CHAPTER 6

BLOCK CIPHER OPERATION

Many savages at the present day regard their names as vital parts of themselves, and therefore take great pains to conceal their real names, lest these should give to evil-disposed persons a handle by which to injure their owners.

—*The Golden Bough*, Sir James George Frazer

KEY POINTS

- 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.
- Triple DES makes use of three stages of the DES algorithm, using a total of two or three distinct keys.
- A mode of operation is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream.
- Five modes of operation have been standardized by NIST for use with symmetric block ciphers such as DES and AES: electronic codebook mode, cipher block chaining mode, cipher feedback mode, output feedback mode, and counter mode.
- Another important mode, XTS-AES, has been standardized by the IEEE Security in Storage Working Group (P1619). The standard describes a method of encryption for data stored in sector-based devices where the threat model includes possible access to stored data by the adversary.

This chapter continues our discussion of symmetric ciphers. We begin with the topic of multiple encryption, looking in particular at the most widely used multiple-encryption scheme: triple DES.

The chapter next turns to the subject of block cipher modes of operation. We find that there are a number of different ways to apply a block cipher to plaintext, each with its own advantages and particular applications.

6.1 MULTIPLE ENCRYPTION AND TRIPLE DES

Given the potential vulnerability of DES to a brute-force attack, there has been considerable interest in finding an alternative. One approach is to design a completely new algorithm, of which AES is a prime example. Another alternative, which would preserve the existing investment in software and equipment, is to use multiple encryption with DES and multiple keys. We begin by examining the

simplest example of this second alternative. We then look at the widely accepted triple DES (3DES) approach.

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.

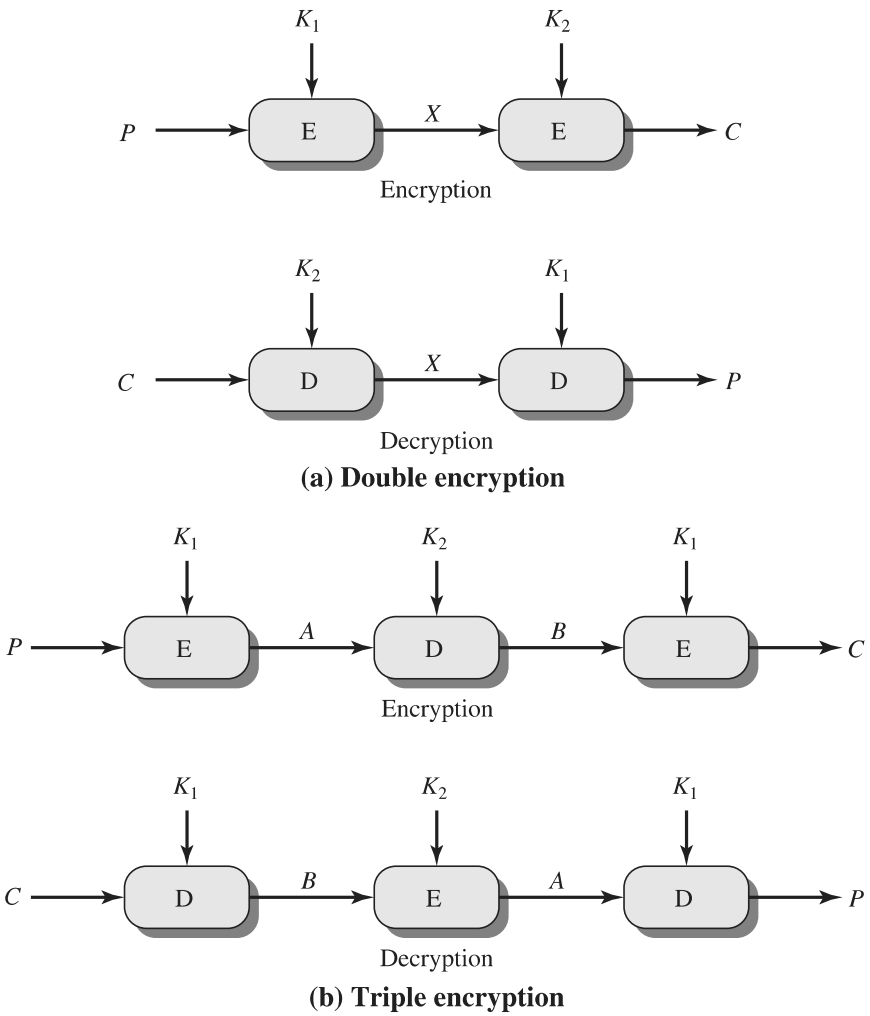


Figure 6.1 Multiple Encryption

Reduction to a Single Stage Suppose it were true for DES, for all 56-bit key values, that given any two keys K_1 and K_2, it would be possible to find a key K_3 such that

$$\mathrm{E}(K_2, \mathrm{E}(K_1, P)) = \mathrm{E}(K_3, P) \tag{6.1}$$

If this were the case, then double encryption, and indeed any number of stages of multiple encryption with DES, would be useless because the result would be equivalent to a single encryption with a single 56-bit key.

On the face of it, it does not appear that Equation (6.1) is likely to hold. Consider that encryption with DES is a mapping of 64-bit blocks to 64-bit blocks. In fact, the mapping can be viewed as a permutation. That is, if we consider all 2^{64} possible input blocks, DES encryption with a specific key will map each block into a unique 64-bit block. Otherwise, if, say, two given input blocks mapped to the same output block, then decryption to recover the original plaintext would be impossible. With 2^{64} possible inputs, how many different mappings are there that generate a permutation of the input blocks? The value is easily seen to be

$$(2^{64})! = 10^{34738000000000000000} > (10^{10^{20}})$$

On the other hand, DES defines one mapping for each different key, for a total number of mappings:

$$2^{56} < 10^{17}$$

Therefore, it is reasonable to assume that if DES is used twice with different keys, it will produce one of the many mappings that are not defined by a single application of DES. Although there was much supporting evidence for this assumption, it was not until 1992 that the assumption was proven [CAMP92].

Meet-in-the-Middle Attack Thus, the use of double DES results in a mapping that is not equivalent to a single DES encryption. But there is a way to attack this scheme, one that does not depend on any particular property of DES but that will work against any block encryption cipher.

The algorithm, known as a **meet-in-the-middle attack**, was first described in [DIFF77]. It is based on the observation that, if we have

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

then (see Figure 6.1a)

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

Given a known pair, (P, C), the attack proceeds as follows. First, encrypt P for all 2^{56} possible values of K_1. Store these results in a table and then sort the table by the values of X. Next, decrypt C using all 2^{56} possible values of K_2. As each decryption is produced, check the result against the table for a match. If a match occurs, then test the two resulting keys against a new known plaintext–ciphertext pair. If the two keys produce the correct ciphertext, accept them as the correct keys.

For any given plaintext P, there are 2^{64} possible ciphertext values that could be produced by double DES. Double DES uses, in effect, a 112-bit key, so that

there are 2^{112} possible keys. Therefore, on average, for a given plaintext P, the number of different 112-bit keys that will produce a given ciphertext C is $2^{112}/2^{64} = 2^{48}$. Thus, the foregoing procedure will produce about 2^{48} false alarms on the first (P, C) pair. A similar argument indicates that with an additional 64 bits of known plaintext and ciphertext, the false alarm rate is reduced to $2^{48-64} = 2^{-16}$. Put another way, if the meet-in-the-middle attack is performed on two blocks of known plaintext–ciphertext, the probability that the correct keys are determined is $1 - 2^{-16}$. The result is that a known plaintext attack will succeed against double DES, which has a key size of 112 bits, with an effort on the order of 2^{56}, which is not much more than the 2^{55} required for single DES.

