PART FOUR

Cryptographic Algorithms

CHAPTER 19

SYMMETRIC ENCRYPTION AND MESSAGE CONFIDENTIALITY

Symmetric encryption, also referred to as conventional encryption, secret-key, or single-key encryption, was the only type of encryption in use prior to the development of public-key encryption in the late 1970s.[1] It remains by far the most widely used of the two types of encryption.

This chapter begins with a look at a general model for the symmetric encryption process; this will enable us to understand the context within which the algorithms are used. Then we look at three important block encryption algorithms: DES, triple DES, and AES. Next, the chapter introduces symmetric stream encryption and describes the widely used stream cipher RC4. We then examine the application of these algorithms to achieve confidentiality.

19.1 SYMMETRIC ENCRYPTION PRINCIPLES

At this point the reader should review Section 2.1. Recall that a symmetric encryption scheme has five ingredients (Figure 2.1):

- **Plaintext:** This is the original message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.
- **Secret key:** The secret key is also input to the algorithm. The exact substitutions and transformations performed by the algorithm depend on the key.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts.
- **Decryption algorithm:** This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the same secret key and produces the original plaintext.

Cryptography

Cryptographic systems are generically classified along three independent dimensions:

1. **The type of operations used for transforming plaintext to ciphertext.** All encryption algorithms are based on two general principles: substitution, in which each element in the plaintext (bit, letter, group of bits or letters) is mapped into another element, and transposition, in which elements in the plaintext are rearranged. The fundamental requirement is that no information be lost (that is, that all operations be reversible). Most systems, referred to as product systems, involve multiple stages of substitutions and transpositions.
2. **The number of keys used.** If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver each use a different key, the system is referred to as asymmetric, two-key, or public-key encryption.

[1]Public-key encryption was first described in the open literature in 1976; the National Security Agency (NSA) claims to have discovered it some years earlier.

3. **The way in which the plaintext is processed.** A *block cipher* processes the input one block of elements at a time, producing an output block for each input block. A *stream cipher* processes the input elements continuously, producing output one element at a time, as it goes along.

Cryptanalysis

The process of attempting to discover the plaintext or key is known as *cryptanalysis*. The strategy used by the cryptanalyst depends on the nature of the encryption scheme and the information available to the cryptanalyst.

Table 19.1 summarizes the various types of cryptanalytic attacks, based on the amount of information known to the cryptanalyst. The most difficult problem is presented when all that is available is the *ciphertext only*. In some cases, not even the encryption algorithm is known, but in general we can assume that the opponent does know the algorithm used for encryption. One possible attack under these circumstances is the brute-force approach of trying all possible keys. If the key space is very large, this becomes impractical. Thus, the opponent must rely on an analysis of the ciphertext itself, generally applying various statistical tests to it. To use this approach, the opponent must have some general idea of the type of plaintext that is concealed, such as English or French text, an EXE file, a Java source listing, an accounting file, and so on.

Table 19.1 Types of Attacks on Encrypted Messages

Type of Attack	Known to Cryptanalyst
Ciphertext only	• Encryption algorithm • Ciphertext to be decoded
Known plaintext	• Encryption algorithm • Ciphertext to be decoded • One or more plaintext-ciphertext pairs formed with the secret key
Chosen plaintext	• Encryption algorithm • Ciphertext to be decoded • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key
Chosen ciphertext	• Encryption algorithm • Ciphertext to be decoded • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key
Chosen text	• Encryption algorithm • Ciphertext to be decoded • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key

The ciphertext-only attack is the easiest to defend against because the opponent has the least amount of information to work with. In many cases, however, the analyst has more information. The analyst may be able to capture one or more plaintext messages as well as their encryptions. Or the analyst may know that certain plaintext patterns will appear in a message. For example, a file that is encoded in the Postscript format always begins with the same pattern, or there may be a standardized header or banner to an electronic funds transfer message, and so on. All these are examples of *known plaintext*. With this knowledge, the analyst may be able to deduce the key on the basis of the way in which the known plaintext is transformed.

Closely related to the known-plaintext attack is what might be referred to as a probable-word attack. If the opponent is working with the encryption of some general prose message, he or she may have little knowledge of what is in the message. However, if the opponent is after some very specific information, then parts of the message may be known. For example, if an entire accounting file is being transmitted, the opponent may know the placement of certain key words in the header of the file. As another example, the source code for a program developed by a corporation might include a copyright statement in some standardized position.

