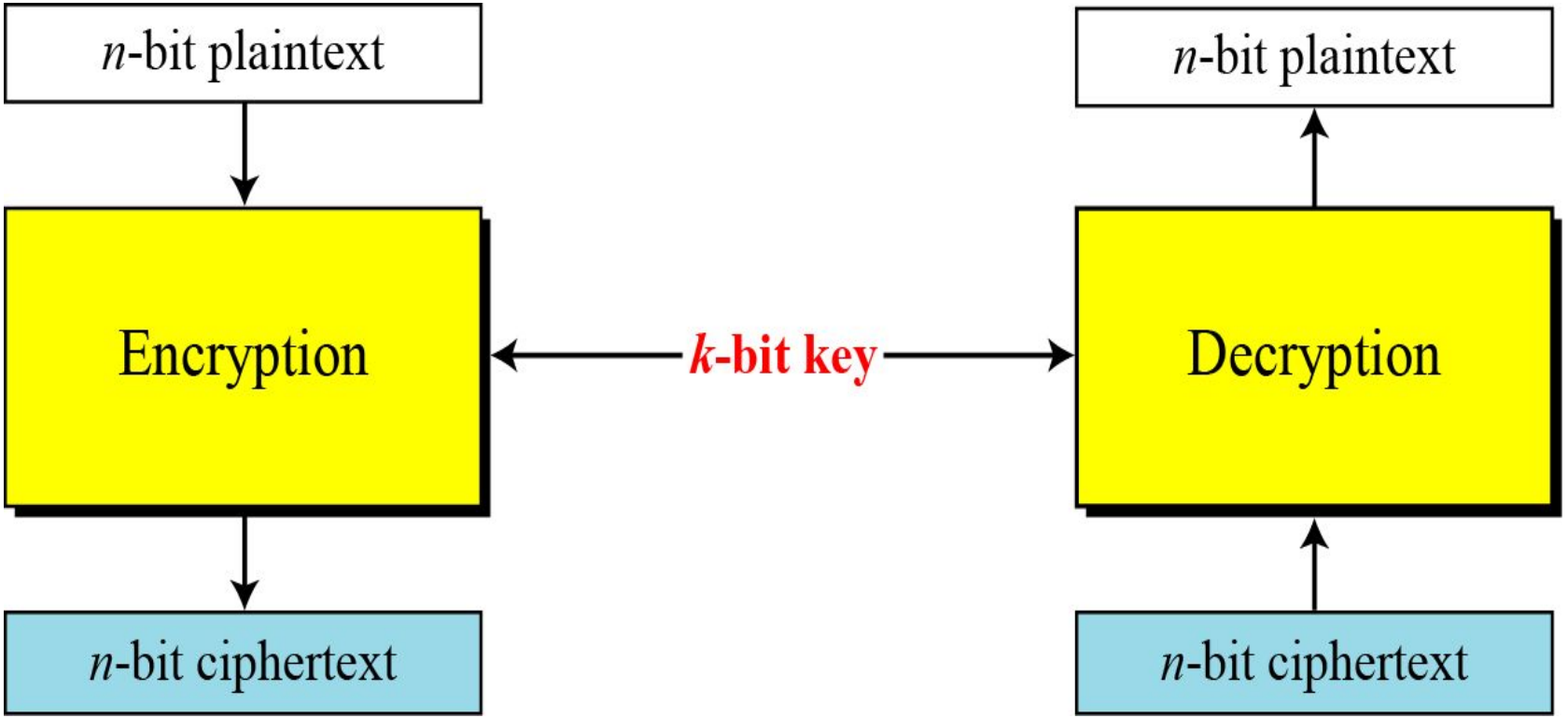
Symmetric Key Cryptography

Modern Block Ciphers

A symmetric-key modern block cipher encrypts an n-bit block of plaintext or decrypts an n-bit block of ciphertext. The encryption or decryption algorithm uses a k-bit key.


n-bit plaintext
Encryption
n-bit ciphertext
k-bit key
n-bit plaintext
Decryption
n-bit ciphertext

5.1.3 Components of a Modern Block Cipher

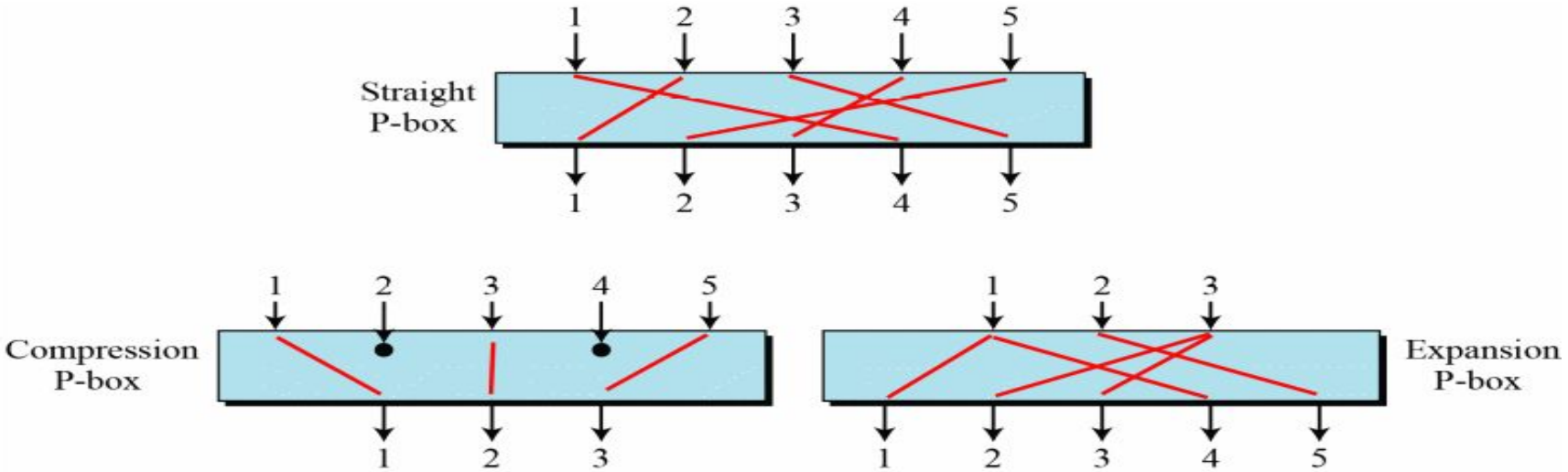
Modern block ciphers normally are keyed substitution ciphers in which the key allows only partial mappings from the possible inputs to the possible outputs.

P-Boxes

A P-box (permutation box) parallels the traditional transposition cipher for characters. It transposes bits.

5.1.3 Continued

Figure 5.4 *Three types of P-boxes*



S-Box

An S-box (substitution box) can be thought of as a miniature substitution cipher.

Note

An S-box is an $m \times n$ substitution unit, where m and n are not necessarily the same.

Example 5.8

In an S-box with three inputs and two outputs, we have

$$y_1 = x_1 \oplus x_2 \oplus x_3 \qquad y_2 = x_1$$

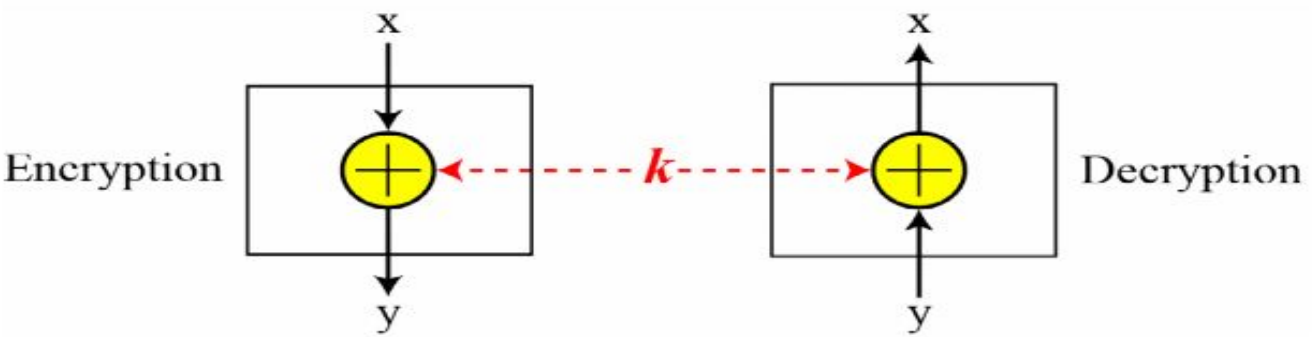
The S-box is linear because $a_{1,1} = a_{1,2} = a_{1,3} = a_{2,1} = 1$ and $a_{2,2} = a_{2,3} = 0$. The relationship can be represented by matrices, as shown below:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Exclusive-Or

An important component in most block ciphers is the exclusive-or operation.

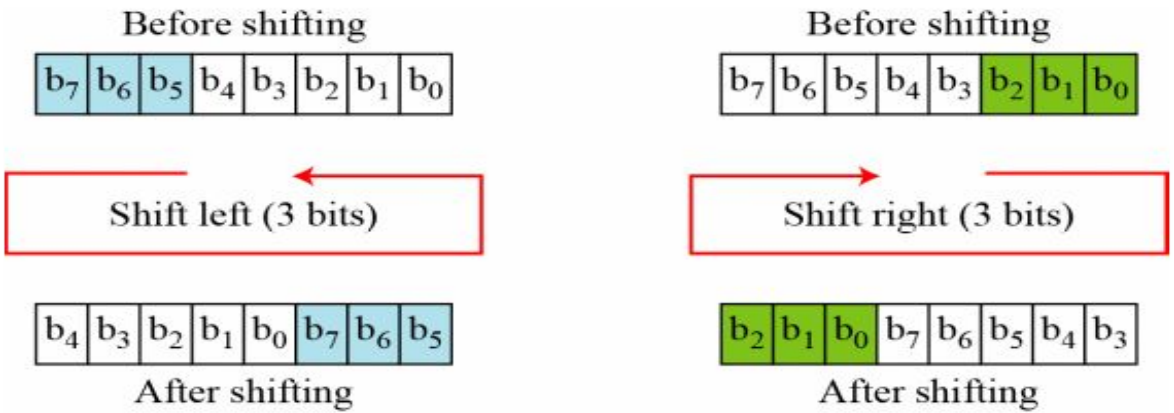
Figure 5.9 *Invertibility of the exclusive-or operation*



Circular Shift

Another component found in some modern block ciphers is the circular shift operation.