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):

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

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

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:

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

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

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]

Currently, there are no practical cryptanalytic attacks on 3DES. Coppersmith [COPP94] notes that the cost of a brute-force key search on 3DES is on the order of $2^{112} \approx (5 \times 10^{33})$ and estimates that the cost of differential cryptanalysis suffers an exponential growth, compared to single DES, exceeding 10^{52}.

It is worth looking at several proposed attacks on 3DES that, although not practical, give a flavor for the types of attacks that have been considered and that could form the basis for more successful future attacks.

The first serious proposal came from Merkle and Hellman [MERK81]. Their plan involves finding plaintext values that produce a first intermediate value of $A = 0$

[1]American National Standard (ANS): *Financial Institution Key Management (Wholesale)*. From its title, X9.17 appears to be a somewhat obscure standard. Yet a number of techniques specified in this standard have been adopted for use in other standards and applications, as we shall see throughout this book.

(Figure 6.1b) and then using the meet-in-the-middle attack to determine the two keys. The level of effort is 2^{56}, but the technique requires 2^{56} chosen plaintext–ciphertext pairs, which is a number unlikely to be provided by the holder of the keys.

A known-plaintext attack is outlined in [VANO90]. This method is an improvement over the chosen-plaintext approach but requires more effort. The attack is based on the observation that if we know A and C (Figure 6.1b), then the problem reduces to that of an attack on double DES. Of course, the attacker does not know A, even if P and C are known, as long as the two keys are unknown. However, the attacker can choose a potential value of A and then try to find a known (P, C) pair that produces A. The attack proceeds as follows.

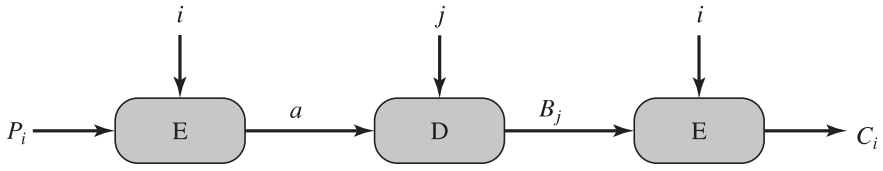
1. Obtain n (P, C) pairs. This is the known plaintext. Place these in a table (Table 1) sorted on the values of P (Figure 6.2b).
2. Pick an arbitrary value a for A, and create a second table (Figure 6.2c) with entries defined in the following fashion. For each of the 2^{56} possible keys $K_1 = i$, calculate the plaintext value P_i that produces a:

$$P_i = \mathrm{D}(i, a)$$

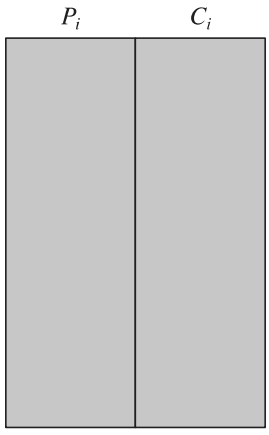
For each P_i that matches an entry in Table 1, create an entry in Table 2 consisting of the K_1 value and the value of B that is produced for the (P, C) pair from Table 1, assuming that value of K_1:

$$B = \mathrm{D}(i, C)$$

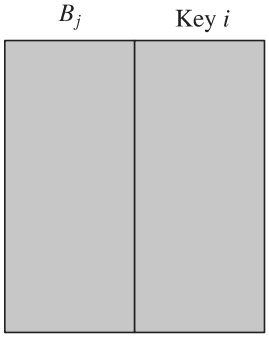
At the end of this step, sort Table 2 on the values of B.



(a) Two-key triple encryption with candidate pair of keys



(b) Table of *n* known plaintext-ciphertext pairs, sorted on *P*



(c) Table of intermediate values and candidate keys

Figure 6.2 Known-Plaintext Attack on Triple DES

3. We now have a number of candidate values of K_1 in Table 2 and are in a position to search for a value of K_2. For each of the 2^{56} possible keys $K_2 = j$, calculate the second intermediate value for our chosen value of a:

$$B_j = \mathrm{D}(j, a)$$

At each step, look up B_j in Table 2. If there is a match, then the corresponding key i from Table 2 plus this value of j are candidate values for the unknown keys (K_1, K_2). Why? Because we have found a pair of keys (i, j) that produce a known (P, C) pair (Figure 6.2a).

4. Test each candidate pair of keys (i, j) on a few other plaintext–ciphertext pairs. If a pair of keys produces the desired ciphertext, the task is complete. If no pair succeeds, repeat from step 1 with a new value of a.

For a given known (P, C), the probability of selecting the unique value of a that leads to success is $1/2^{64}$. Thus, given n (P, C) pairs, the probability of success for a single selected value of a is $n/2^{64}$. A basic result from probability theory is that the expected number of draws required to draw one red ball out of a bin containing n red balls and $N - n$ green balls is $(N + 1)/(n + 1)$ if the balls are not replaced. So the expected number of values of a that must be tried is, for large n,

$$\frac{2^{64} + 1}{n + 1} \approx \frac{2^{64}}{n}$$

Thus, the expected running time of the attack is on the order of

$$\left(2^{56}\right)\frac{2^{64}}{n} = 2^{120 - \log_2 n}$$

Triple DES with Three Keys

Although the attacks just described appear impractical, anyone using two-key 3DES may feel some concern. Thus, many researchers now feel that three-key 3DES is the preferred alternative (e.g., [KALI96a]). Three-key 3DES has an effective key length of 168 bits and is defined as

$$C = \mathrm{E}(K_3, \mathrm{D}(K_2, \mathrm{E}(K_1, P)))$$

Backward compatibility with DES is provided by putting $K_3 = K_2$ or $K_1 = K_2$.

A number of Internet-based applications have adopted three-key 3DES, including PGP and S/MIME, both discussed in Chapter 18.

6.2 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

up into b-bit blocks. When multiple blocks of plaintext are encrypted using the same key, a number of security issues arise. To apply a block cipher in a variety of applications, five *modes of operation* have been defined by NIST (SP 800-38A). In essence, a mode of operation is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream. The five modes are intended to cover a wide variety of applications of encryption for which a block cipher could be used. These modes are intended for use with any symmetric block cipher, including triple DES and AES. The modes are summarized in Table 6.1 and described in this and the following sections.