If the analyst is able somehow to get the source system to insert into the system a message chosen by the analyst, then a *chosen-plaintext* attack is possible. In general, if the analyst is able to choose the messages to encrypt, the analyst may deliberately pick patterns that can be expected to reveal the structure of the key.

Table 19.1 lists two other types of attack: chosen ciphertext and chosen text. These are less commonly employed as cryptanalytic techniques but are nevertheless possible avenues of attack.

Only relatively weak algorithms fail to withstand a ciphertext-only attack. Generally, an encryption algorithm is designed to withstand a known-plaintext attack.

An encryption scheme is **computationally secure** if the ciphertext generated by the scheme meets one or both of the following criteria:

- The cost of breaking the cipher exceeds the value of the encrypted information.
- The time required to break the cipher exceeds the useful lifetime of the information.

Unfortunately, it is very difficult to estimate the amount of effort required to cryptanalyze ciphertext successfully. However, assuming there are no inherent mathematical weaknesses in the algorithm, then a brute-force approach is indicated, and here we can make some reasonable estimates about costs and time.

A brute-force approach involves trying every possible key until an intelligible translation of the ciphertext into plaintext is obtained. On average, half of all possible keys must be tried to achieve success. This type of attack is discussed in Section 2.1.

Feistel Cipher Structure

Many symmetric block encryption algorithms, including DES, have a structure first described by Horst Feistel of IBM in 1973 [FEIS73] and shown in Figure 19.1. 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

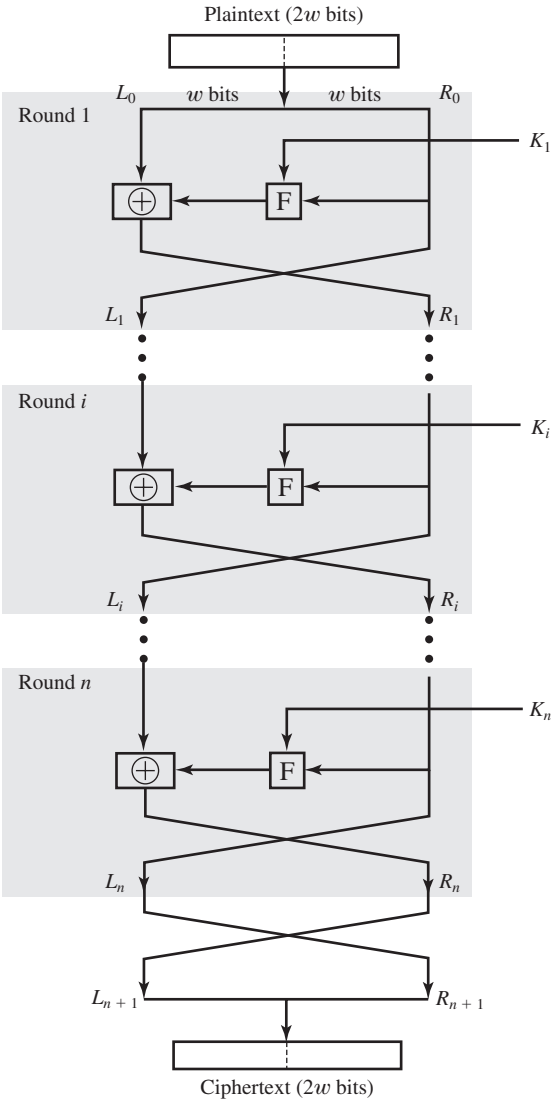


Figure 19.1 **Classical Feistel Network**

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 and are generated from the key by a subkey generation algorithm.

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 (XOR) 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. Following this

substitution, a permutation is performed that consists of the interchange of the two halves of the data.

The Feistel structure is a particular example of the more general structure used by all symmetric block ciphers. In general, a symmetric block cipher consists of a sequence of rounds, with each round performing substitutions and permutations conditioned by a secret key value. The exact realization of a symmetric block cipher 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. A block size of 128 bits is a reasonable tradeoff and is nearly universal among recent block cipher designs.
- **Key size:** Larger key size means greater security but may decrease encryption/decryption speed. The most common key length in modern algorithms is 128 bits.
- **Number of rounds:** The essence of a symmetric block 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:** Again, greater complexity generally means greater resistance to cryptanalysis.

