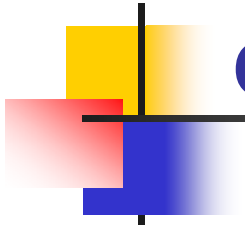
Chapter 8

Encipherment Using Modern Symmetric-Key Ciphers



Chapter 18

Objectives

- To show how modern standard ciphers, such as DES or AES, can be used to encipher long messages.
- To discuss five modes of operation designed to be used with modern block ciphers.
- To define which mode of operation creates stream ciphers out of the underlying block ciphers.
- To discuss the security issues and the error propagation of different modes of operation.
- To discuss two stream ciphers used for real-time processing of data.

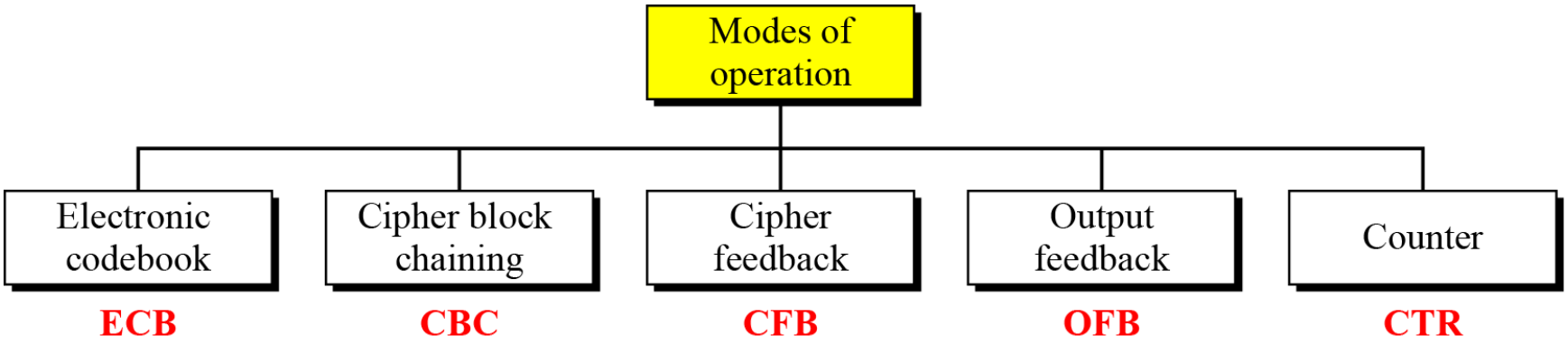
8-1 USE OF MODERN BLOCK CIPHERS

Symmetric-key encipherment can be done using modern block ciphers. Modes of operation have been devised to encipher text of any size employing either DES or AES.

Topics discussed in this section:

8-1 Continued

Figure 8.1 *Modes of operation*



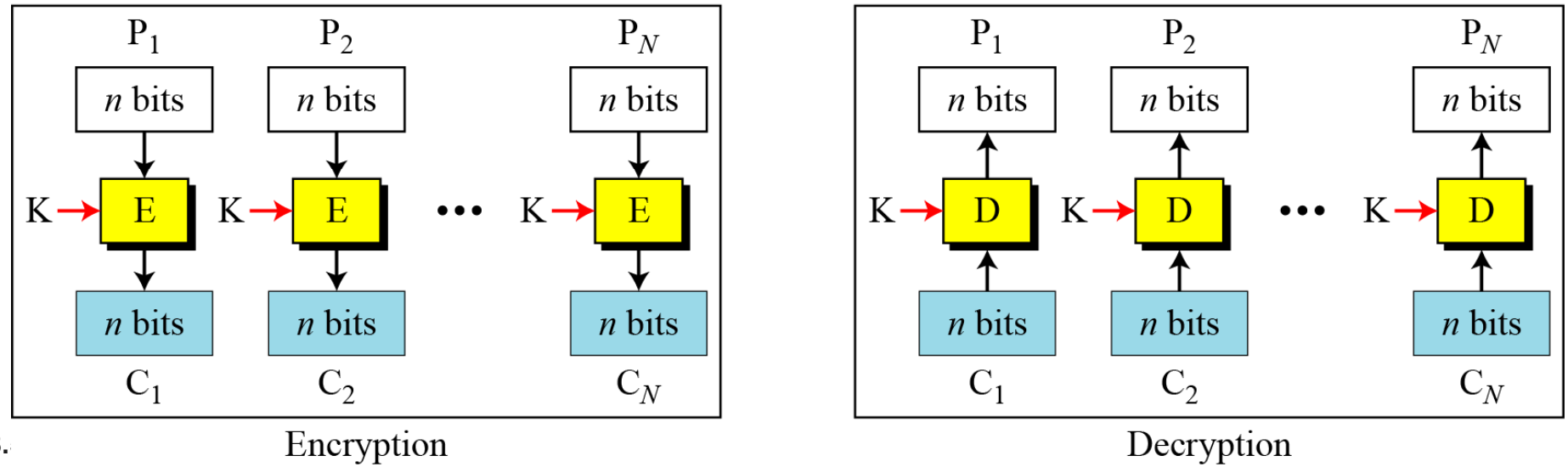
8.1.1 Electronic Codebook (ECB) Mode

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

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

Figure 8.2 *Electronic codebook (ECB) mode*

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



8.1.1 Continued

Example 8.1

It can be proved that each plaintext block at Alice's site is exactly recovered at Bob's site. Because encryption and decryption are inverses of each other,

$$\text{Encryption: } C_i = E_K(P_i) \qquad \text{Decryption: } P_i = D_K(C_i)$$

Example 8.2

This mode is called electronic codebook because one can precompile 2^K codebooks (one for each key) in which each codebook has 2^n entries in two columns. Each entry can list the plaintext and the corresponding ciphertext blocks. However, if K and n are large, the codebook would be far too large to precompile and maintain.

8.1.1 Continued

Example 8.3

Assume that Eve works in a company a few hours per month (her monthly payment is very low). She knows that the company uses several blocks of information for each employee in which the seventh block is the amount of money to be deposited in the employee's account. Eve can intercept the ciphertext sent to the bank at the end of the month, replace the block with the information about her payment with a copy of the block with the information about the payment of a full-time colleague. Each month Eve can receive more money than she deserves.

8.1.1 Continued

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.