Figure 5.10 *Circular shifting an 8-bit word to the left or right*



Product Ciphers

- Diffusion
- Confusion
- Rounds

5.1.4 Continued

Figure 5.13 *A product cipher made of two rounds*

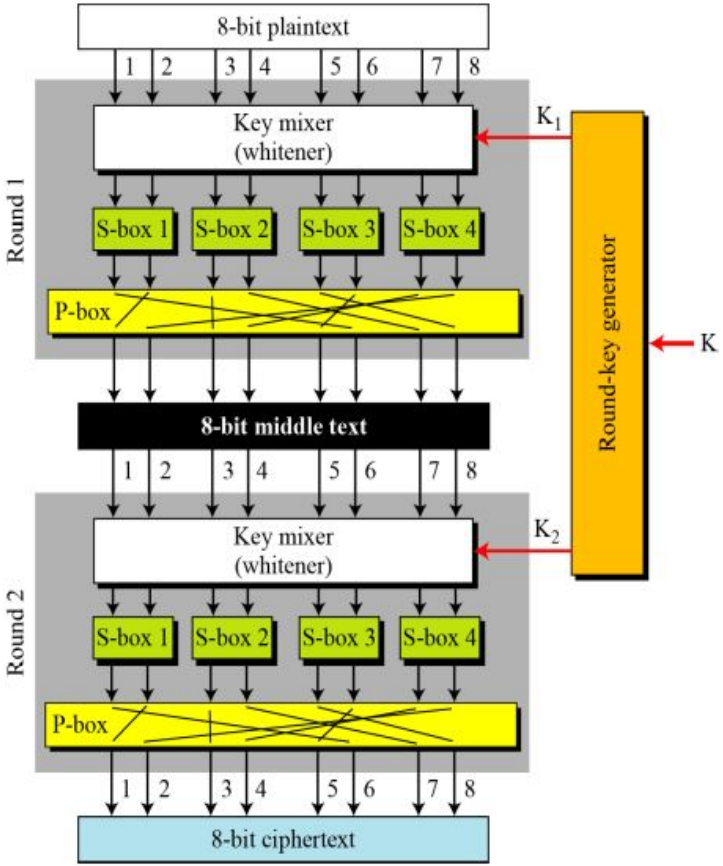
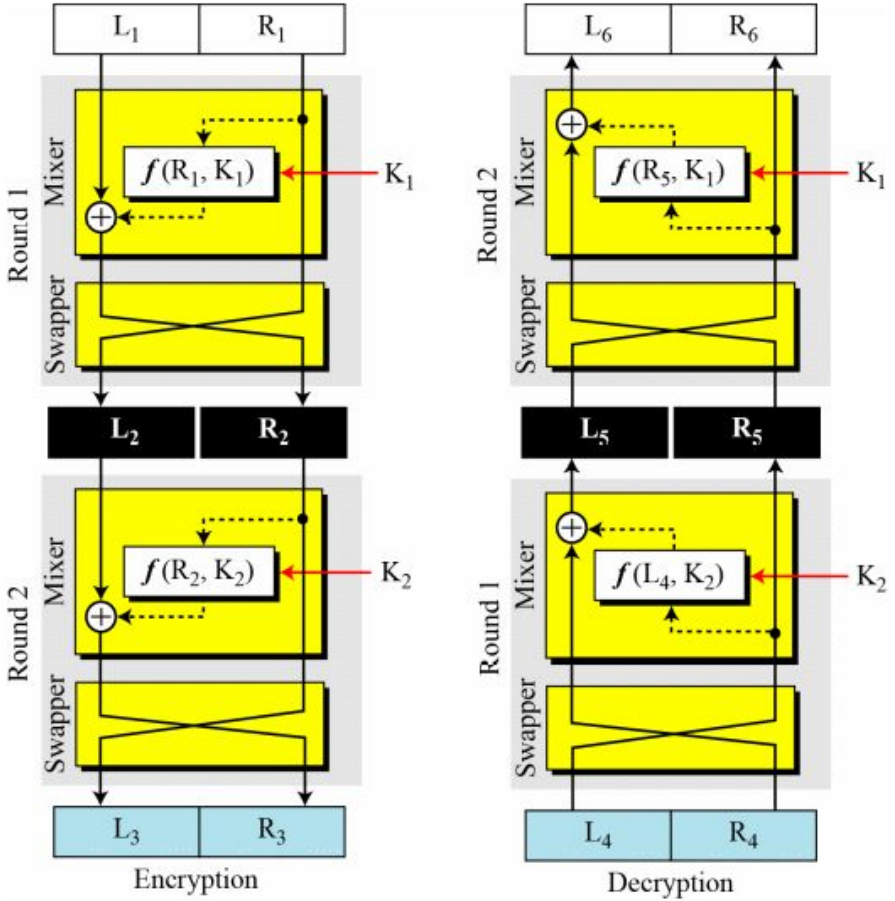


Figure 5.17 *Final design of a Feistel cipher with two rounds*

Fiestal and Non- fiestal



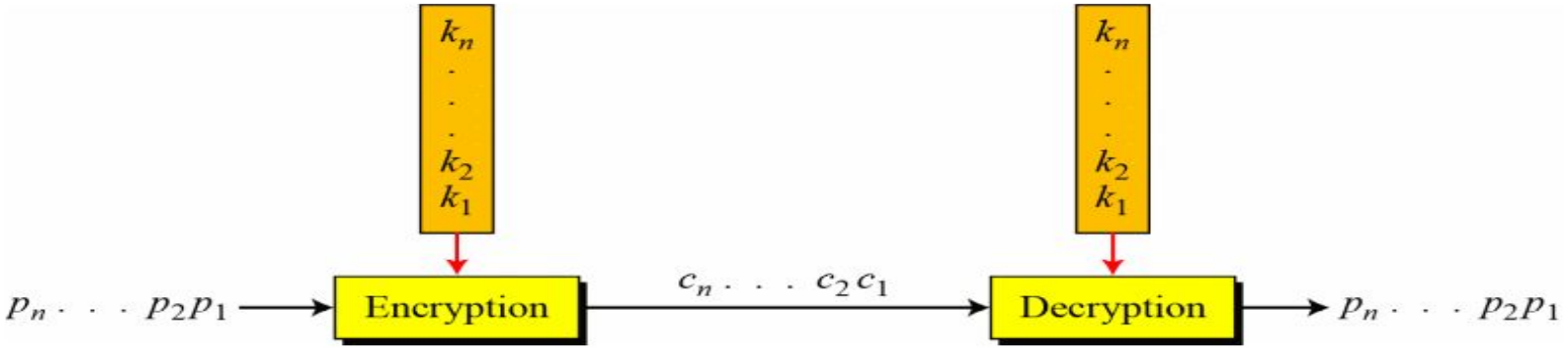
Cryptanalysis

- Differential (chosen P.T.)
- Linear (known P.T.)

Modern Stream Ciphers

5.2 Continued

Figure 5.20 *Stream cipher*



Note

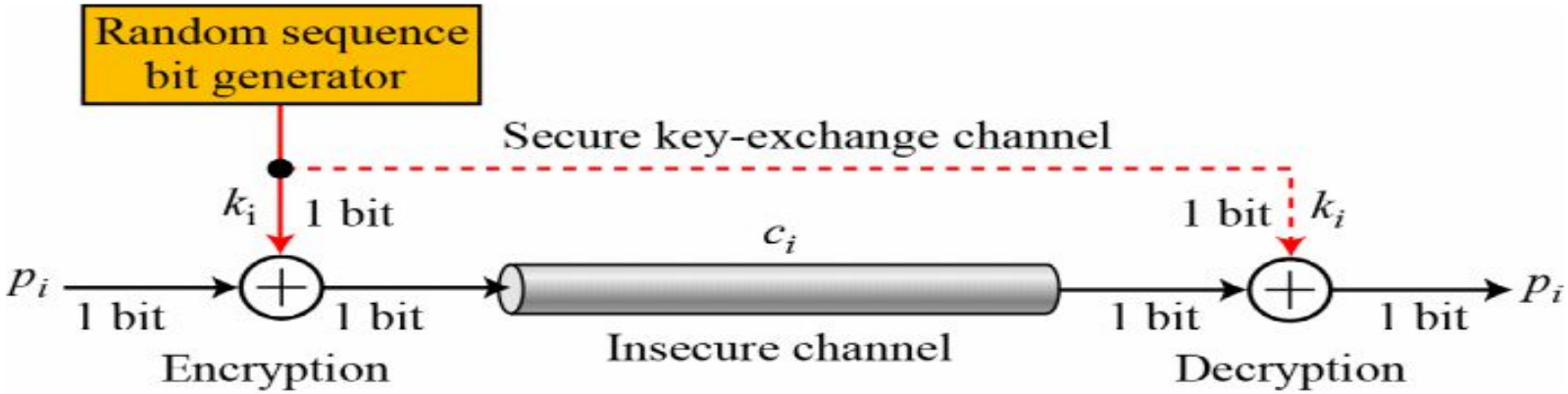
In a modern stream cipher, each *r*-bit word in the plaintext stream is enciphered using an *r*-bit word in the key stream to create the corresponding *r*-bit word in the ciphertext stream.