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

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

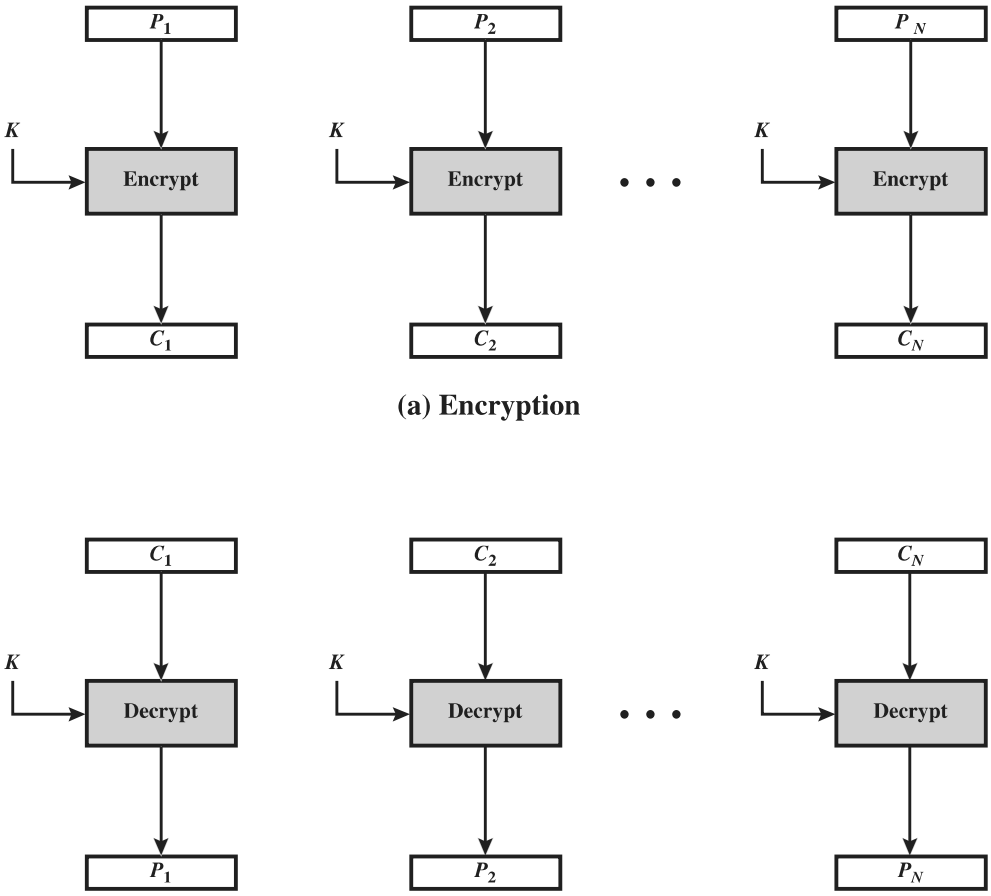


Figure 6.3 Electronic Codebook (ECB) Mode

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

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

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.

The most significant characteristic of ECB is that if the same b-bit block of plaintext appears more than once in the message, it always produces the same ciphertext.

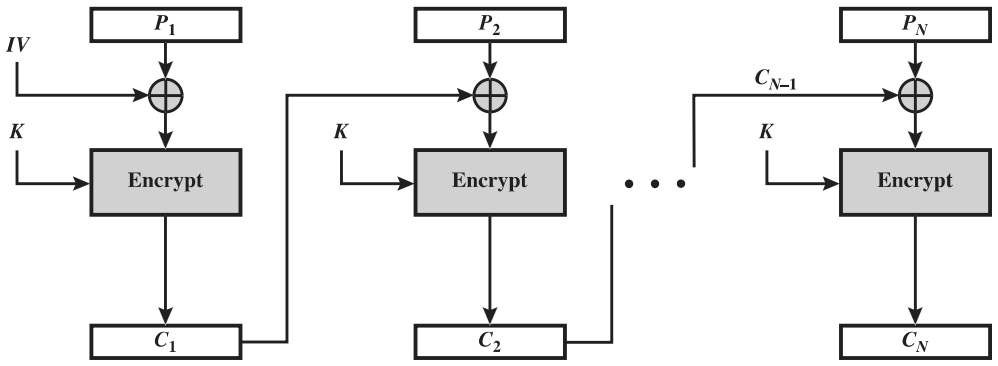
For lengthy messages, the ECB mode may not be secure. If the message is highly structured, it may be possible for a cryptanalyst to exploit these regularities. For example, if it is known that the message always starts out with certain predefined fields, then the cryptanalyst may have a number of known plaintext–ciphertext pairs to work with. If the message has repetitive elements with a period of repetition a multiple of b bits, then these elements can be identified by the analyst. This may help in the analysis or may provide an opportunity for substituting or rearranging blocks.

6.3 CIPHER BLOCK CHAINING MODE

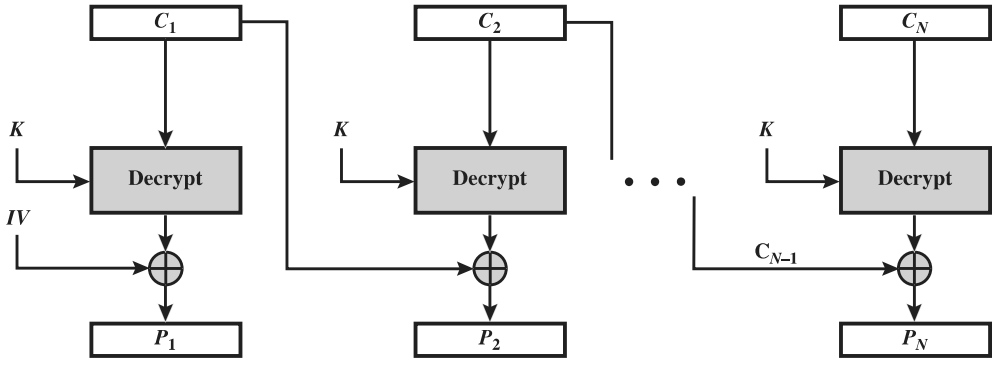
To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks. A simple way to satisfy this requirement is the **cipher block chaining** (**CBC**) mode (Figure 6.4). In this scheme, the input to the encryption 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

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

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

The IV must be known to both the sender and receiver but be unpredictable by a third party. In particular, for any given plaintext, it must not be possible to predict the IV that will be associated to the plaintext in advance of the generation of the IV. For maximum security, the IV should be protected against unauthorized changes. This could be done by sending the IV using ECB encryption. One reason for protecting the IV is as follows: If an opponent is able to fool the receiver into using a different value for IV, then the opponent is able to invert selected bits in the first block of plaintext. To see this, consider

$$C_1 = \mathrm{E}(K, [\mathrm{IV} \oplus P_1])$$
$$P_1 = \mathrm{IV} \oplus \mathrm{D}(K, C_1)$$

Now use the notation that $X[i]$ denotes the ith bit of the b-bit quantity X. Then

$$P_1[i] = \mathrm{IV}[i] \oplus \mathrm{D}(K, C_1)[i]$$

Then, using the properties of XOR, we can state

$$P_1[i]' = \mathrm{IV}[i]' \oplus \mathrm{D}(K, C_1)[i]$$

where the prime notation denotes bit complementation. This means that if an opponent can predictably change bits in IV, the corresponding bits of the received value of P_1 can be changed.

For other possible attacks based on prior knowledge of IV, see [VOYD83].

So long as it is unpredictable, the specific choice of IV is unimportant. Sp800-38a recommends two possible methods: The first method is to apply the encryption function, under the same key that is used for the encryption of the plaintext, to a **nonce**.[2] The nonce must be a data block that is unique to each execution of the encryption operation. For example, the nonce may be a counter, a timestamp, or

[2]NIST SP-800-90 (*Recommendation for Random Number Generation Using Deterministic Random Bit Generators*) defines nonce as follows: A time-varying value that has at most a negligible chance of repeating, e.g., a random value that is generated anew for each use, a timestamp, a sequence number, or some combination of these.

a message number. The second method is to generate a random data block using a random number generator.