Algorithm 8.1 *Encryption for ECB mode*

```
ECB_Encryption (K, Plaintext blocks)
{
    for (i = 1 to N)
    {
        C_i ← E_K (P_i)
    }
    return Ciphertext blocks
}
```

8.1.1 Continued

Ciphertext Stealing

A technique called ciphertext stealing (CTS) can make it possible to use ECB mode without padding. In this technique the last two plaintext blocks, P_{N-1} and P_N, are encrypted differently and out of order, as shown below, assuming that P_{N-1} has n bits and P_N has m bits, where $m \leq n$.

$$X = E_K(P_{N-1}) \rightarrow C_N = head_m(X)$$

$$Y = P_N \mid tail_{n-m}(X) \rightarrow C_{N-1} = E_K(Y)$$

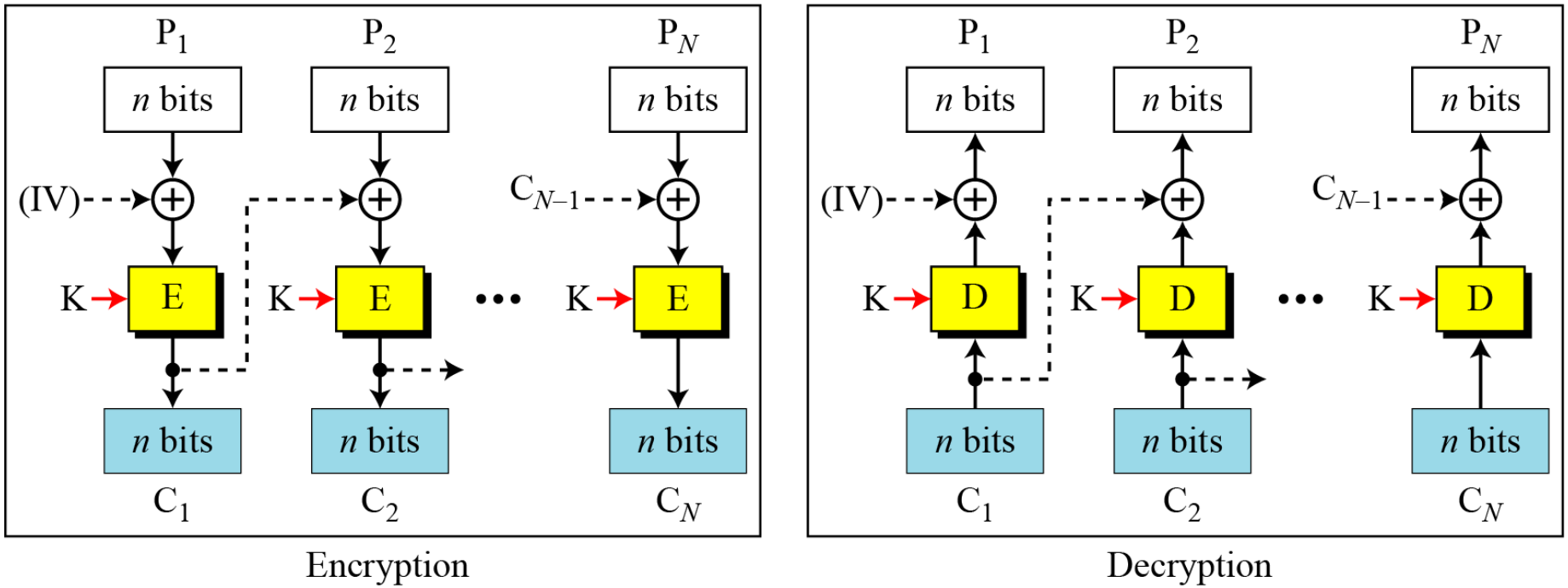
8.1.2 Cipher Block Chaining (CBC) Mode

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

Figure 8.3 *Cipher block chaining (CBC) mode*

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

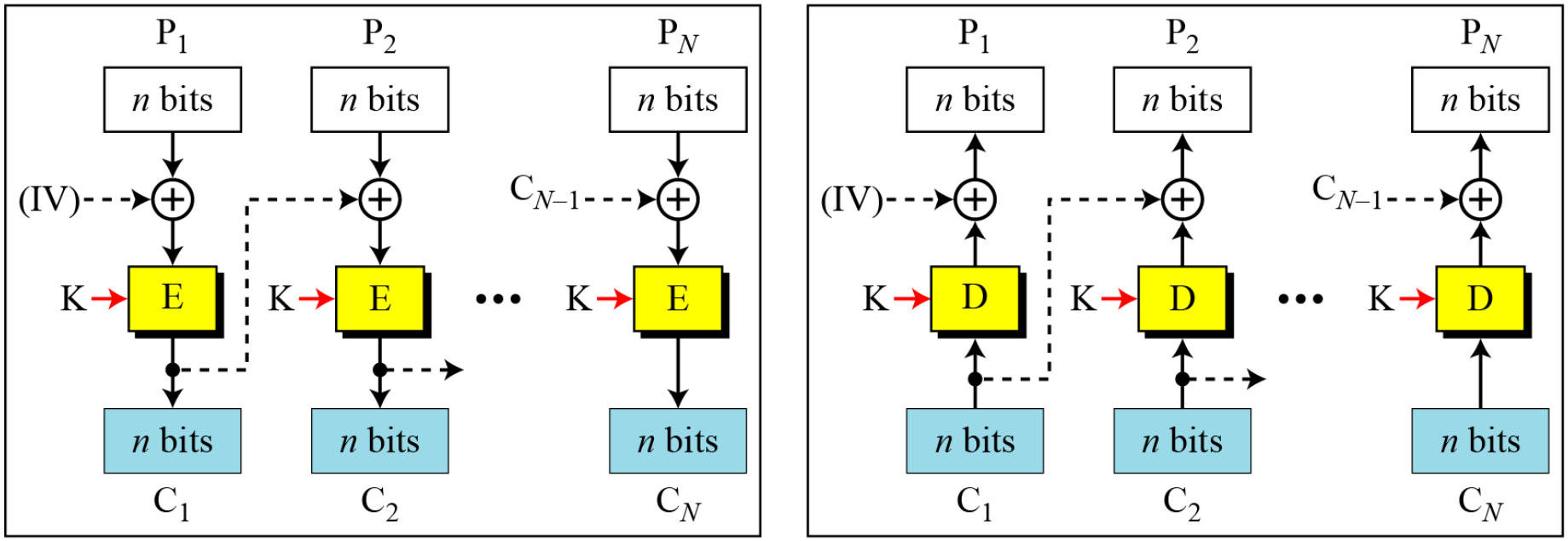
D : Decryption
C_i : Ciphertext block i
IV: Initial vector (C_0)



8.1.2 Continued

Figure 8.3 *Cipher block chaining (CBC) mode*

E: Encryption D : Decryption
P_i: Plaintext block *i* C_i : Ciphertext block *i*
K: Secret key IV: Initial vector (C_0)



Encryption:	**Decryption:**
$C_0 = IV$	$C_0 = IV$
$C_i = E_K (P_i \oplus C_{i-1})$	$P_i = D_K (C_i) \oplus C_{i-1}$

8.1.2 Continued

Example 8.4

It can be proved that each plaintext block at Alice's site is recovered exactly at Bob's site. Because encryption and decryption are inverses of each other,

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

Initialization Vector (IV)

The initialization vector (IV) should be known by the sender and the receiver.