There are two other considerations in the design of a symmetric block cipher:

- **Fast software encryption/decryption:** In many cases, encryption is embedded in applications or utility functions in such a way as to preclude a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.
- **Ease of analysis:** Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength. DES, for example, does not have an easily analyzed functionality.

Decryption with a symmetric block cipher is essentially the same as the encryption process. The rule is as follows: Use the ciphertext as input to the algorithm, but use the subkeys K_i in reverse order. That is, use K_n in the first round, K_{n-1} in the second round, and so on until K_1 is used in the last round. This is a nice feature because it means we need not implement two different algorithms, one for encryption and one for decryption.

19.2 DATA ENCRYPTION STANDARD

The most commonly used symmetric encryption algorithms are block ciphers. A block cipher processes the plaintext input in fixed-size blocks and produces a block of ciphertext of equal size for each plaintext block. This section and the

next focus on the three most important symmetric block ciphers: the Data Encryption Standard (DES) and triple DES (3DES), and the Advanced Encryption Standard (AES).

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).[2]

The DES algorithm can be described as follows. The plaintext is 64 bits in length and the key is 56 bits in length; longer plaintext amounts are processed in 64-bit blocks. The DES structure is a minor variation of the Feistel network shown in Figure 19.1. There are 16 rounds of processing. From the original 56-bit key, 16 subkeys are generated, one of which is used for each round.

The process of decryption with DES is essentially the same as the encryption process. The rule is as follows: Use the ciphertext as input to the DES algorithm, but use the subkeys K_i in reverse order. That is, use K_{16} on the first iteration, K_{15} on the second iteration, and so on until K_1 is used on the sixteenth and last iteration.

Triple DES

Triple DES (3DES) was first standardized for use in financial applications in ANSI standard X9.17 in 1985. 3DES was incorporated as part of the Data Encryption Standard in 1999, with the publication of FIPS PUB 46-3.

3DES uses three keys and three executions of the DES algorithm. The function follows an encrypt-decrypt-encrypt (EDE) sequence (Figure 19.2a):

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

where

C = ciphertext
P = plaintext
$\mathrm{E}[K, X]$ = encryption of X using key K
$\mathrm{D}[K, Y]$ = decryption of Y using key K

Decryption is simply the same operation with the keys reversed (Figure 19.2b):

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

[2]The terminology is a bit confusing. Until recently, the terms *DES* and *DEA* could be used interchangeably. However, the most recent edition of the DES document includes a specification of the DEA described here plus the triple DEA (3DES) described subsequently. Both DEA and 3DES are part of the Data Encryption Standard. Further, until the recent adoption of the official term 3*DES*, the triple DEA algorithm was typically referred to as *triple DES* and written as 3DES. For the sake of convenience, we will use 3DES.

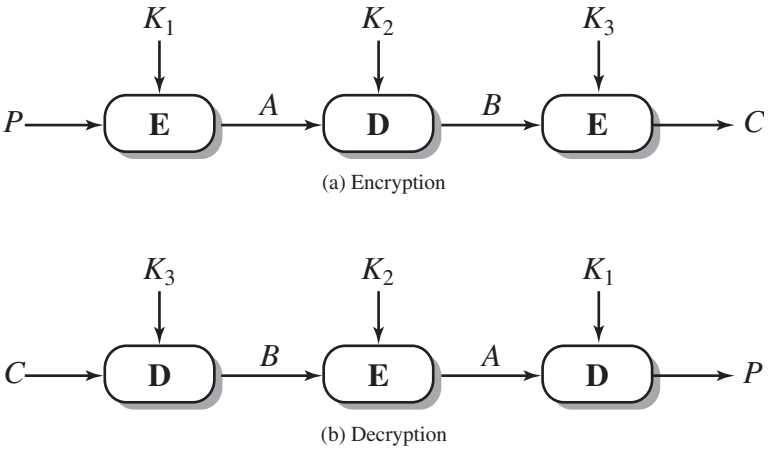


Figure 19.2 **Triple DES**

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

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

With three distinct keys, 3DES has an effective key length of 168 bits. FIPS 46-3 also allows for the use of two keys, with $K_1 = K_3$; this provides for a key length of 112 bits. FIPS 46-3 includes the following guidelines for 3DES:

- 3DES is the FIPS approved symmetric encryption algorithm of choice.
- The original DES, which uses a single 56-bit key, is permitted under the standard for legacy systems only. New procurements should support 3DES.
- Government organizations with legacy DES systems are encouraged to transition to 3DES.
- It is anticipated that 3DES and the Advanced Encryption Standard (AES) will coexist as FIPS-approved algorithms, allowing for a gradual transition to AES.