5.2.1 Synchronous Stream Ciphers

Note

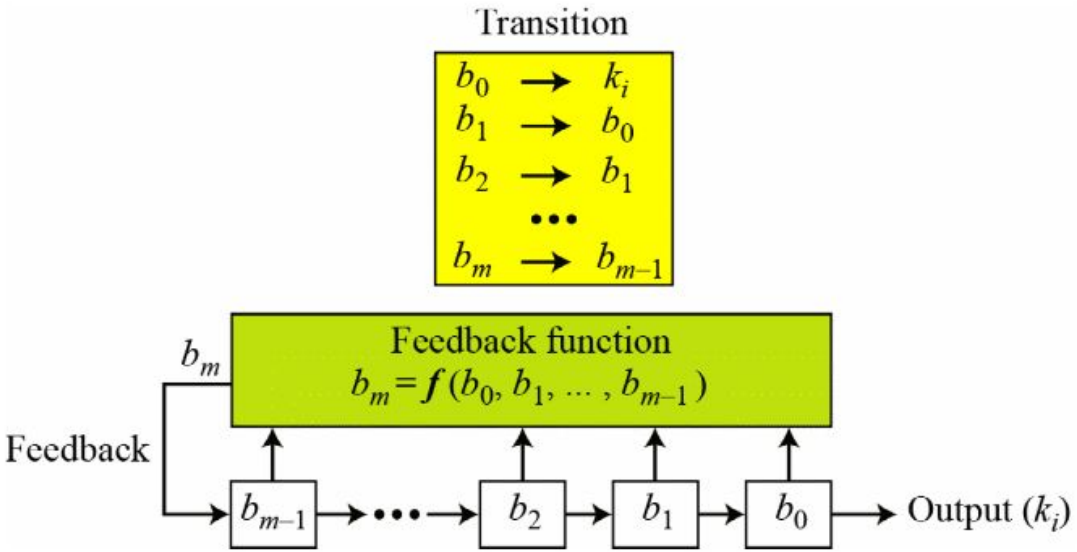
In a synchronous stream cipher the key is independent of the plaintext or ciphertext.

Figure 5.22 *One-time pad*



5.2.1 Continued

Figure 5.23 *Feedback shift register (FSR)*



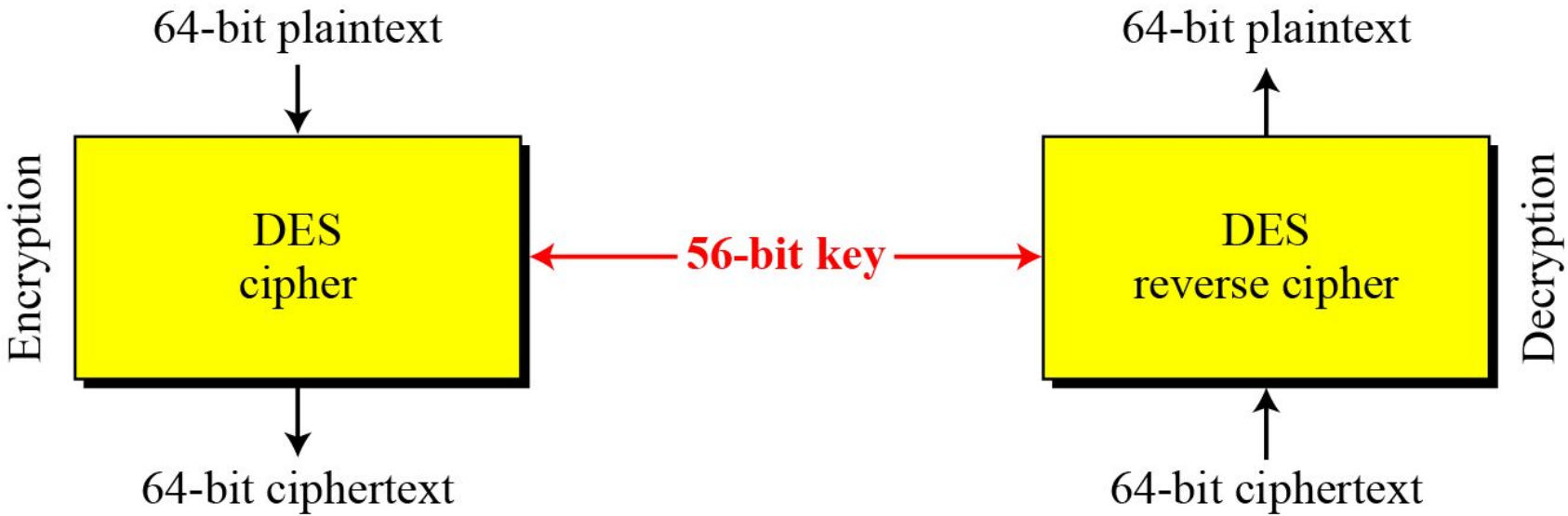
5.2.2 Nonsynchronous Stream Ciphers

In a nonsynchronous stream cipher, each key in the key stream depends on previous plaintext or ciphertext.

Note

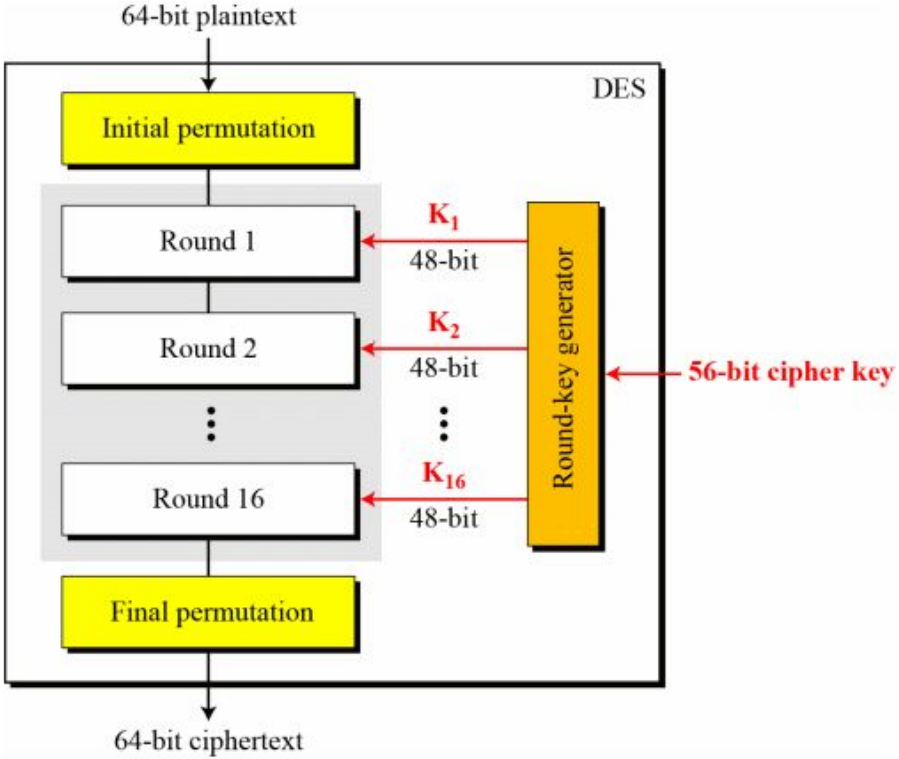
In a nonsynchronous stream cipher, the key depends on either the plaintext or ciphertext.

Data Encryption Standard (DES)