8.1.2 Continued

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.

Algorithm 8.2 *Encryption algorithm for ECB mode*

```
CBC_Encryption (IV, K, Plaintext blocks)
{
    C0 ← IV
    for (i = 1 to N)
    {
        Temp ← Pi ⊕ Ci−1
        Ci ← EK (Temp)
    }
    return Ciphertext blocks
}
```

8.1.2 Continued

Ciphertext Stealing

The ciphertext stealing technique described for ECB mode can also be applied to CBC mode, as shown below.

$$\begin{array}{lcllcl} U = P_{N-1} \oplus C_{N-2} & \rightarrow & X = E_K(U) & \rightarrow & C_N & = head_m(X) \\ V = P_N \mid pad_{n-m}(0) & \rightarrow & Y = X \oplus V & \rightarrow & C_{N-1} & = E_K(Y) \end{array}$$

The head function is the same as described in ECB mode; the pad function inserts 0's.

8.1.3 Cipher Feedback (CFB) Mode

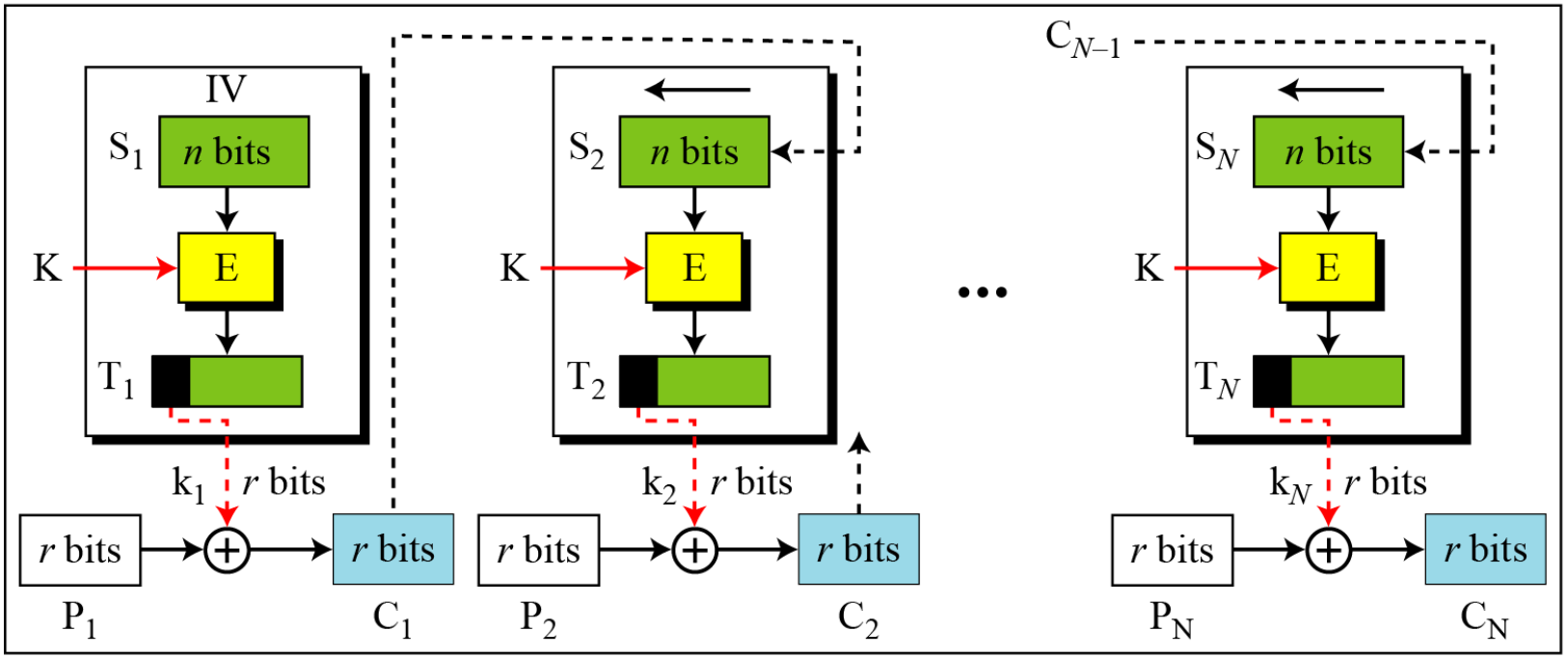
In some situations, we need to use DES or AES as secure ciphers, but the plaintext or ciphertext block sizes are to be smaller.

Figure 8.4 *Encryption in cipher feedback (CFB) mode*

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

D : Decryption
C_i: Ciphertext block i
IV: Initial vector (S_1)

S_i: Shift register
T_i: Temporary register



Encryption

8.1.3 Continued

Note

In CFB mode, encipherment and decipherment use the encryption function of the underlying block cipher.