In conclusion, because of the chaining mechanism of CBC, it is an appropriate mode for encrypting messages of length greater than b bits.

In addition to its use to achieve confidentiality, the CBC mode can be used for authentication. This use is described in Chapter 12.

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.

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.

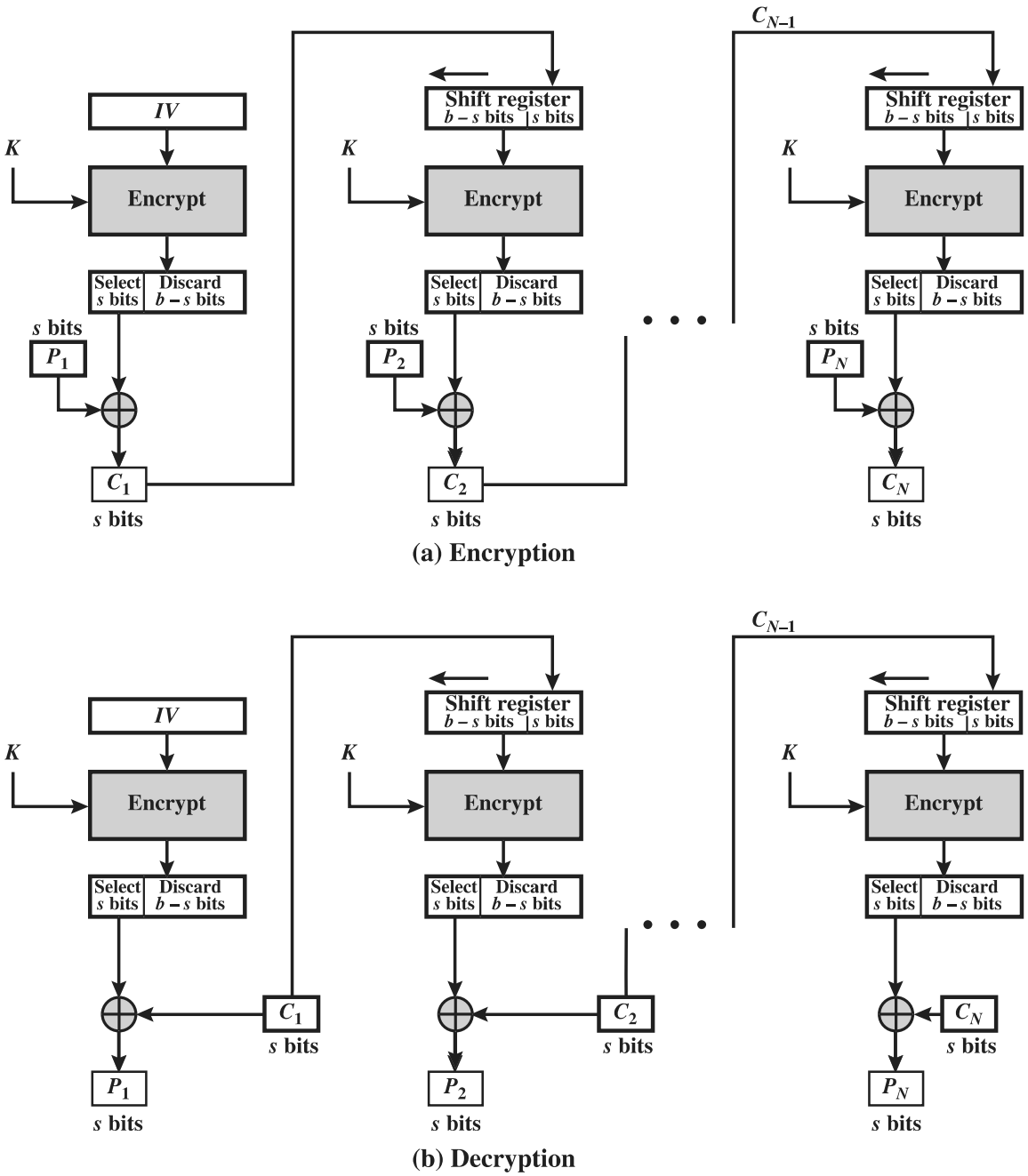


Figure 6.5 *s*-bit Cipher Feedback (CFB) Mode

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

Although CFB can be viewed as a stream cipher, it does not conform to the typical construction of a stream cipher. In a typical stream cipher, the cipher takes as input some initial value and a key and generates a stream of bits, which is then XORed with the plaintext bits (see Figure 3.1). In the case of CFB, the stream of bits that is XORed with the plaintext also depends on the plaintext.