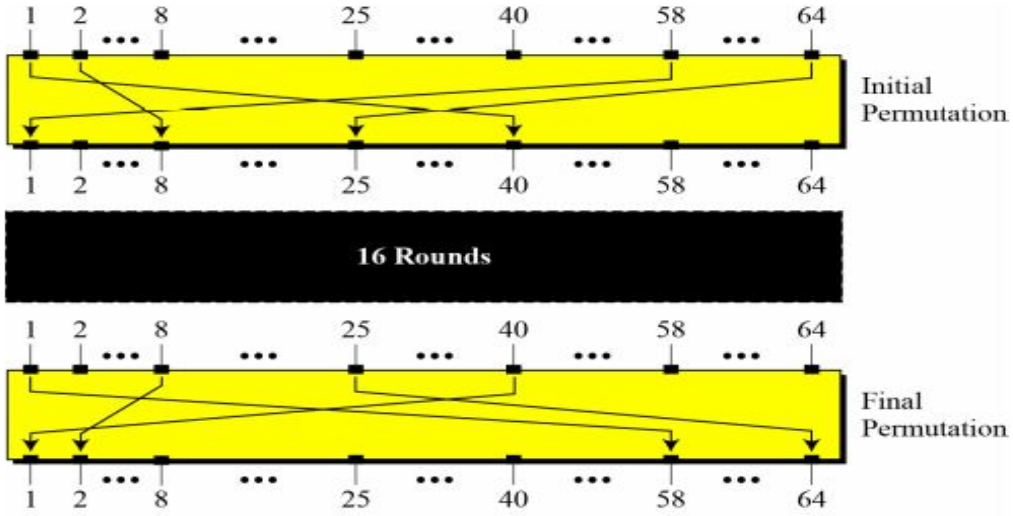
6-2 Continue

Figure 6.2 *General structure of DES*



6.2.1 Initial and Final Permutations

Figure 6.3 *Initial and final permutation steps in DES*



6.2.2 Rounds

- ***DES uses 16 rounds. Each round of DES is a Feistel cipher.***

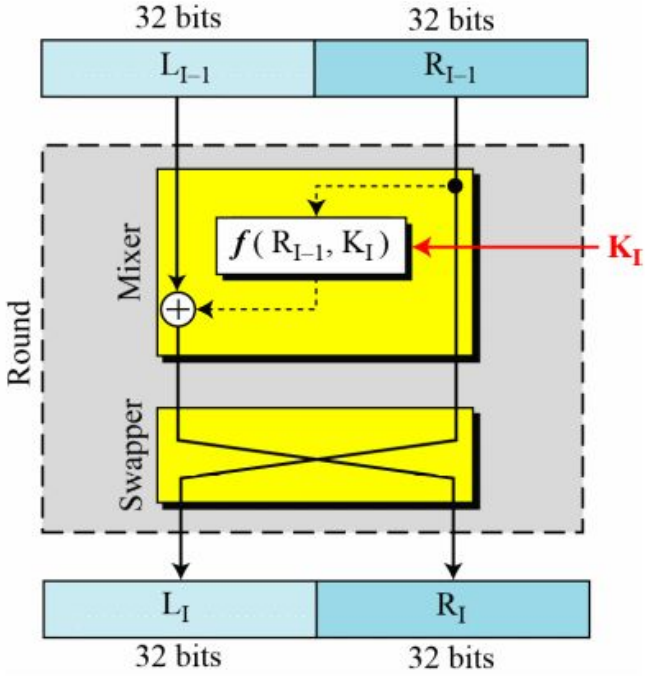


Figure 6.4
A round in DES (encryption site)

6.2.2 Continued

DES Function

The heart of DES is the DES function. The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output.

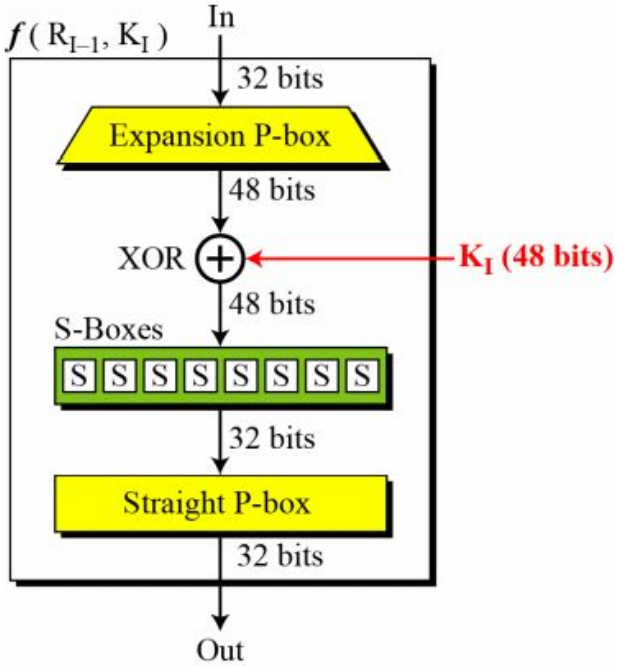
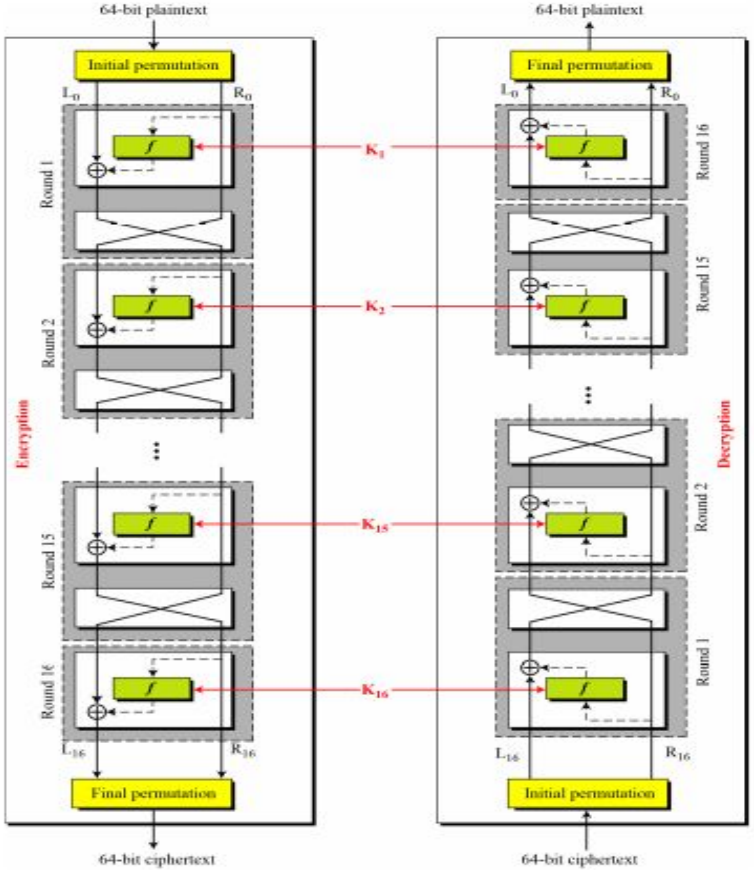


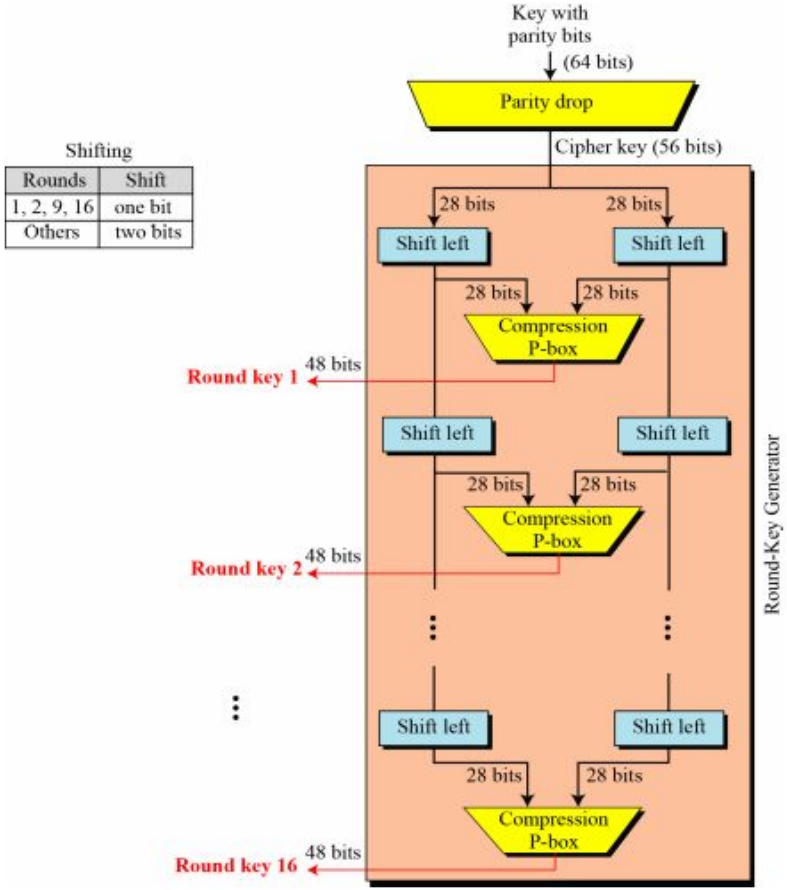
Figure 6.5
DES function

6.2.3 Continued

Figure 6.9 DES cipher and reverse cipher for the first approach



6.2.3 Continued



Shifting

Rounds	Shift
1, 2, 9, 16	one bit
Others	two bits

Figure 6.10
Key generation

Avalanche effect

Completeness effect

Differential cryptanalysis- designed S-boxes and chose 16 as the number of rounds to make DES specifically resistant to this type of attack.

Linear cryptanalysis- DES is more vulnerable to linear cryptanalysis than to differential cryptanalysis. S-boxes are not very resistant to linear cryptanalysis. It has been shown that DES can be broken using 243 pairs of known plaintexts. However, from the practical point of view, finding so many pairs is very unlikely.

Multiple DES.

