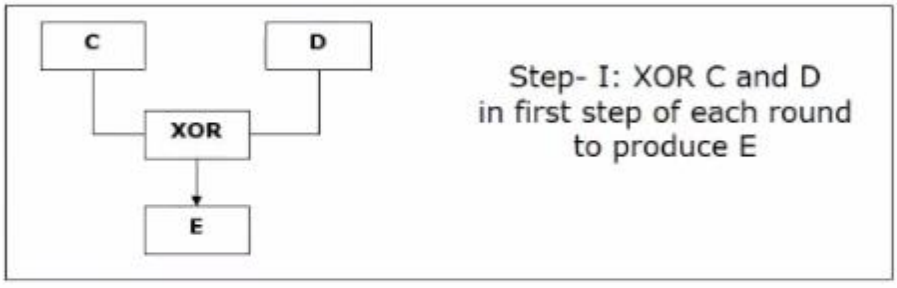
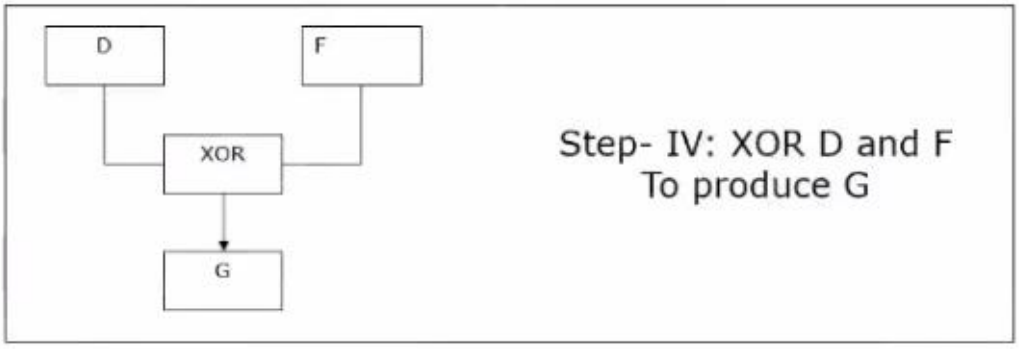


Fig: One-time Initial operation in RC5

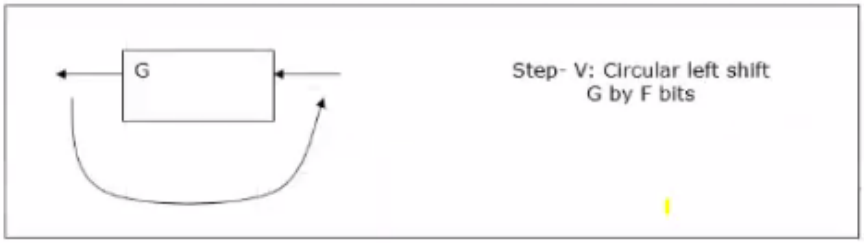
Step -1,2

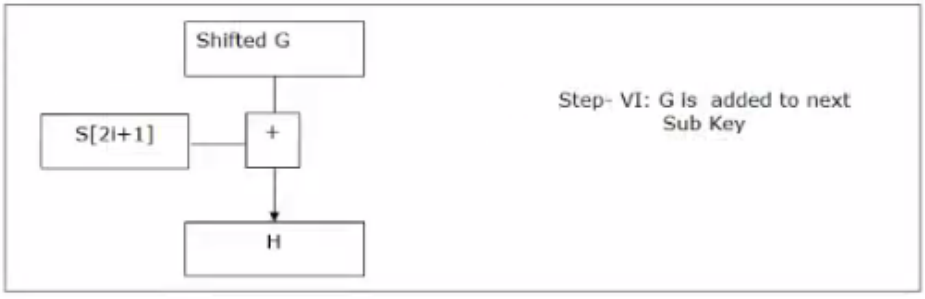


Step – 3, 4



Step – 5, 6





Step VII: Miscellaneous task

- In this step we check to see if all the rounds are over or not for this we perform the following steps
- Increment I by 1
- Check to see if i<r perform following

  I=i+1
  If i<r
  Call F as C again
  Call H as D again
  Go back to step1
  Else
  Stop
  End if

Mathematical representation of RC5 (Encryption):

A=A+S[0]

B=B+S[1]

For i=1 to r

A=((A XOR B) <<< B)+S[2i]

B=((B XOR A) <<< A)+S[2i+1]

Next i

Mathematical representation of RC5 (Decryption) :

For I = r to 1 step-1

A= ((B-S [2i+1] >>> A) XOR A

B= ((A-S [2i] >>> B) XOR B

Next I

B= B-S [1]

A= A-S [0]

Encryption

A=A+S[0]

B=B+S[1]

For i=1 to r

A=((A XOR B) <<< B)+S[2i]

B=((B XOR A) <<< A)+S[2i+1]