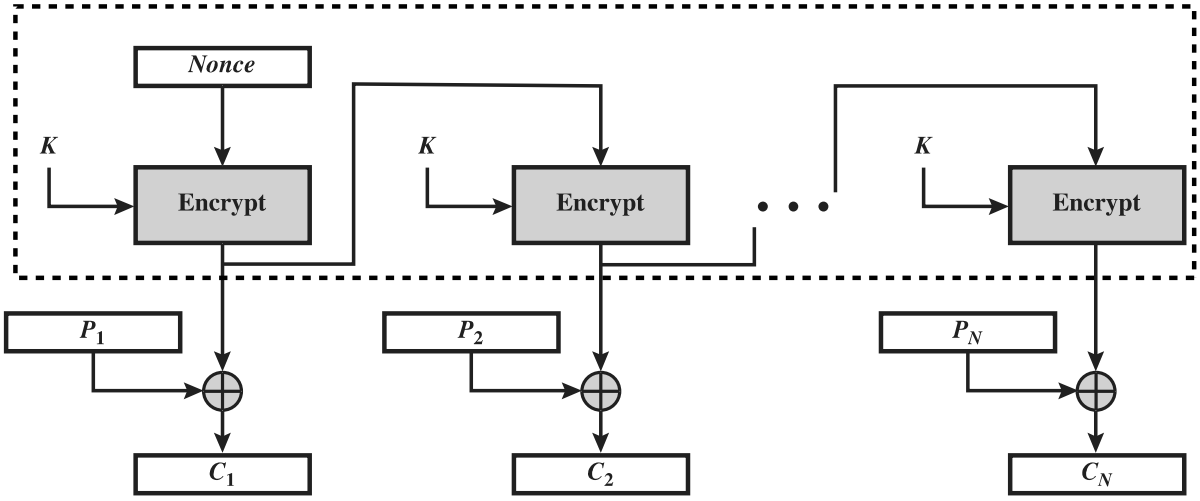
6.5 OUTPUT FEEDBACK MODE

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

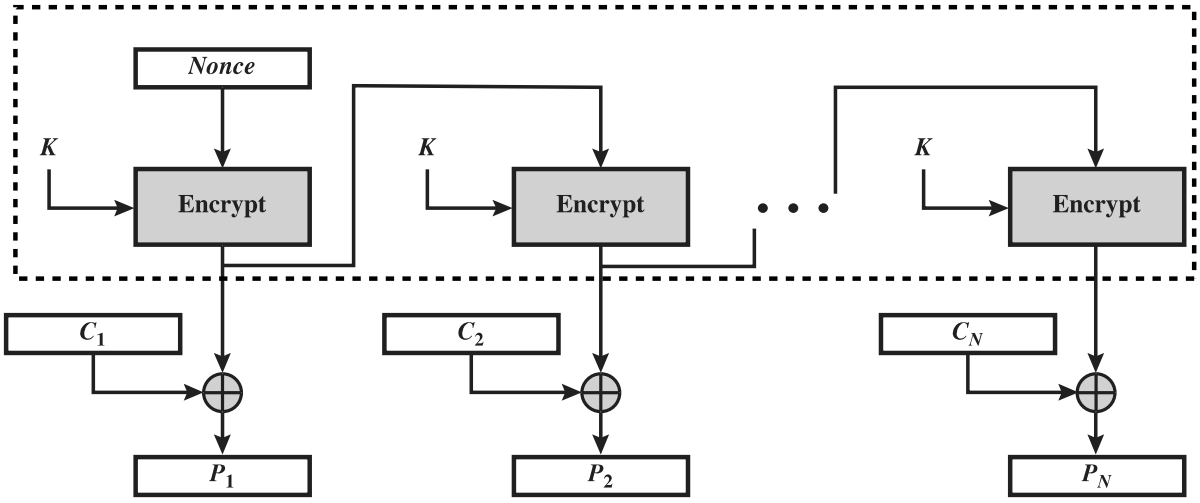
$$C_j = P_j \oplus \mathrm{E}(K, [C_{j-i} \oplus P_{j-1}])$$

By rearranging terms, we can demonstrate that decryption works.

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



(a) Encryption



(b) Decryption

Figure 6.6 Output Feedback (OFB) Mode

We can define OFB mode as follows.

OFB	$I_1 = Nonce$ $I_j = O_{j-1}$ $j = 2, \ldots, N$ $O_j = \mathrm{E}(K, I_j)$ $j = 1, \ldots, N$ $C_j = P_j \oplus O_j$ $j = 1, \ldots, N-1$ $C_N^* = P_N^* \oplus \mathrm{MSB}_u(O_N)$	$I_1 = Nonce$ $I_j = \mathrm{LSB}_{b-s}(I_{j-1}) \parallel C_{j-1}$ $j = 2, \ldots, N$ $O_j = \mathrm{E}(K, I_j)$ $j = 1, \ldots, N$ $P_j = C_j \oplus O_j$ $j = 1, \ldots, N-1$ $P_N^* = C_N^* \oplus \mathrm{MSB}_u(O_N)$

Let the size of a block be b. If the last block of plaintext contains u bits (indicated by *), with $u < b$, the most significant u bits of the last output block O_N are used for the XOR operation; the remaining $b-u$ bits of the last output block are discarded.

As with CBC and CFB, the OFB mode requires an initialization vector. In the case of OFB, the IV must be a nonce; that is, the IV must be unique to each execution of the encryption operation. The reason for this is that the sequence of encryption output blocks, O_i, depends only on the key and the IV and does not depend on the plaintext. Therefore, for a given key and IV, the stream of output bits used to XOR with the stream of plaintext bits is fixed. If two different messages had an identical block of plaintext in the identical position, then an attacker would be able to determine that portion of the O_i stream.

One advantage of the OFB method is that bit errors in transmission do not propagate. For example, if a bit error occurs in C_1, only the recovered value of P_1 is affected; subsequent plaintext units are not corrupted. With CFB, C_1 also serves as input to the shift register and therefore causes additional corruption downstream.

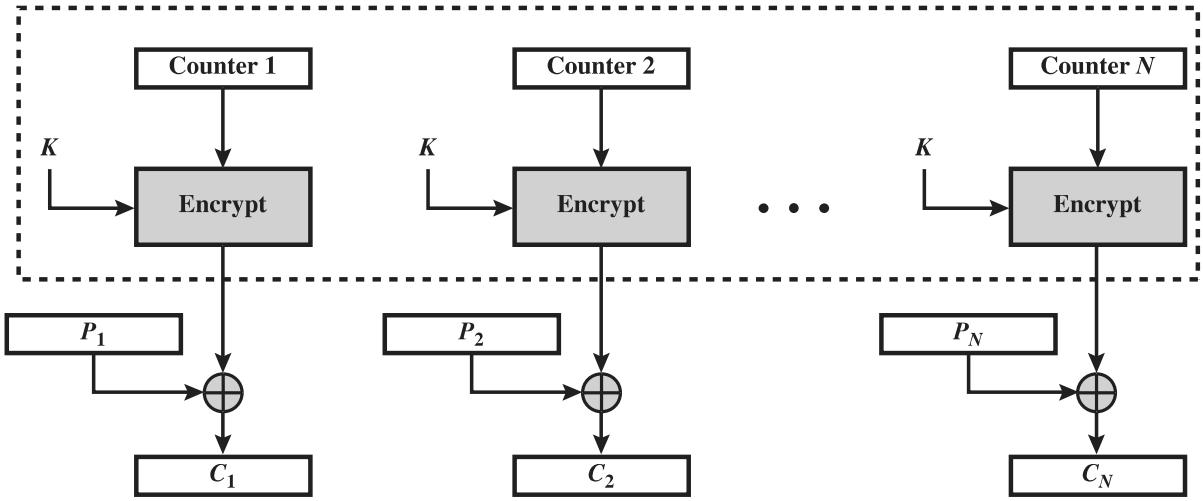
The disadvantage of OFB is that it is more vulnerable to a message stream modification attack than is CFB. Consider that complementing a bit in the ciphertext complements the corresponding bit in the recovered plaintext. Thus, controlled changes to the recovered plaintext can be made. This may make it possible for an opponent, by making the necessary changes to the checksum portion of the message as well as to the data portion, to alter the ciphertext in such a way that it is not detected by an error-correcting code. For a further discussion, see [VOYD83].

OFB has the structure of a typical stream cipher, because the cipher generates a stream of bits as a function of an initial value and a key, and that stream of bits is XORed with the plaintext bits (see Figure 3.1). The generated stream that is XORed with the plaintext is itself independent of the plaintext; this is highlighted by dashed boxes in Figure 6.6. One distinction from the stream ciphers we discuss in Chapter 7 is that OFB encrypts plaintext a full block at a time, where typically a block is 64 or 128 bits. Many stream ciphers encrypt one byte at a time.