AES

7.1.3 Rounds.

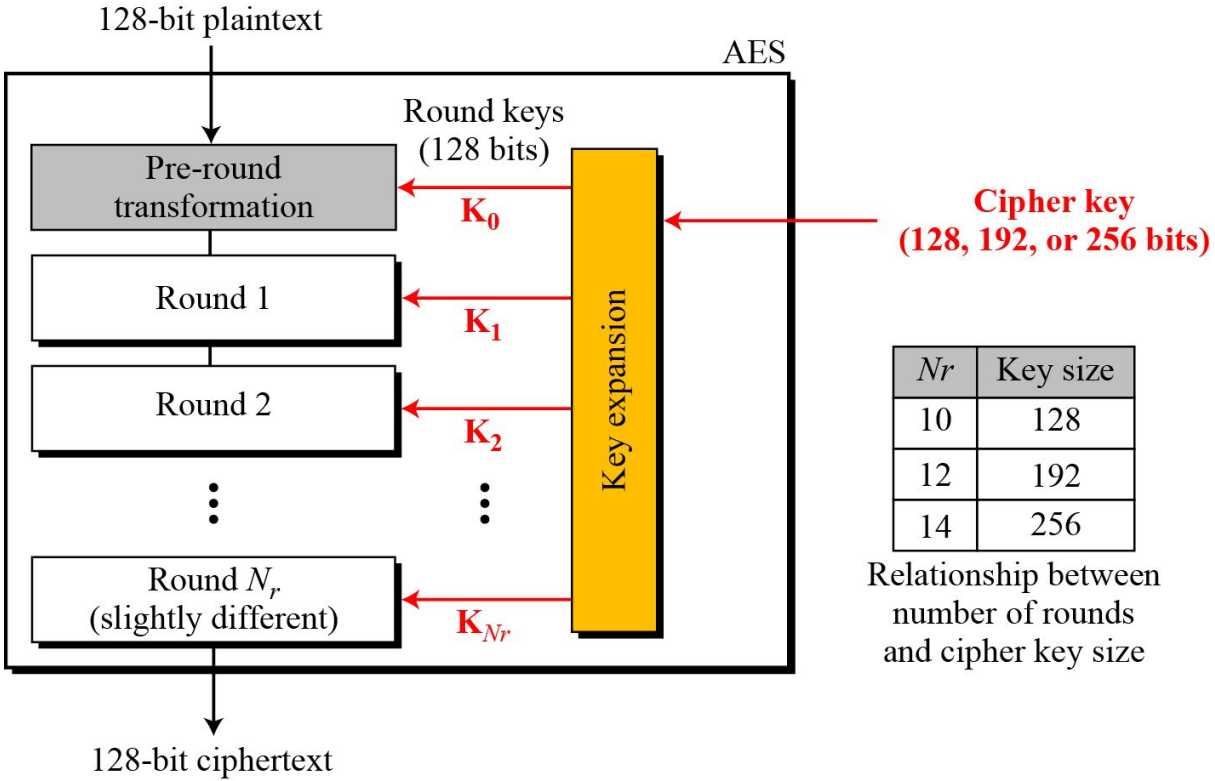
AES is a non-Feistel cipher that encrypts and decrypts a data block of 128 bits. It uses 10, 12, or 14 rounds. The key size, which can be 128, 192, or 256 bits, depends on the number of rounds.

Note

AES has defined three versions, with 10, 12, and 14 rounds.
Each version uses a different cipher key size (128, 192, or 256), but the round keys are always 128 bits.

7.1.3 Continue

Figure 7.1 *General design of AES encryption cipher*

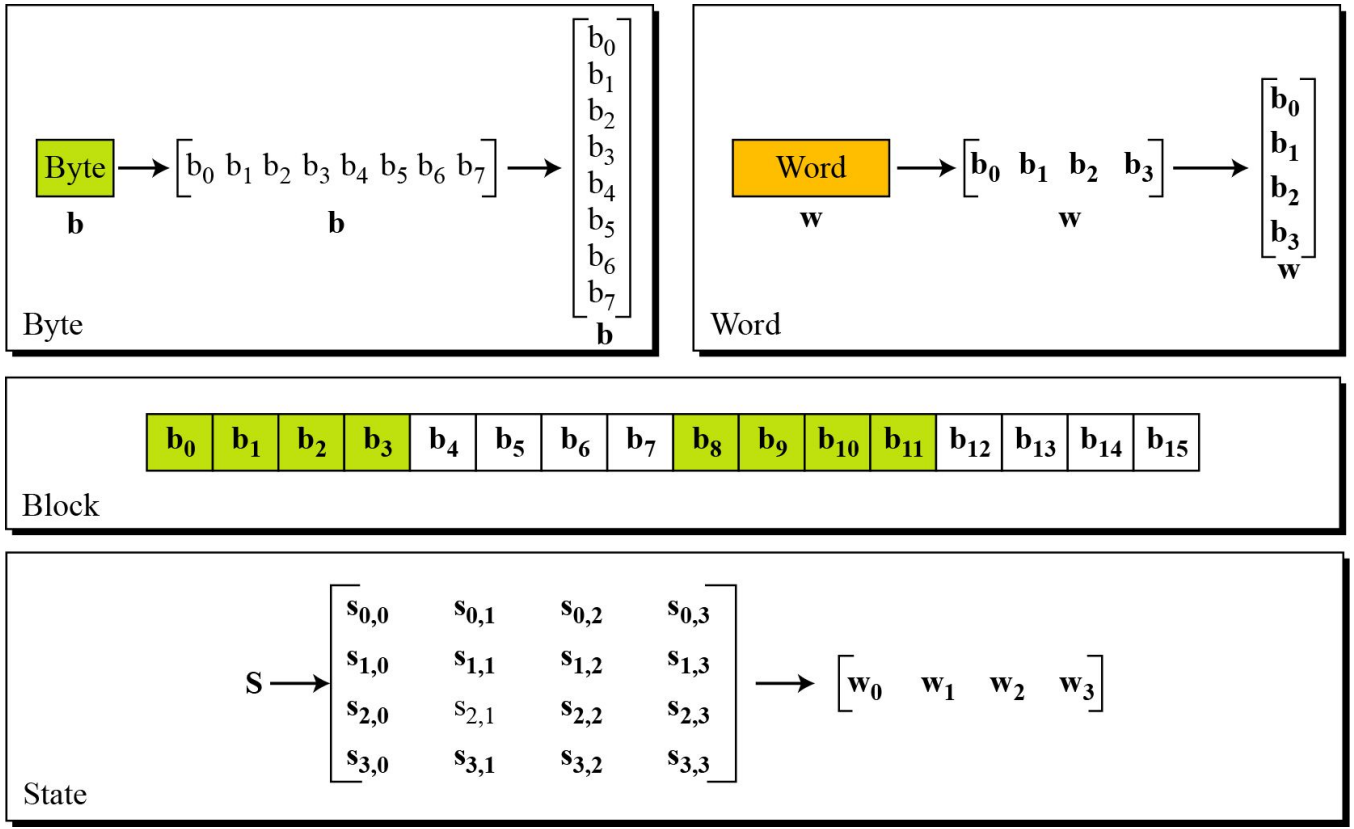


Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

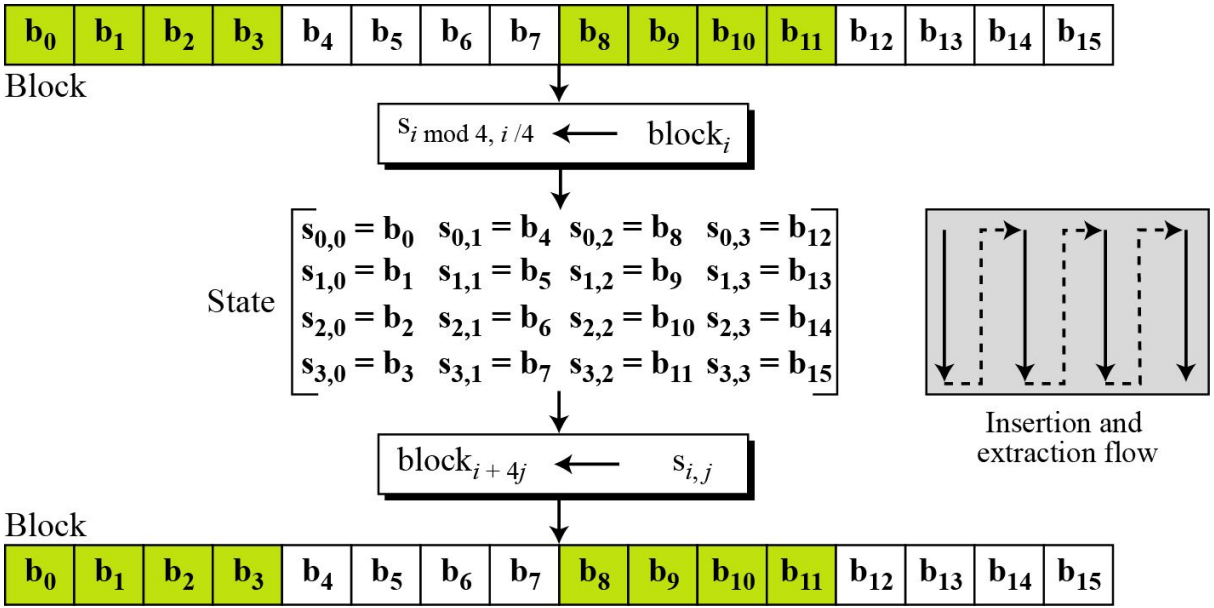
7.1.4 Data Units.

Figure 7.2 *Data units used in AES*



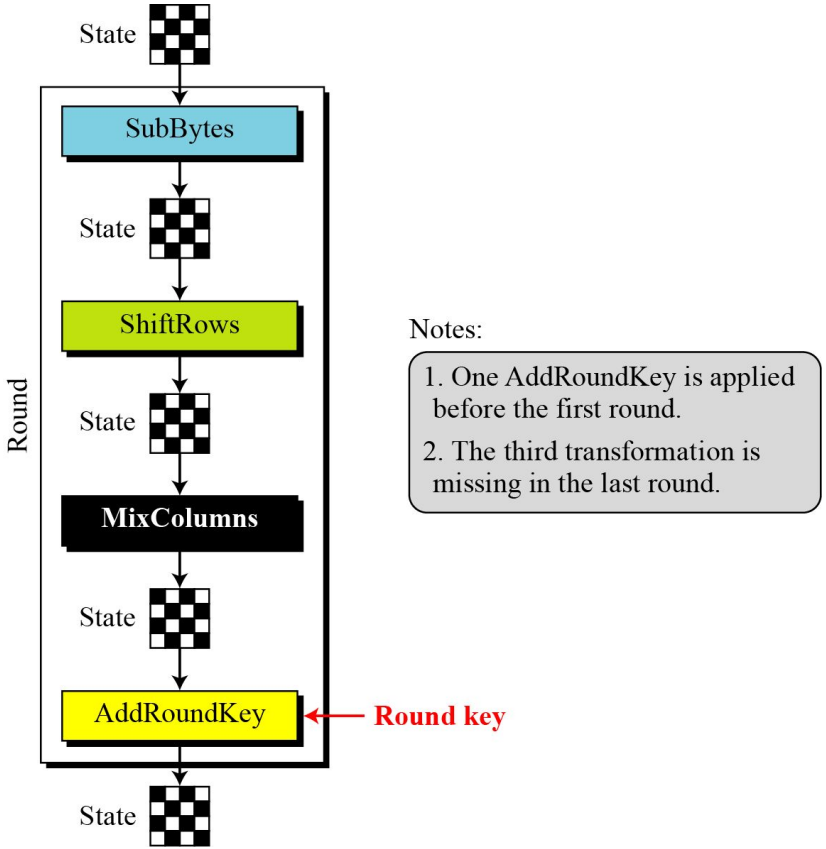
7.1.4 Continue

Figure 7.3 *Block-to-state and state-to-block transformation*



7.1.5 Structure of Each Round

Figure 7.5 *Structure of each round at the encryption site*



7-2 TRANSFORMATIONS

To provide security, AES uses four types of transformations: substitution, permutation, mixing, and key-adding.