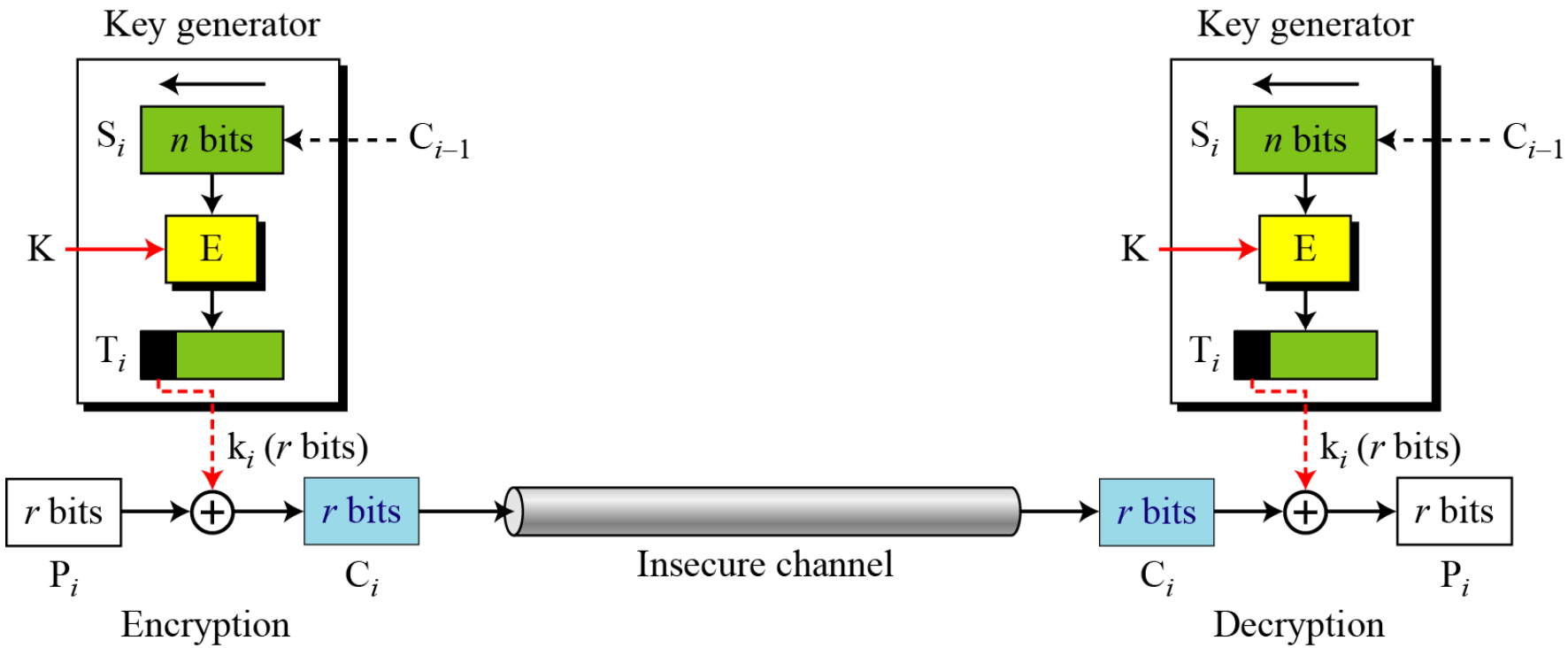
The relation between plaintext and ciphertext blocks is shown below:

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

8.1.3 Continued

CFB as a Stream Cipher

Figure 8.5 *Cipher feedback (CFB) mode as a stream cipher*



8.1.3 Continued

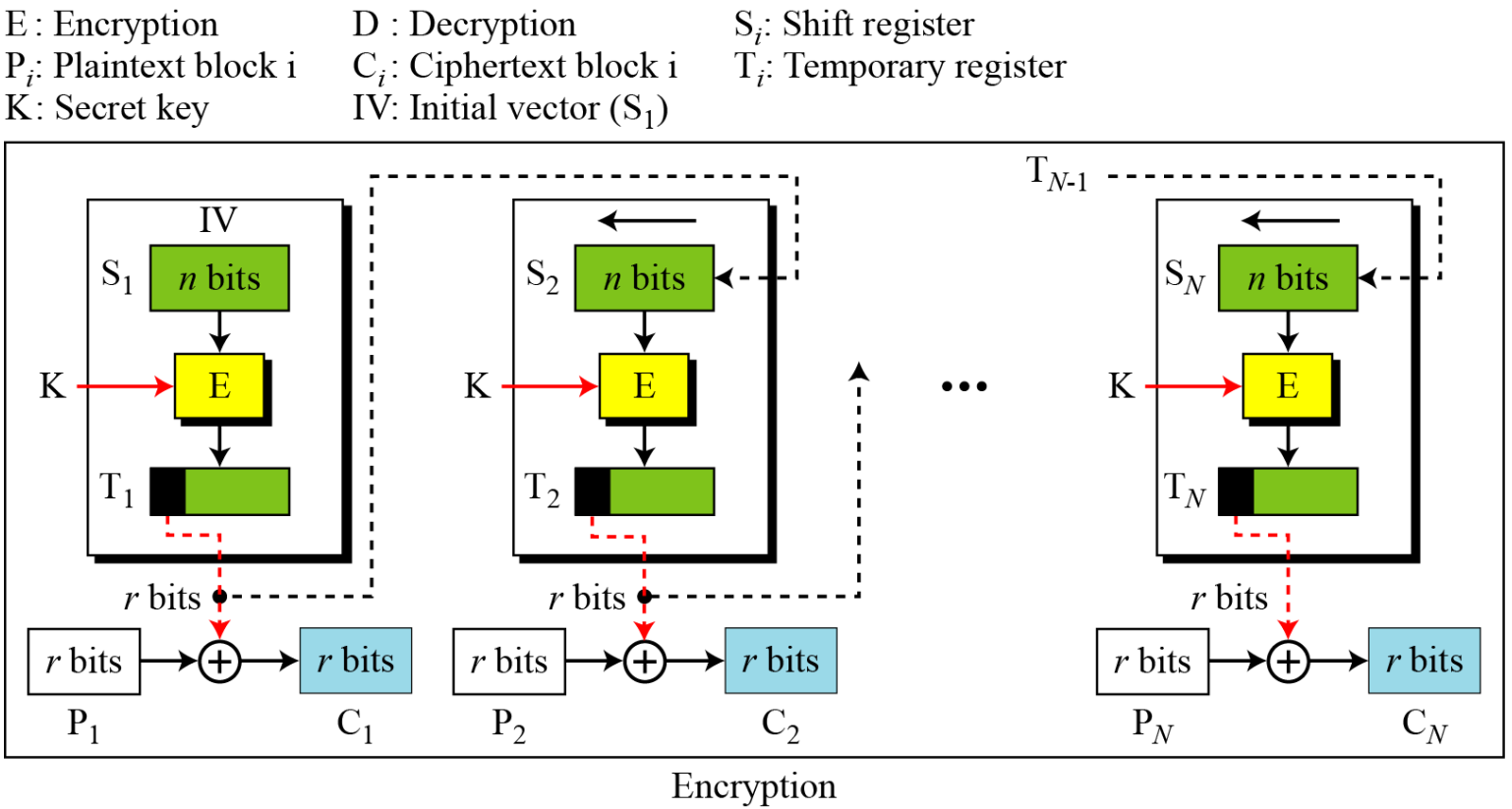
Algorithm 8.3 *Encryption algorithm for CFB*

```
CFB_Encryption (IV, K, r)
{
    i ← 1
    while (more blocks to encrypt)
    {
    input (P_i)
    if (i = 1)
        S ← IV
    else
        {
        Temp ← shiftLeft_r (S)
        S ← concatenate (Temp, C_{i−1})
        }
    T ← E_K(S)
    k_i ← selectLeft_r (T)
    C_i ← P_i ⊕ k_i
    output (C_i)
    i ← i + 1
    }
}
```

18.1.4 Output Feedback (OFB) Mode

In this mode each bit in the ciphertext is independent of the previous bit or bits. This avoids error propagation.