It is easy to see that 3DES is a formidable algorithm. Because the underlying cryptographic algorithm is DEA, 3DES can claim the same resistance to cryptanalysis based on the algorithm as is claimed for DEA. Further, with a 168-bit key length, brute-force attacks are effectively impossible.

Ultimately, AES is intended to replace 3DES, but this process will take a number of years. NIST anticipates that 3DES will remain an approved algorithm (for U.S. government use) for the foreseeable future.

19.3 ADVANCED ENCRYPTION STANDARD

The Advanced Encryption Standard (AES) was issued as a federal information processing standard (FIPS 197). It is intended to replace DES and triple DES with an algorithm that is more secure and efficient.

Overview of the Algorithm

AES uses a block length of 128 bits and a key length that can be 128, 192, or 256 bits. In the description of this section, we assume a key length of 128 bits, which is likely to be the one most commonly implemented.

Figure 19.3 shows the overall structure of AES. The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is

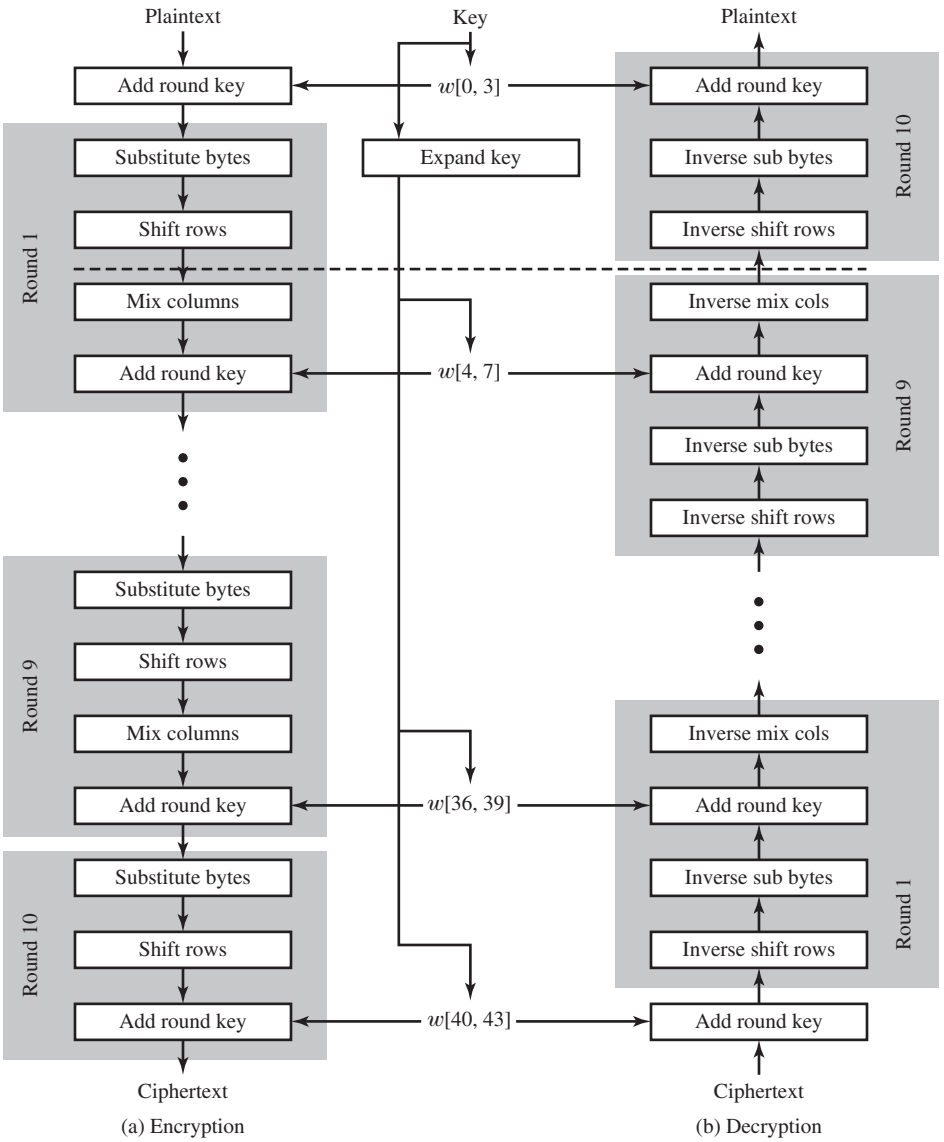


Figure 19.3 AES Encryption and Decryption

depicted as a 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. Similarly, the 128-bit key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words; each word is 4 bytes and the total key schedule is 44 words for the 128-bit key. The ordering of bytes within a matrix is by column. So, for example, the first 4 bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the **in** matrix, the second 4 bytes occupy the second column, and so on. Similarly, the first 4 bytes of the expanded key, which form a word, occupy the first column of the **w** matrix.

The following comments give some insight into AES:

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 does not use a Feistel structure but processes the entire data block in parallel 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, **w**[*i*]. Four distinct words (128 bits) serve as a round key for each round.
3. Four different stages are used, one of permutation and three of substitution:
   - **Substitute Bytes:** Uses a table, referred to as an S-box,[3] to perform a byte-by-byte substitution of the block
   - **Shift Rows:** A simple permutation that is performed row by row
   - **Mix Columns:** A substitution that alters each byte in a column as a function of all of the bytes in the column
   - **Add Round key:** 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 Add Round Key stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. Figure 19.4 depicts the structure of a full encryption round.
5. Only the Add Round Key stage makes use of the key. For this reason, the cipher begins and ends with an Add Round Key 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 Add Round Key stage by itself would not be formidable. The other three stages together scramble the bits, 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 (Add Round Key) of a block, followed by scrambling of the

[3]The term *S-box*, or substitution box, is commonly used in the description of symmetric ciphers to refer to a table used for a table-lookup type of substitution mechanism.