Topics discussed in this section:

Substitution

AES, like DES, uses substitution. AES uses two invertible transformations.

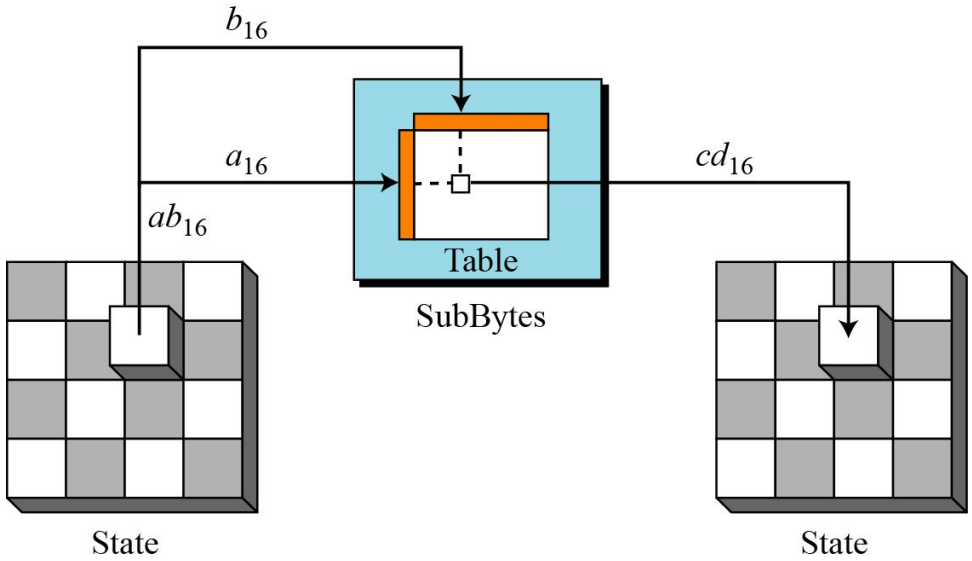
SubBytes

The first transformation, SubBytes, is used at the encryption site. To substitute a byte, we interpret the byte as two hexadecimal digits.

The SubBytes operation involves 16 independent byte-to-byte transformations.

7.2.1 Continue

Figure 7.6 *SubBytes transformation*



7.2.1 Continue

Table 7.1 *SubBytes transformation table*

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
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

7.2.1 Continue

Table 7.1 *SubBytes transformation table (continued)*

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
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	CB	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

InvSubBytes

Table 7.2 *InvSubBytes transformation table*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

7.2.1 Continue

InvSubBytes (Continued)

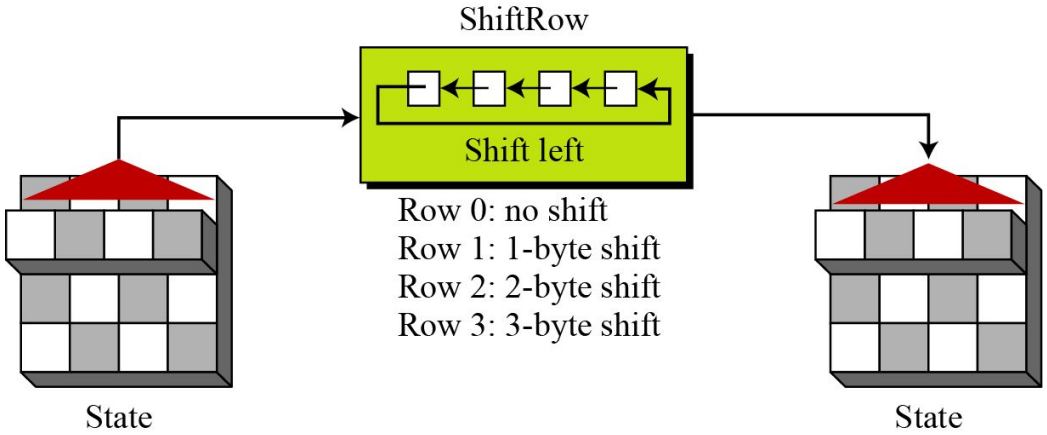
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	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

Permutation

Another transformation found in a round is shifting, which permutes the bytes.

ShiftRows

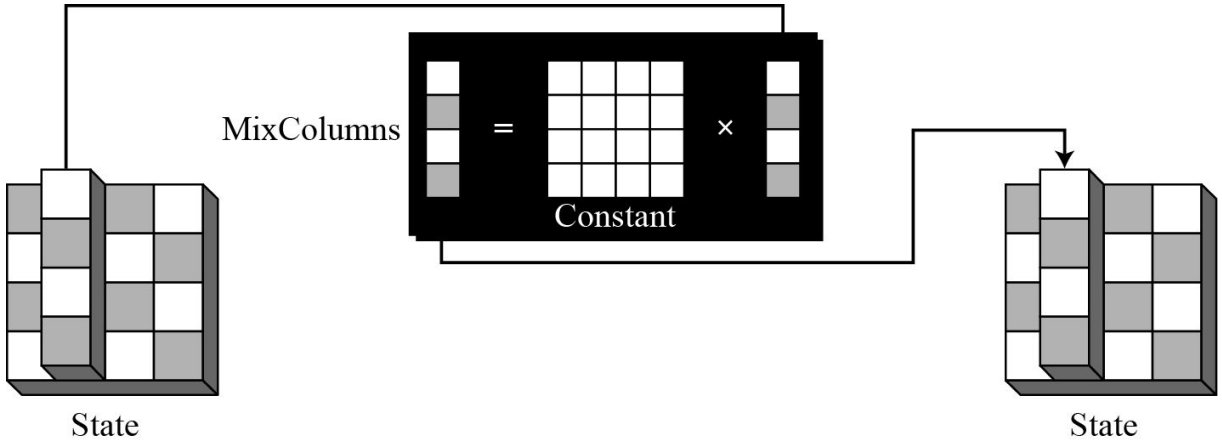
In the encryption, the transformation is called ShiftRows.



MixColumns

The MixColumns transformation operates at the column level; it transforms each column of the state to a new column.

Figure 7.13 *MixColumns transformation*



Mixing

We need an interbyte transformation that changes the bits inside a byte, based on the bits inside the neighboring bytes. We need to mix bytes to provide diffusion at the bit level.

Figure 7.11 *Mixing bytes using matrix multiplication*

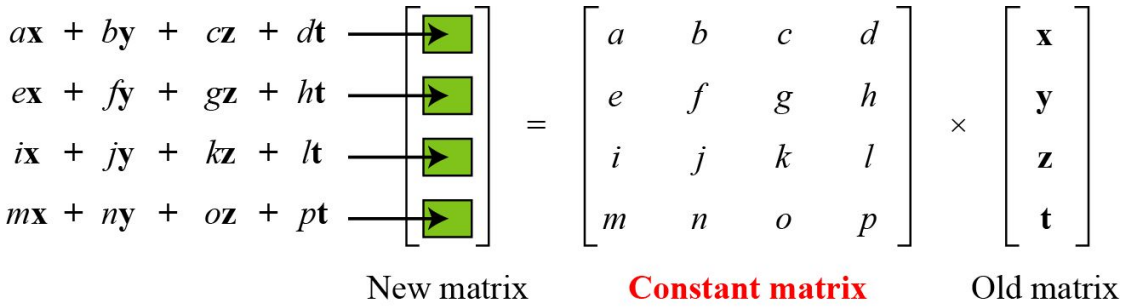
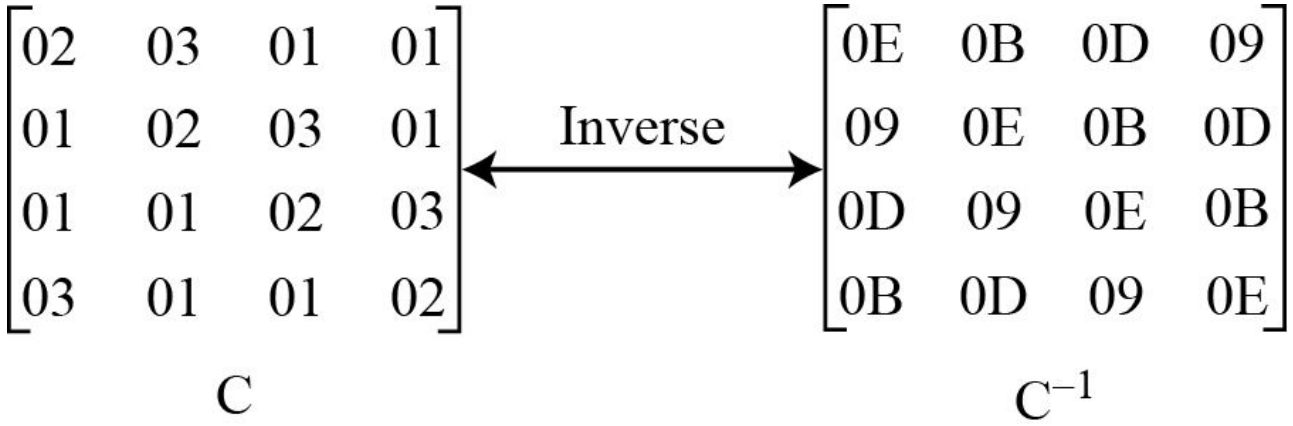


Figure 7.12 *Constant matrices used by MixColumns and InvMixColumns*



InvMixColumns

The InvMixColumns transformation is basically the same as the MixColumns transformation.