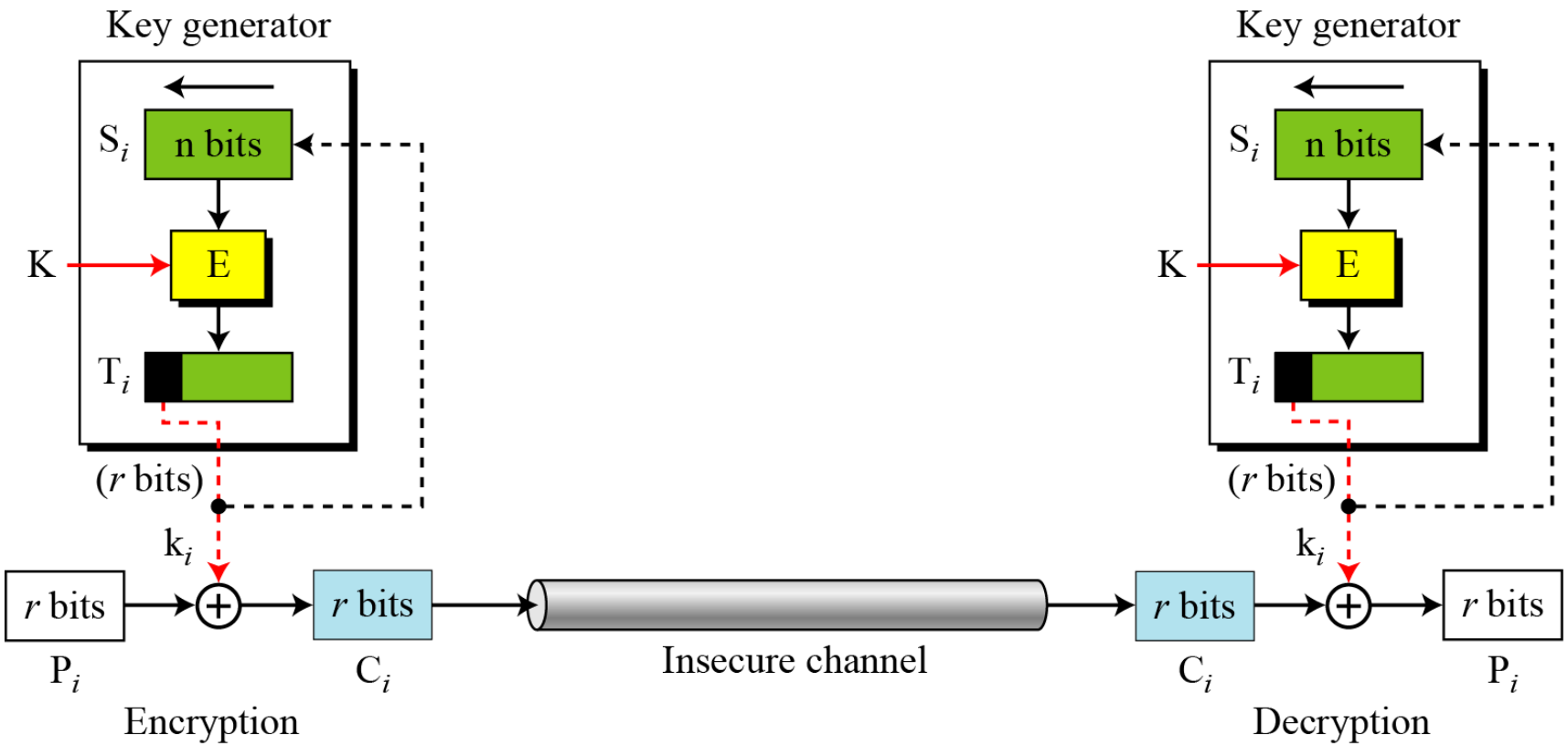
Figure 8.6 *Encryption in output feedback (OFB) mode*



8.1.4 Continued

OFB as a Stream Cipher

Figure 8.7 *Output feedback (OFB) mode as a stream cipher*



8.1.4 Continued

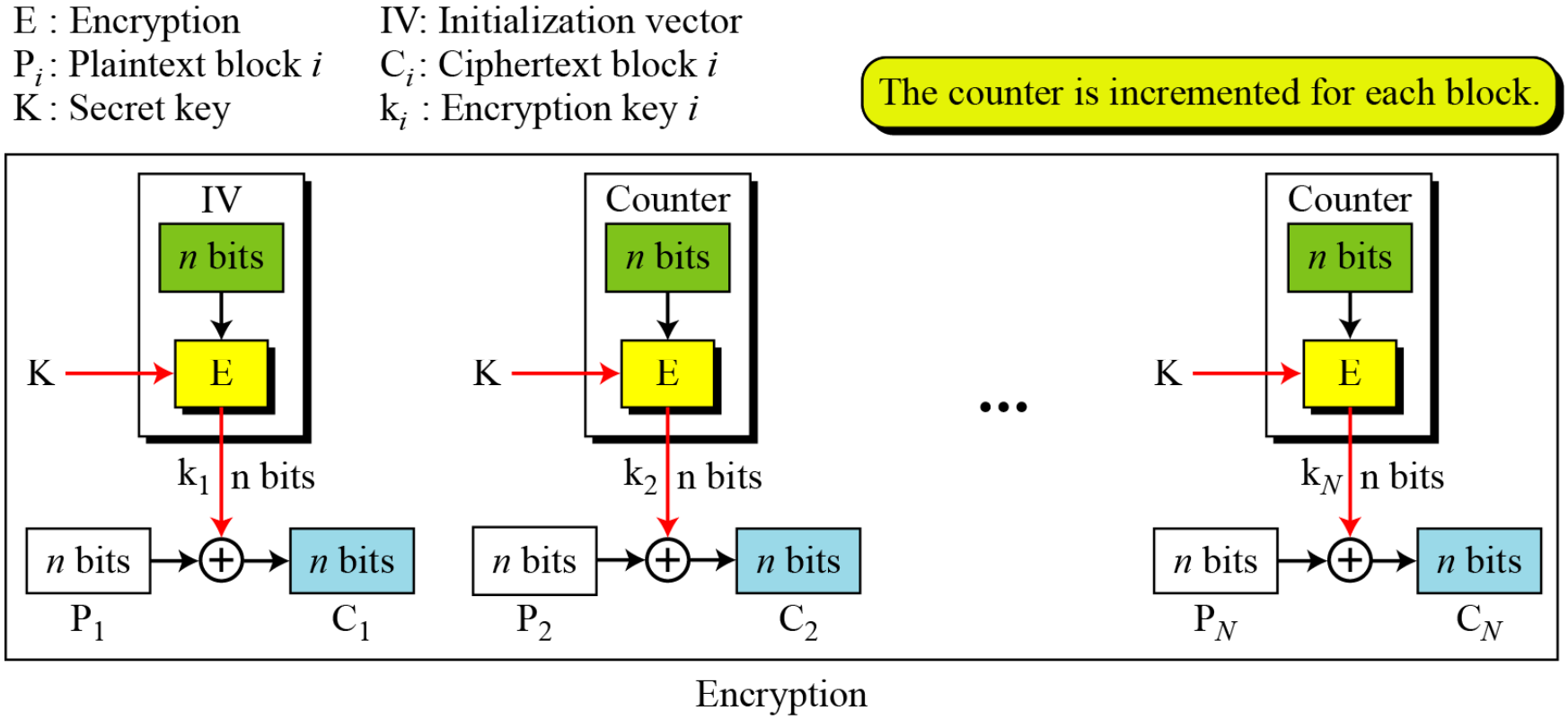
Algorithm 8.4 *Encryption algorithm for OFB*

```
OFB_Encryption (IV, K, r)
{
    i ← 1
    while (more blocks to encrypt)
    {
        input (P_i)
        if (i = 1)    S ← IV
        else
        {
            Temp ← shiftLeft_r (S)
            S ← concatenate (Temp, k_{i−1})
        }
        T ← E_K (S)
        k_i ← selectLeft_r (T)
        C_i ← P_i ⊕ k_i
        output (C_i)
        i ← i + 1
    }
}
```