Next i

Decryption

For I = r to 1 step-1

A= ((B-S [2i+1] >>> A) XOR A

B= ((A-S [2i] >>> B) XOR B

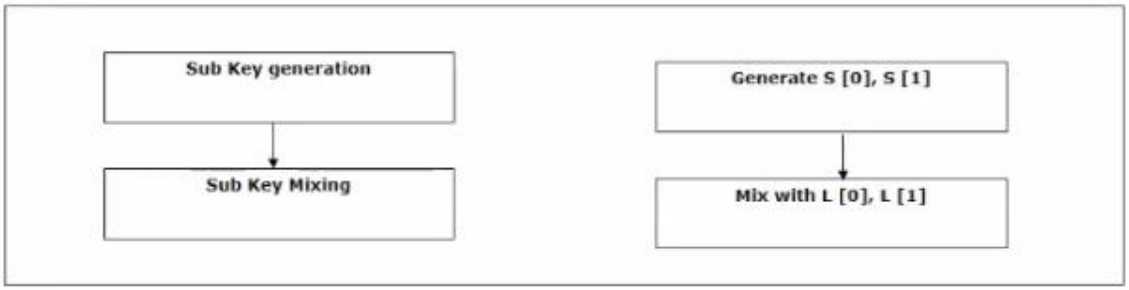
Next I

B= B-S [1]

A= A-S [0]

Sub key Creation Process

- Sub key creation is two step process
- First step the sub keys (denoted by s[0] ,s[1],_ _ _ _) are generated
- Original key is called as L in second step the sub key (s[0], s[1],_ __) are mixed with corresponding sub-portion or original key i.e. L[0],L[1]_ _ _ _ as shown



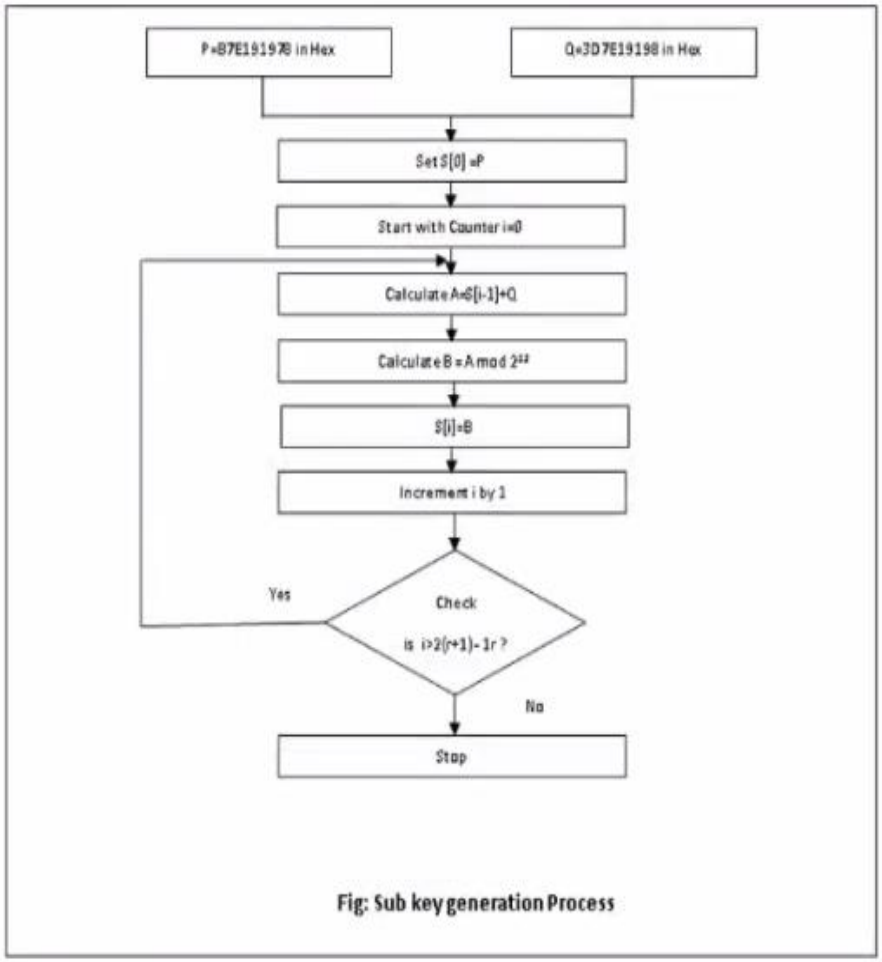


Fig: Sub key generation Process

Mathematical Representation of Sub-key generation

S[0]=P

For i = 1 to 2(r+1) – 1

S[i]= (S[i-1]+Q) mod 2^{32}

Next i

Mathematical Representation of Sub-key Mixing

i=j=0
A = B =0
Do 3n times (where n is the maximum of 2(r+1) and c)

A=s[i]=(s[i] + A + B) <<< 3
B=L[i]=(L[i] + A + B) <<< (A+B)

I = (I + 1) mod 2(r+1)
J = (j + 1) mod c

End Do