Note

The MixColumns and InvMixColumns transformations are inverses of each other.

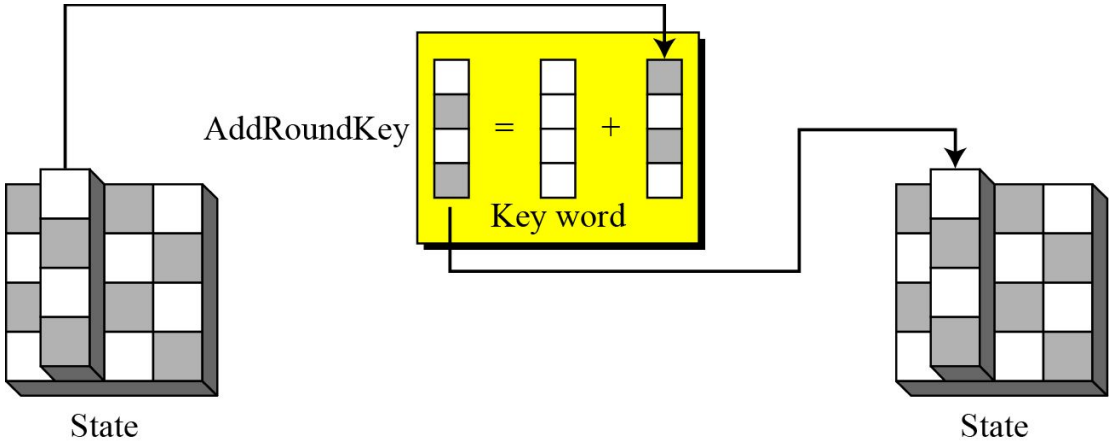
7.2.4 Key Adding

AddRoundKey

AddRoundKey proceeds one column at a time. AddRoundKey adds a round key word with each state column matrix; the operation in AddRoundKey is matrix addition.

The AddRoundKey transformation is the inverse of itself.

Figure 7.15 *AddRoundKey transformation*

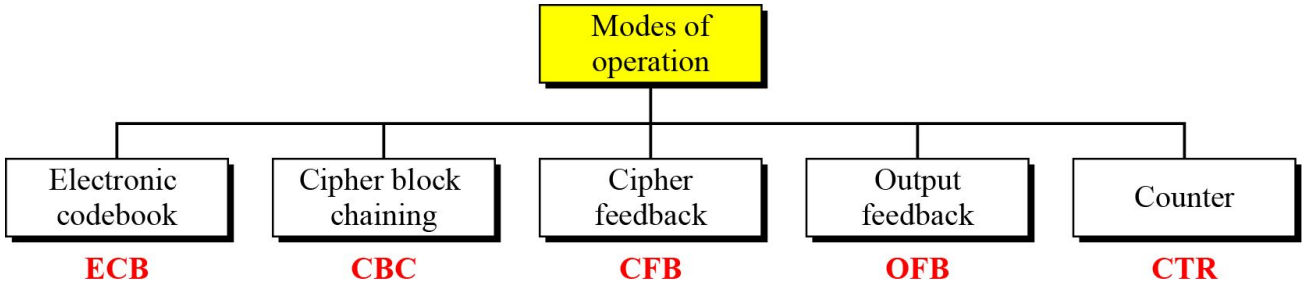


Algorithm 7.4 *Pseudocode for AddRoundKey transformation*

```
AddRoundKey (S)
{
    for (c = 0 to 3)
        s_c ← s_c ⊕ w_round + 4c
}
```

8-1 Continued

Figure 8.1 *Modes of operation*

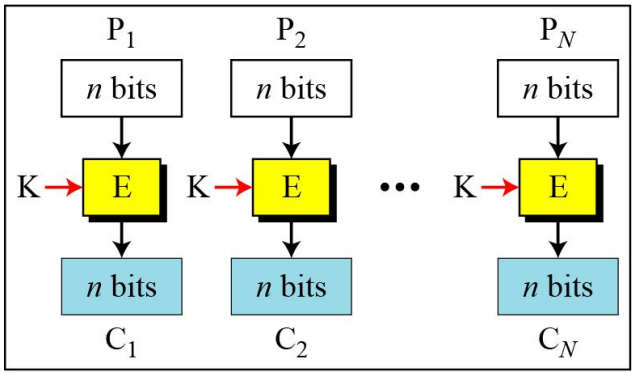


8.1.1 Electronic Codebook (ECB) Mode

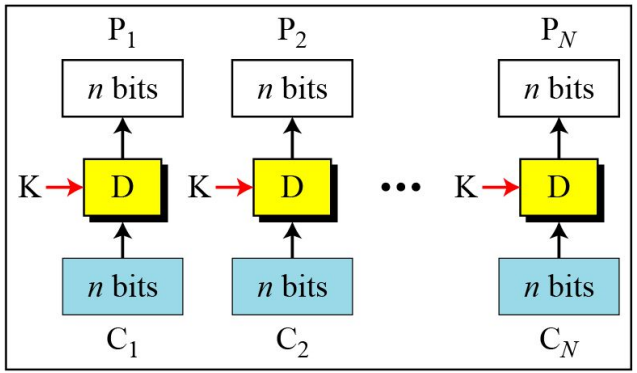
The simplest mode of operation is called the electronic codebook (ECB) mode.

Encryption: $C_i = E_K(P_i)$ Decryption: $P_i = D_K(C_i)$

E: Encryption D: Decryption
P_i: Plaintext block i C_i: Ciphertext block i
K: Secret key



Encryption



Decryption

- *Error Propagation*

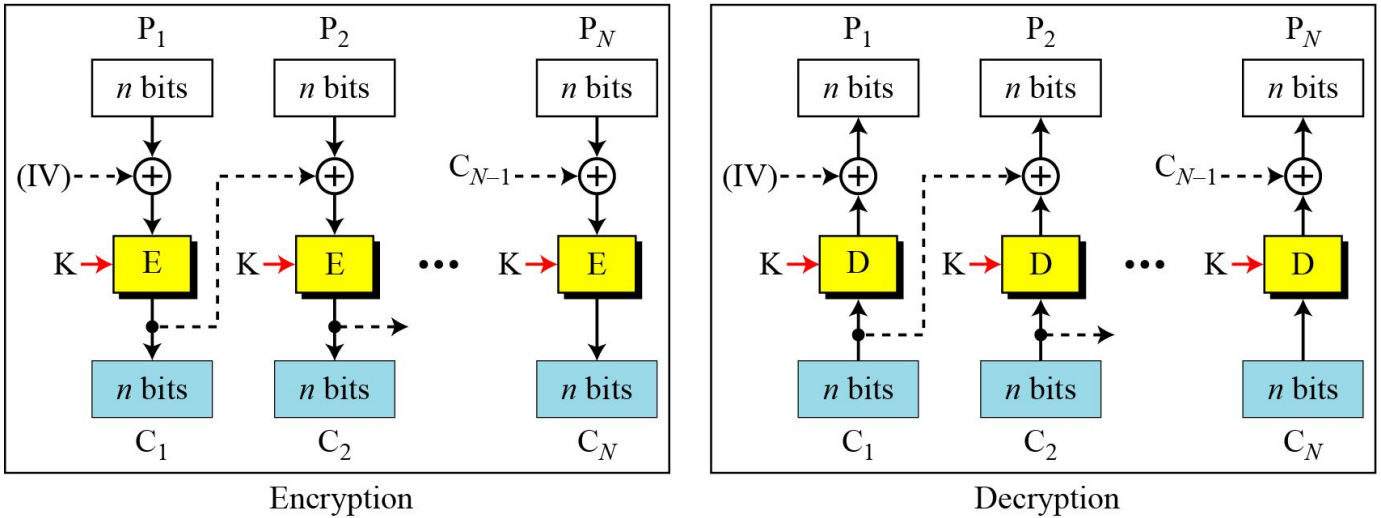
A single bit error in transmission can create errors in several in the corresponding block. However, the error does not have any effect on the other blocks.

8.1.2 Cipher Block Chaining (CBC) Mode

In CBC mode, each plaintext block is exclusive-ored with the previous ciphertext block before being encrypted.