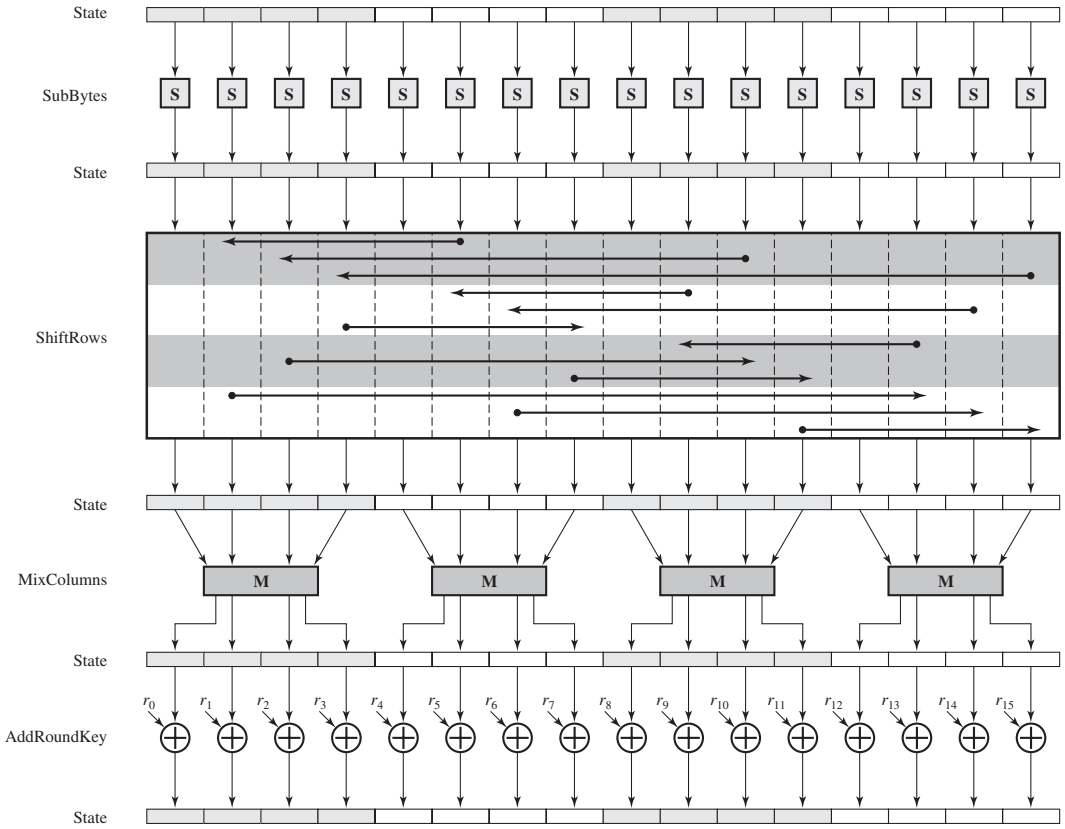


Figure 19.4 **AES Encryption Round**

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, Shift Row, and Mix Columns stages, an inverse function is used in the decryption algorithm. For the Add Round Key stage, the inverse is achieved by XORing the same round key to the block, using the result that $A \oplus A \oplus B = B$.
8. As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not 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 19.3 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.

Algorithm Details

We now look briefly at the principal elements of AES in more detail.

Substitute Bytes Transformation The **forward substitute byte transformation**, called SubBytes, is a simple table lookup. AES defines a 16×16 matrix of byte values, called an S-box (Table 19.2a), 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[4] {95} references row 9, column 5 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

The S-box is constructed using properties of finite fields. The topic of finite fields is beyond the scope of this book; it is discussed in detail in [STAL06a].

The **inverse substitute byte transformation**, called InvSubBytes, makes use of the inverse S-box shown in Table 19.2b. Note, for example, that the input {2A} produces the output {95}, and the input {95} to the S-box produces {2A}.

The S-box is designed to be resistant to known cryptanalytic attacks. Specifically, the AES developers sought a design that has a low correlation between input bits and output bits and the property that the output cannot be described as a simple mathematical function of the input.

Shift Row Transformation For the **forward shift row transformation**, called ShiftRows, 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.

[4]In FIPS PUB 197, a hexadecimal number is indicated by enclosing it in curly brackets. We use that convention.

Table 19.2 AES S-Boxes

(a) S-box

		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	BI	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