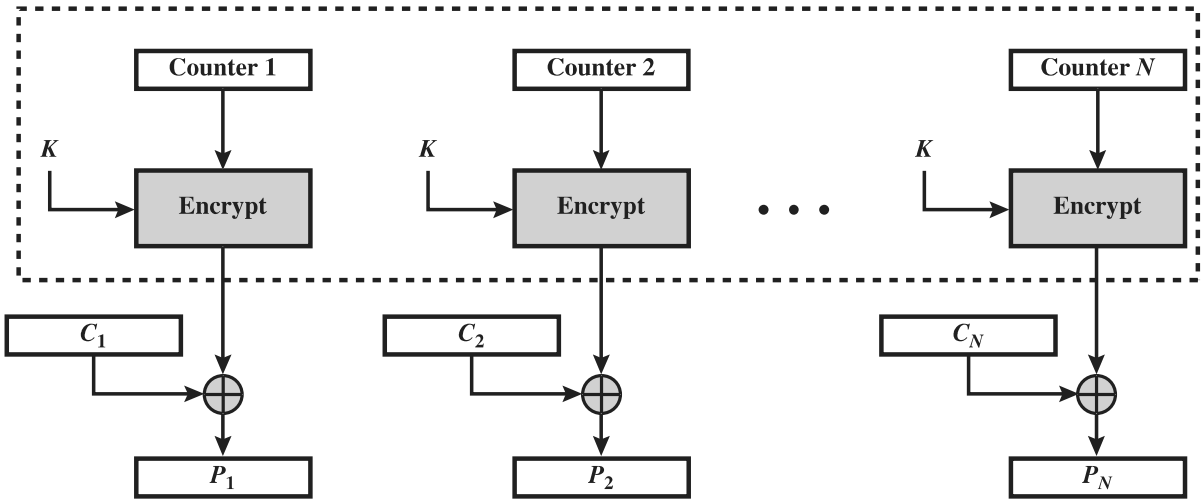
6.6 COUNTER MODE

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



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

For the last plaintext block, which may be a partial block of u bits, the most significant u bits of the last output block are used for the XOR operation; the remaining $b-u$ bits are discarded. Unlike the ECB, CBC, and CFB modes, we do not need to use padding because of the structure of the CTR mode.

As with the OFB mode, the initial counter value must be a nonce; that is, T_1 must be different for all of the messages encrypted using the same key. Further, all T_i values across all messages must be unique. If, contrary to this requirement, a counter value is used multiple times, then the confidentiality of all of the plaintext blocks corresponding to that counter value may be compromised. In particular, if any plaintext block that is encrypted using a given counter value is known, then the output of the encryption function can be determined easily from the associated ciphertext block. This output allows any other plaintext blocks that are encrypted using the same counter value to be easily recovered from their associated ciphertext blocks.

One way to ensure the uniqueness of counter values is to continue to increment the counter value by 1 across messages. That is, the first counter value of the each message is one more than the last counter value of the preceding message.

[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 ith 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.

Note that, with the exception of ECB, all of the NIST-approved block cipher modes of operation involve feedback. This is clearly seen in Figure 6.8. To highlight the feedback mechanism, it is useful to think of the encryption function as taking input from a input register whose length equals the encryption block length and with output stored in an output register. The input register is updated one block at a time by the feedback mechanism. After each update, the encryption algorithm is executed, producing a result in the output register. Meanwhile, a block of plaintext is accessed. Note that both OFB and CTR produce output that is independent of both the plaintext and the ciphertext. Thus, they are natural candidates for stream ciphers that encrypt plaintext by XOR one full block at a time.

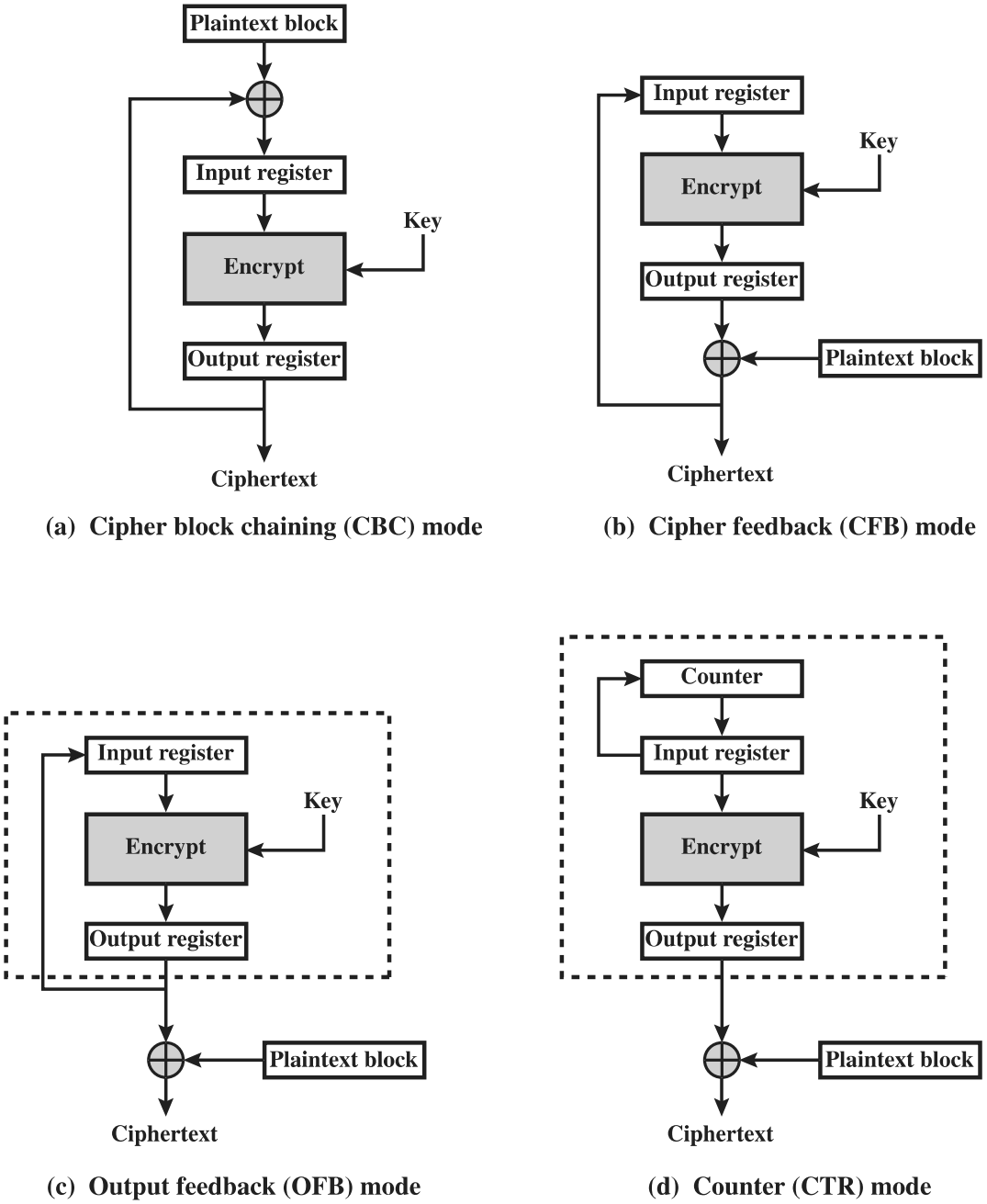


Figure 6.8 Feedback Characteristic of Modes of Operation

6.7 XTS-AES MODE FOR BLOCK-ORIENTED STORAGE DEVICES

NIST is currently in the process of approving an additional block cipher mode of operation, XTS-AES. This mode is also an IEEE standard, IEEE Std 1619-2007, which was developed by the IEEE Security in Storage Working Group (P1619). The standard describes a method of encryption for data stored in sector-based devices where the threat model includes possible access to stored data by the adversary.

The XTS-AES mode is based on the concept of a tweakable block cipher, introduced in [LISK02]. The form of this concept used in XTS-AES was first described in [ROGA04]. The standard has received widespread industry support.