8.1.5 Counter (CTR) Mode

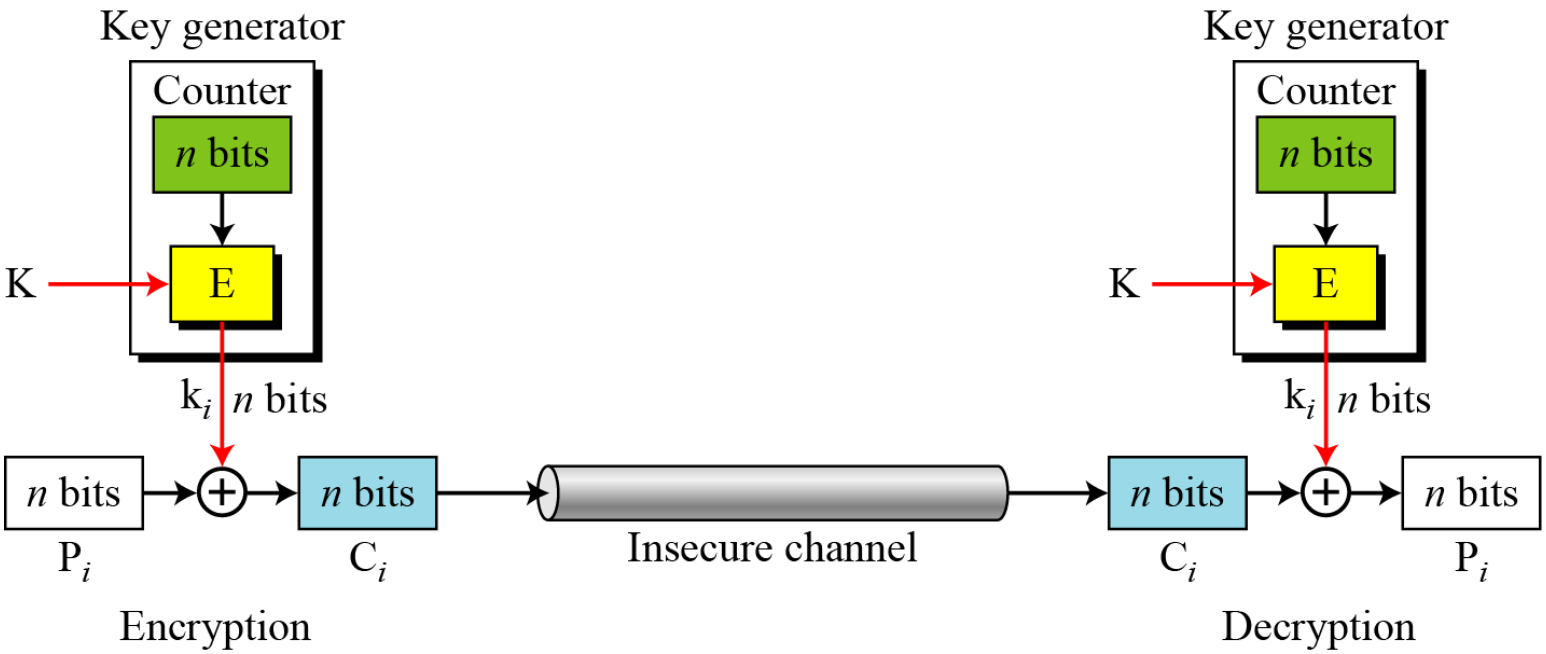
In the counter (CTR) mode, there is no feedback. The pseudorandomness in the key stream is achieved using a counter.

Figure 8.8 *Encryption in counter (CTR) mode*



8.1.5 Continued

Figure 8.9 *Counter (CTR) mode as a stream cipher*



8.1.5 Continued

Algorithm 8.5 *Encryption algorithm for CTR*

```
CTR_Encryption (IV, K, Plaintext blocks)
{
    Counter ← IV
    for (i = 1 to N)
    {
        Counter ← (Counter + i − 1) mod 2^N
        k_i ← E_K (Counter)
        C_i ← P_i ⊕ k_i
    }
    return Ciphertext blocks
}
```

8.1.5 Continued

Comparison of Different Modes

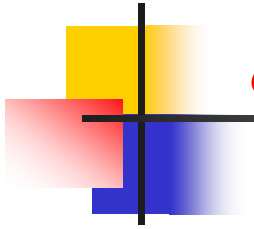
Table 8.1 *Summary of operation modes*

Operation Mode	*Description*	*Type of Result*	*Data Unit Size*
ECB	Each n-bit block is encrypted independently with the same cipher key.	Block cipher	n
CBC	Same as ECB, but each block is first exclusive-ored with the previous ciphertext.	Block cipher	n
CFB	Each r-bit block is exclusive-ored with an r-bit key, which is part of previous cipher text	Stream cipher	$r \leq n$
OFB	Same as CFB, but the shift register is updated by the previous r-bit key.	Stream cipher	$r \leq n$
CTR	Same as OFB, but a counter is used instead of a shift register.	Stream cipher	n

8-2 USE OF STREAM CIPHERS

Although the five modes of operations enable the use of block ciphers for encipherment of messages or files in large units and small units, sometimes pure stream are needed for enciphering small units of data such as characters or bits.

Topics discussed in this section:



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.