(b) Inverse 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	FA
	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

For the third 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.

The shift row transformation is more substantial than it may first appear. This is because the **State**, as well as the cipher input and output, is treated as an array of four 4-byte columns. Thus, on encryption, the first 4 bytes of the plaintext are copied to the first column of **State**, and so on. Further, as will be seen, the round key is applied to **State** column by column. Thus, a row shift moves an individual byte from one column to another, which is a linear distance of a multiple of 4 bytes. Also note that the transformation ensures that the 4 bytes of one column are spread out to four different columns.

Mix Column Transformation 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 4 bytes in the column. The mapping makes use of equations over finite fields. The following is an example of MixColumns:

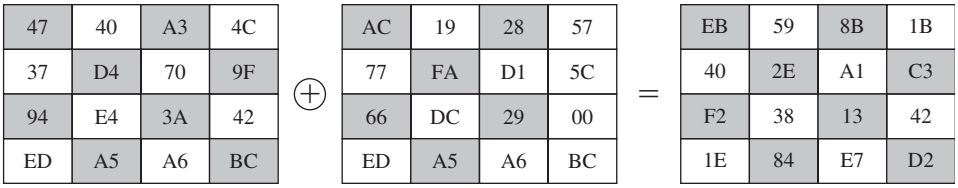
87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

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

The mapping is designed to provide a good mixing among the bytes of each column. The mix column transformation combined with the shift row transformation ensures that after a few rounds, all output bits depend on all input bits.

Add Round Key Transformation In the **forward add round key transformation**, called AddRoundKey, the 128 bits of **State** are bitwise XORed with the 128 bits of the round key. The operation is viewed as a column-wise operation between the four bytes of a **State** column and one word of the round key; it can also be viewed as a byte-level operation. The following is an example of AddRoundKey:



The first matrix is **State**, and the second matrix is the round key.

The **inverse add round key transformation** is identical to the forward add round key transformation, because the XOR operation is its own inverse.