Storage Encryption Requirements

The requirements for encrypting stored data, also referred to as "data at rest" differ somewhat from those for transmitted data. The P1619 standard was designed to have the following characteristics:

1. The ciphertext is freely available for an attacker. Among the circumstances that lead to this situation:
   a. A group of users has authorized access to a database. Some of the records in the database are encrypted so that only specific users can successfully read/write them. Other users can retrieve an encrypted record but are unable to read it without the key.
   b. An unauthorized user manages to gain access to encrypted records.
   c. A data disk or laptop is stolen, giving the adversary access to the encrypted data.
2. The data layout is not changed on the storage medium and in transit. The encrypted data must be the same size as the plaintext data.
3. Data are accessed in fixed sized blocks, independently from each other. That is, an authorized user may access one or more blocks in any order.
4. Encryption is performed in 16-byte blocks, independently from other blocks (except the last two plaintext blocks of a sector, if its size is not a multiple of 16 bytes).
5. There are no other metadata used, except the location of the data blocks within the whole data set.
6. The same plaintext is encrypted to different ciphertexts at different locations, but always to the same ciphertext when written to the same location again.
7. A standard conformant device can be constructed for decryption of data encrypted by another standard conformant device.

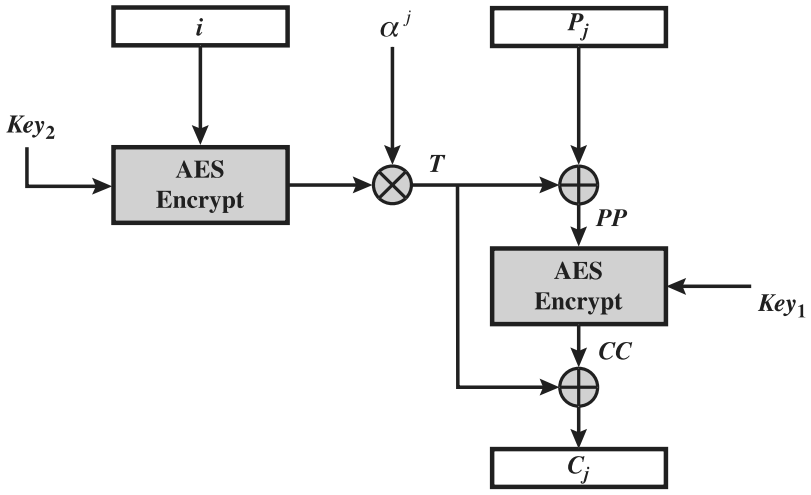
The P1619 group considered some of the existing modes of operation for use with stored data. For CTR mode, an adversary with write access to the encrypted media can flip any bit of the plaintext simply by flipping the corresponding ciphertext bit.

Next, consider requirement 6 and the use of CBC. To enforce the requirement that the same plaintext encrypt to different ciphertext in different locations, the IV could be derived from the sector number. Each sector contains multiple blocks. An adversary with read/write access to the encrypted disk can copy a ciphertext sector

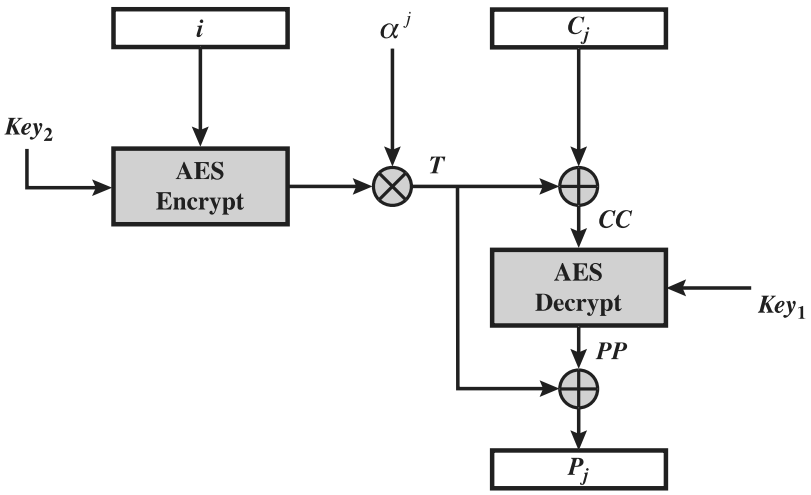
from one position to another, and an application reading the sector off the new location will still get the same plaintext sector (except perhaps the first 128 bits). For example, this means that an adversary that is allowed to read a sector from the second position but not the first can find the content of the sector in the first position by manipulating the ciphertext. Another weakness is that an adversary can flip any bit of the plaintext by flipping the corresponding ciphertext bit of the previous block, with the side-effect of "randomizing" the previous block.

Operation on a Single Block

Figure 6.9 shows the encryption and decryption of a single block. The operation involves two instances of the AES algorithm with two keys. The following parameters are associated with the algorithm.



(a) Encryption



(b) Decryption

Figure 6.9 XTS-AES Operation on Single Block

Key	The 256 or 512 bit XTS-AES key; this is parsed as a concatenation of two fields of equal size called Key_1 and Key_2, such that $Key = Key_1 \| Key_2$.
P_j	The jth block of plaintext. All blocks except possibly the final block have a length of 128 bits. A plaintext data unit, typically a disk sector, consists of a sequence of plaintext blocks $P_1, P_2, \ldots, P_m$.
C_j	The jth block of ciphertext. All blocks except possibly the final block have a length of 128 bits.
j	The sequential number of the 128-bit block inside the data unit.
i	The value of the 128-bit tweak. Each data unit (sector) is assigned a tweak value that is a nonnegative integer. The tweak values are assigned consecutively, starting from an arbitrary nonnegative integer.
α	A primitive element of GF(2^{128}) that corresponds to polynomial x (i.e., $0000 \ldots 010_2$).
α^j	a multiplied by itself j times, in GF(2^{128}).
$\oplus$	Bitwise XOR.
$\otimes$	Modular multiplication of two polynomials with binary coefficients modulo $x^{128} + x^7 + x^2 + x + 1$. Thus, this is multiplication in GF(2^{128}).

In essence, the parameter j functions much like the counter in CTR mode. It assures that if the same plaintext block appears at two different positions within a data unit, it will encrypt to two different ciphertext blocks. The parameter i functions much like a nonce at the data unit level. It assures that, if the same plaintext block appears at the same position in two different data units, it will encrypt to two different ciphertext blocks. More generally, it assures that the same plaintext data unit will encrypt to two different ciphertext data units for two different data unit positions.