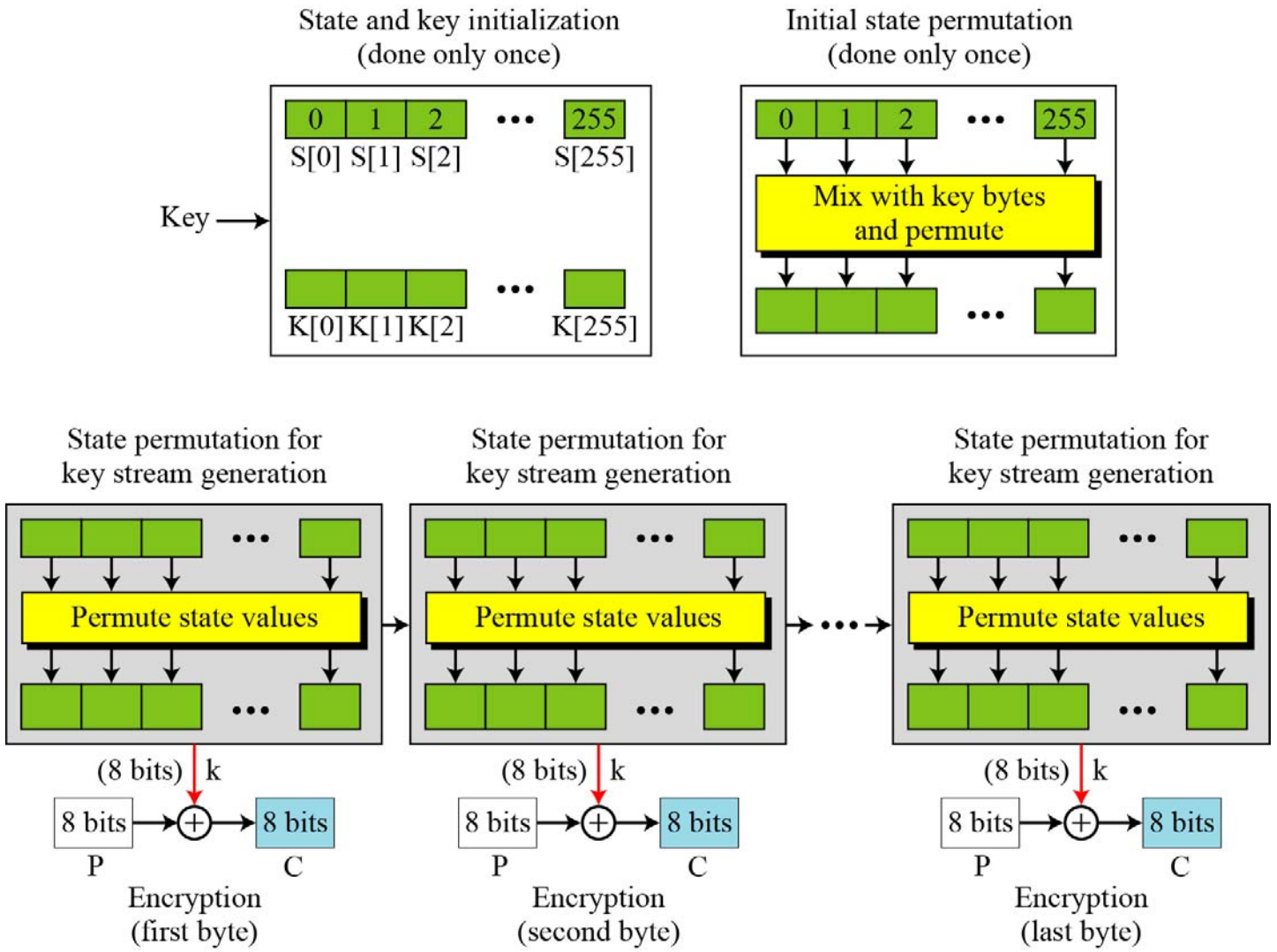
State

RC4 is based on the concept of a state.

S[0]	S[1]	S[2]	···	S[255]

8.2.1 Continued

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

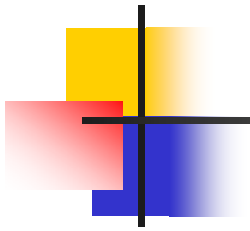
Algorithm 8.6 *Encryption algorithm for RC4*

```
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
    }
}
```



8.2.1 Continued

Example 8.5

To show the randomness of the stream key, we use a secret key with all bytes set to 0. The key stream for 20 values of *k* is (222, 24, 137, 65, 163, 55, 93, 58, 138, 6, 30, 103, 87, 110, 146, 109, 199, 26, 127, 163).

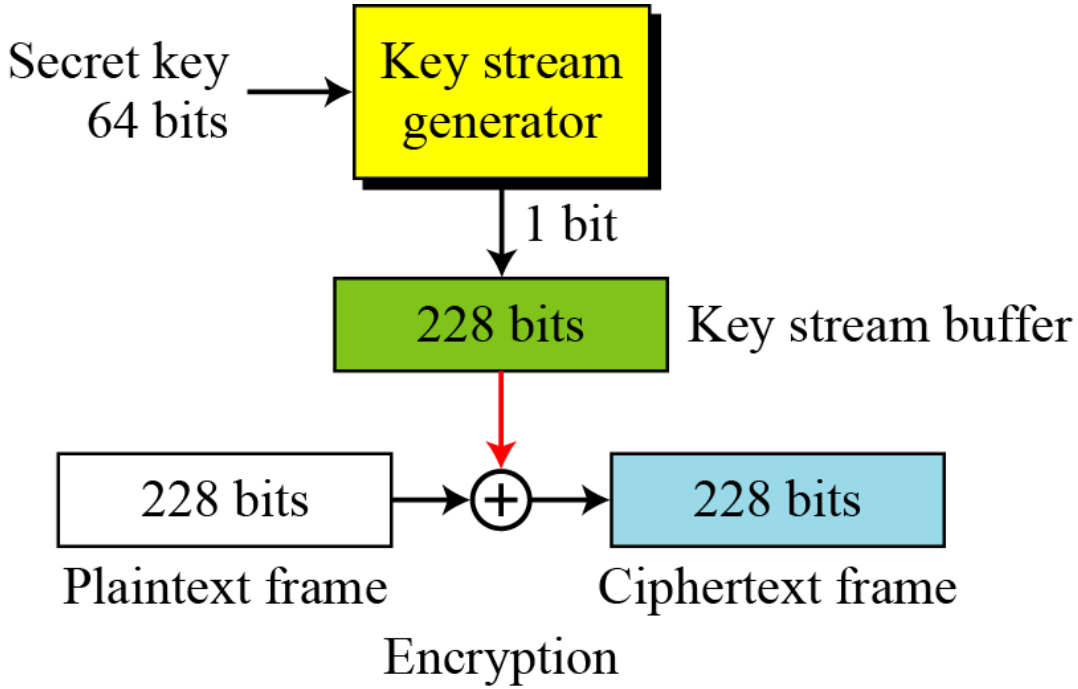
Example 8.6

Repeat Example 8.5, but let the secret key be five bytes of (15, 202, 33, 6, 8). The key stream is (248, 184, 102, 54, 212, 237, 186, 133, 51, 238, 108, 106, 103, 214, 39, 242, 30, 34, 144, 49). Again the randomness in the key stream is obvious.