E: Encryption
P_i: Plaintext block i
K: Secret key

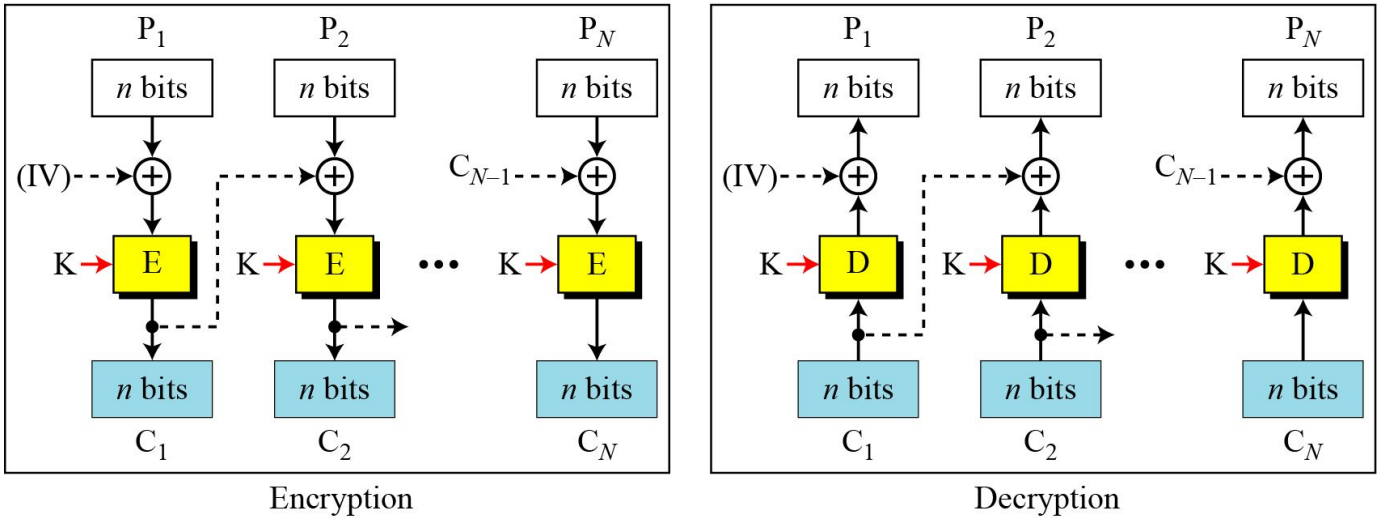
D : Decryption
C_i : Ciphertext block i
IV: Initial vector (C_0)



8.1.2 Continued

E: Encryption
P_i: Plaintext block i
K: Secret key

D : Decryption
C_i : Ciphertext block i
IV: Initial vector (C_0)



Encryption

Decryption

Encryption:

$C_0 = IV$

$C_i = E_K (P_i \oplus C_{i-1})$

Decryption:

$C_0 = IV$

$P_i = D_K (C_i) \oplus C_{i-1}$

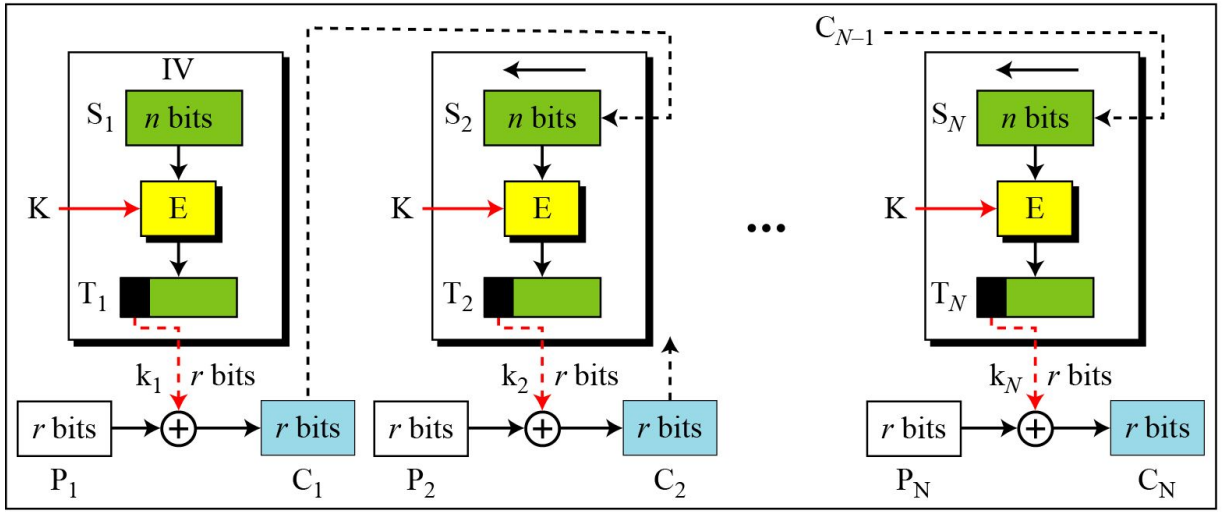
Error Propagation

In CBC mode, a single bit error in ciphertext block C_j during transmission may create error in most bits in plaintext block P_j during decryption + 1 bit in the next block

8.1.3 Cipher Feedback (CFB) Mode

plaintext or ciphertext block sizes are smaller

E : Encryption
D : Decryption
S_i: Shift register
P_i: Plaintext block i
C_i: Ciphertext block i
T_i: Temporary register
K: Secret key
IV: Initial vector (S_1)



Encryption

Encryption: $C_i = P_i \oplus \text{SelectLeft}_r \{E_K [\text{ShiftLeft}_r (S_{i-1}) \mid C_{i-1})]\}$

Decryption: $P_i = C_i \oplus \text{SelectLeft}_r \{E_K [\text{ShiftLeft}_r (S_{i-1}) \mid C_{i-1})]\}$

- *Error Propagation*

A single bit error in transmission creates: 1 bit error in the corresponding block + most of the bits in the successive plaintext blocks until the shift registrar is totally refreshed.

8.2.1 RC4

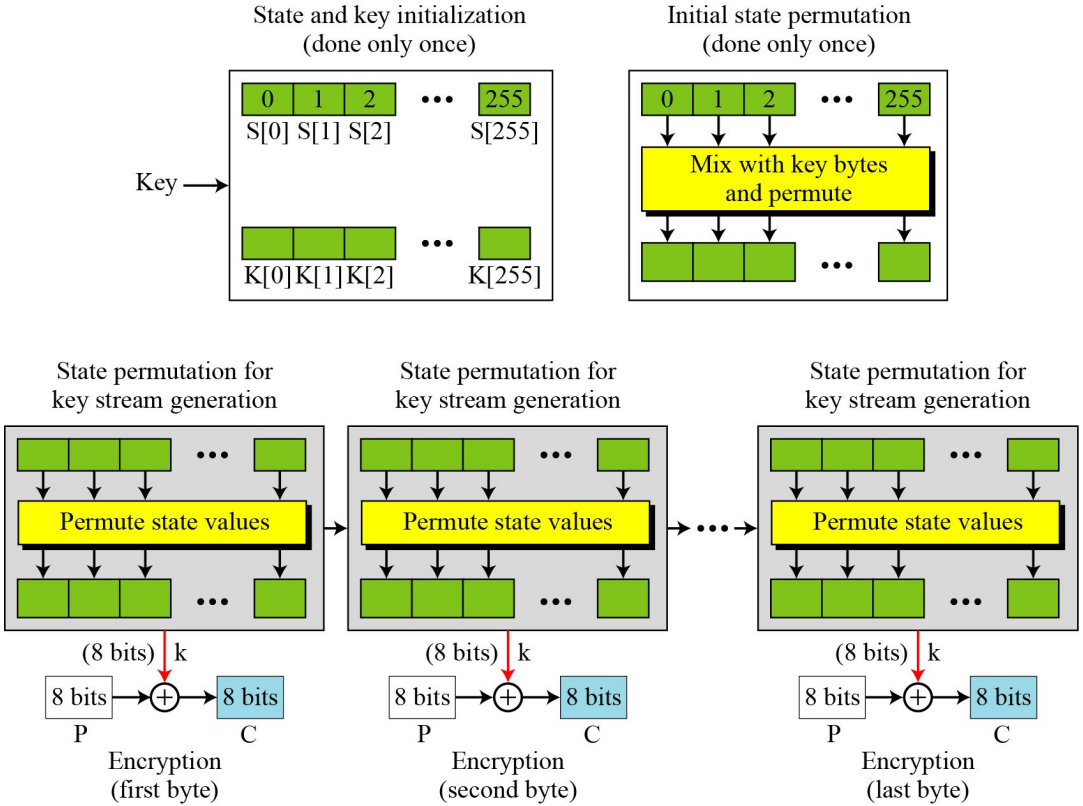
RC4 is a byte-oriented stream cipher in which a byte (8 bits) of a plaintext is exclusive-ored with a byte of key to produce a byte of a ciphertext.

State

RC4 is based on the concept of a state.

S[0] S[1] S[2] ··· S[255]

Figure 8.10 *The idea of RC4 stream cipher*



8.2.1 Continued

Initialization

Initialization is done in two steps:

```
for (i = 0 to 255)
{
    S[i]  ← i
    K[i]  ← Key [i mod KeyLength]
}
```

```
j ← 0
for (i = 0 to 255)
{
    j ← (j + S[i] + K[i]) mod 256
    swap (S[i] , S[j])
}
```

Key Stream Generation

The keys in the key stream are generated, one by one.

```
i  ← (i + 1) mod 256
j  ← (j + S[i]) mod 256
swap (S [i] , S[j])
k  ← S [(S[i] + S[j]) mod 256]
```

8.2.1 Continued

Algorithm

Algorithm 8.6 *Encryption algorithm for RC4*

```
RC4_Encryption (K)
{
    // Creation of initial state and key bytes
    for (i = 0 to 255)
    {
        S[i] ← i
        K[i] ← Key [i mod KeyLength]
    }
    // Permuting state bytes based on values of key bytes
    j ← 0
    for (i = 0 to 255)
    {
        j ← (j + S[i] + K[i]) mod 256
        swap (S[i] , S[j])
    }
```

8.2.1 Continued

Algorithm Continued

```
    // Continuously permuting state bytes, generating keys, and encrypting
    i ← 0
    j ← 0
    while (more byte to encrypt)
    {
        i ← (i + 1) mod 256
        j ← (j + S[i]) mod 256
        swap (S [i] , S[j])
        k ← S [(S[i] + S[j]) mod 256]
        // Key is ready, encrypt
        input P
        C ← P ⊕ k
        output C
    }
}
```

Thank You