The encryption and decryption of a single block can be described as

XTS-AES block operation	$T = \mathrm{E}(K_2, i) \otimes \alpha^j$ $PP = P \oplus T$ $CC = \mathrm{E}(K_1, PP)$ $C = CC \oplus T$	$T = \mathrm{E}(K_2, i) \otimes \alpha^j$ $CC = C \oplus T$ $PP = \mathrm{D}(K_1, CC)$ $P = PP \oplus T$

To see that decryption recovers the plaintext, let us expand the last line of both encryption and decryption. For encryption, we have

$$C = CC \oplus T = \mathrm{E}(K_1, PP) \oplus T = \mathrm{E}(K_1, P \oplus T) \oplus T$$

and for decryption, we have

$$P = PP \oplus T = \mathrm{D}(K_1, CC) \oplus T = \mathrm{D}(K_1, C \oplus T) \oplus T$$

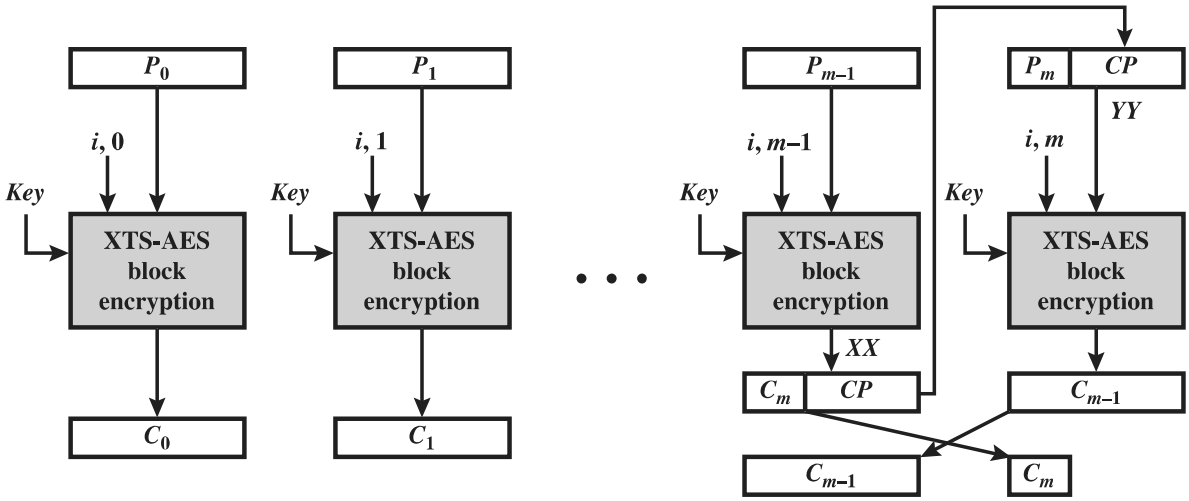
Now, we substitute for C:

$$\begin{aligned} P &= \mathrm{D}(K_1, C \oplus T) \oplus T \\ &= \mathrm{D}(K_1, [E(K_1, P \oplus T) \oplus T] \oplus T) \oplus T \\ &= \mathrm{D}(K_1, E(K_1, P \oplus T)) \oplus T \\ &= (P \oplus T) \oplus T = P \end{aligned}$$

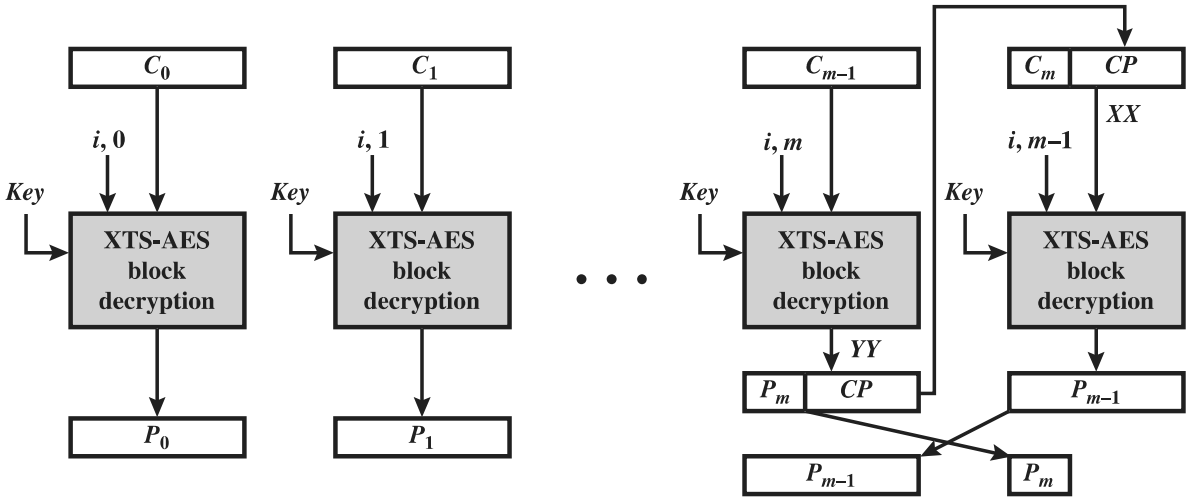
Operation on a Sector

The plaintext of a sector or data unit is organized into blocks of 128 bits. Blocks are labeled $P_0, P_1, \ldots, P_m$. The last block my be null or may contain from 1 to 127 bits. In other words, the input to the XTS-AES algorithm consists of m 128-bit blocks and possibly a final partial block.

For encryption and decryption, each block is treated independently and encrypted/decrypted as shown in Figure 6.9. The only exception occurs when the last block has less than 128 bits. In that case, the last two blocks are encrypted/decrypted using a **ciphertext-stealing** technique instead of padding. Figure 6.10 shows the scheme. P_{m-1} is the last full plaintext block, and $P_m$is the final plaintext block, which contains s bits with $1 \leq s \leq 127$. C_{m-1} is the last full ciphertext block, and $C_m$is the final ciphertext block, which contains s bits.



(a) Encryption



(b) Decryption

Figure 6.10 XTS-AES Mode

Let us label the block encryption and decryption algorithms of Figure 6.9 as

Block encryption: XTS-AES-blockEnc(K, P_j, i, j)
Block decryption: XTS-AES-blockDec(K, C_j, i, j)

Then, if the final block is null, XTS-AES mode is defined as follows:

XTS-AES mode with null final block	$C_j = \text{XTS-AES-blockEnc}(K, P_j, i, j) \quad j = 0, \ldots, m-1$
	$P_j = \text{XTS-AES-blockEnc}(K, C_j, i, j) \quad j = 0, \ldots, m-1$
XTS-AES mode with final block containing s bits	$C_j = \text{XTS-AES-blockEnc}(K, P_j, i, j) \quad j = 0, \ldots, m-2$ $XX = \text{XTS-AES-blockEnc}(K, P_{m-1}, i, m-1)$ $CP = \text{LSB}_{128-s}(XX)$ $YY = P_m \Vert CP$ $C_{m-1} = \text{XTS-AES-blockEnc}(K, YY, i, m)$ $C_m = \text{MSB}_s(XX)$
	$P_j = \text{XTS-AES-blockDec}(K, C_j, i, j) \quad j = 0, \ldots, m-2$ $YY = \text{XTS-AES-blockDec}(K, C_{m-1}, i, m-1)$ $CP = \text{LSB}_{128-s}(YY)$ $XX = C_m \Vert \text{CP}$ $P_{m-1} = \text{XTS-AES-blockDec}(K, XX, i, m)$ $P_m = \text{MSB}_s(YY)$

As can be seen, XTS-AES mode, like CTR mode, is suitable for parallel operation. Because there is no chaining, multiple blocks can be encrypted or decrypted simultaneously. Unlike CTR mode, XTS-AES mode includes a nonce (the parameter i) as well as a counter (parameter j).

6.8 RECOMMENDED WEB SITE

Recommended Web Site:

- **Block cipher modes of operation:** NIST page with full information on NIST-approved modes of operation.

6.9 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

Block cipher modes of operation cipher block chaining mode (CBC) cipher feedback mode (CFB) ciphertext stealing	counter mode (CTR) electronic codebook mode (ECB) meet-in-the-middle attack nonce	output feedback mode (OFB) Triple DES (3DES) XTS-AES mode