8.2.2 A5/1

A5/1 (a member of the A5 family of ciphers) is used in the Global System for Mobile Communication (GSM), a network for mobile telephone communication..

Figure 8.11 *General outline of A5/1*

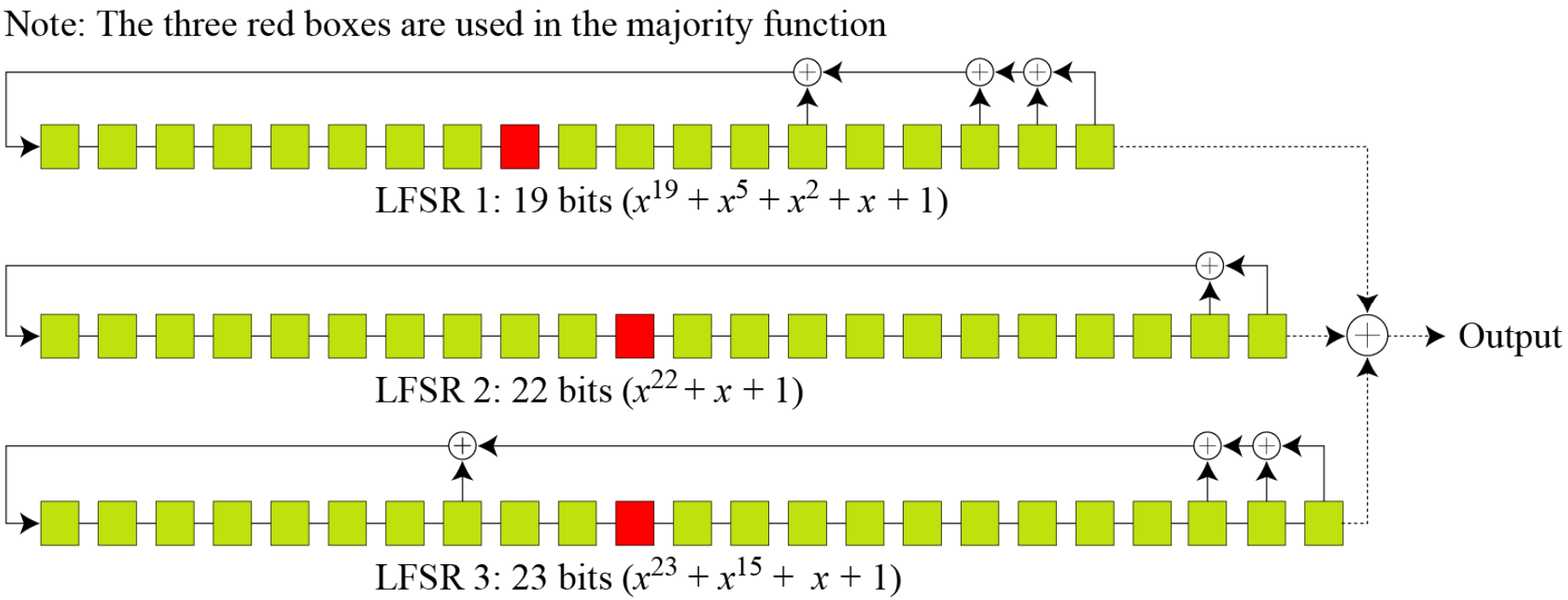


8.2.2 Continued

Key Generator

A5/1 uses three LFSRs with 19, 22, and 23 bits.

Figure 8.12 *Three LFSR's in A5/1*



8.2.2 Continued

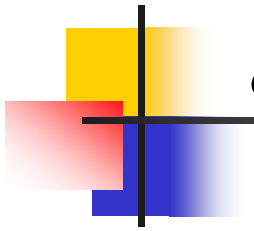
Initialization

1. *set all bits in three LFSRs to 0.*
2.

```
for (i = 0 to 63)
   {
      Exclusive-or K[i] with the leftmost bit in all three registers.
      Clock all three LFSRs
   }
```

3.

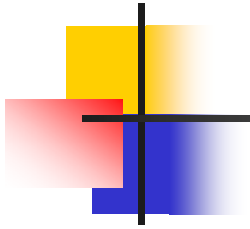
```
for (i = 0 to 21)
   {
      Exclusive-or FrameNumber [i] with the leftmost bit in all three registers.
      Clock all three LFSRs
   }
```



8.2.2 Continued

4.

```
for (i = 0 to 99)
  {
    Clock the whole generator based on the majority function.
  }
```



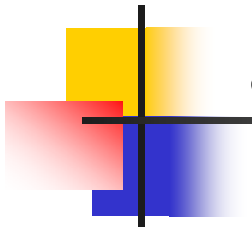
8.2.2 Continued

Example 8.7

At a point of time the clocking bits are 1, 0, and 1. Which LFSR is clocked (shifted)?

Solution

The result of Majority (1, 0, 1) = 1. LFSR1 and LAFS3 are shifted, but LFSR2 is not.



8.2.2 Continued

Encryption/Decryption

The bit streams created from the key generator are buffered to form a 228-bit key that is exclusive-ored with the plaintext frame to create the ciphertext frame. Encryption/decryption is done one frame at a time.

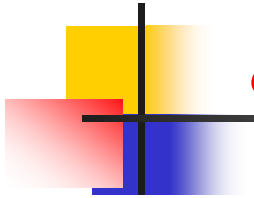
8-3 OTHER ISSUES

Encipherment using symmetric-key block or stream ciphers requires discussion of other issues.

Topics discussed in this section:

8.3.1 Key Management

8.3.2 Key Generation



8.3.1 Key Management

Alice and Bob need to share a secret key between themselves to securely communicate using a symmetric-key cipher. If there are n entities in the community, n(n − 1)/2 keys are needed.

Note

Key management is discussed in Chapter 15.

8.3.2 Key Generation

Different symmetric-key ciphers need keys of different sizes. The selection of the key must be based on a systematic approach to avoid a security leak. The keys need to be chosen randomly. This implies that there is a need for random (or pseudorandom) number generator.

Note

Random number generators are discussed in Appendix K.