The add round key transformation is as simple as possible and affects every bit of **State**. The complexity of the round key expansion, plus the complexity of the other stages of AES, ensure security.

AES Key Expansion The AES key expansion algorithm takes as input a 4-word (16-byte) key and produces a linear array of 44 words (156 bytes). This is sufficient to provide a 4-word round key for the initial Add Round Key stage and each of the 10 rounds of the cipher.

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]. A complex finite-field algorithm is used in generating the expanded key.

19.4 STREAM CIPHERS AND RC4

A *block cipher* processes the input one block of elements at a time, producing an output block for each input block. A *stream cipher* processes the input elements continuously, producing output one element at a time, as it goes along. Although block ciphers are far more common, there are certain applications in which a stream cipher is more appropriate. Examples are given subsequently in this book. In this section we look at perhaps the most popular symmetric stream cipher, RC4. We begin with an overview of stream cipher structure and then examine RC4.

Stream Cipher Structure

A typical stream cipher encrypts plaintext 1 byte at a time, although a stream cipher may be designed to operate on 1 bit at a time or on units larger than a byte at a time. Figure 2.3b is a representative diagram of stream cipher structure. In this structure a key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random. A pseudorandom stream is one that is unpredictable without knowledge of the input key and that has an apparently random character. The output of the generator, called a **keystream**, is combined 1 byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation. For example, if the next byte generated by the generator is 01101100 and the next plaintext byte is 11001100, then the resulting ciphertext byte is

```
  11001100  plaintext
⊕ 01101100  key stream
  10100000  ciphertext
```

Decryption requires the use of the same pseudorandom sequence:

```
  10100000  ciphertext
⊕ 01101100  key stream
  11001100  plaintext
```

[KUMA97] lists the following important design considerations for a stream cipher:

1. The encryption sequence should have a large period. A pseudorandom number generator uses a function that produces a deterministic stream of bits that eventually repeats. The longer the period of repeat, the more difficult it will be to do cryptanalysis.
2. The keystream should approximate the properties of a true random number stream as close as possible. For example, there should be an approximately equal number of 1s and 0s. If the keystream is treated as a stream of bytes, then all of the 256 possible byte values should appear approximately equally often. The more random-appearing the keystream is, the more randomized the ciphertext is, making cryptanalysis more difficult.
3. Note from Figure 2.3b that the output of the pseudorandom number generator is conditioned on the value of the input key. To guard against brute-force attacks, the key needs to be sufficiently long. The same considerations as apply for block ciphers are valid here. Thus, with current technology, a key length of at least 128 bits is desirable.

With a properly designed pseudorandom number generator, a stream cipher can be as secure as block cipher of comparable key length. The primary advantage of a stream cipher is that stream ciphers are almost always faster and use far less code than do block ciphers. The example in this section, RC4, can be implemented in just a few lines of code. Table 19.3 compares execution times of

Table 19.3 Speed Comparisons of Symmetric Ciphers on a Pentium 4

Cipher	Key Length	Speed (Mbps)
DES	56	21
3DES	168	10
AES	128	61
RC4	Variable	113

Source: http://www.cryptopp.com/benchmarks.html

RC4 with three well-known symmetric block ciphers. The advantage of a block cipher is that you can reuse keys. However, if two plaintexts are encrypted with the same key using a stream cipher, then cryptanalysis is often quite simple [DAWS96]. If the two ciphertext streams are XORed together, the result is the XOR of the original plaintexts. If the plaintexts are text strings, credit card numbers, or other byte streams with known properties, then cryptanalysis may be successful.

For applications that require encryption/decryption of a stream of data, such as over a data communications channel or a browser/Web link, a stream cipher might be the better alternative. For applications that deal with blocks of data, such as file transfer, e-mail, and database, block ciphers may be more appropriate. However, either type of cipher can be used in virtually any application.

The RC4 Algorithm

RC4 is a stream cipher designed in 1987 by Ron Rivest for RSA Security. It is a variable-key-size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation. Analysis shows that the period of the cipher is overwhelmingly likely to be greater than 10^{100} [ROBS95a]. Eight to sixteen machine operations are required per output byte, and the cipher can be expected to run very quickly in software. RC4 is used in the SSL/TLS (Secure Sockets Layer/Transport Layer Security) standards that have been defined for communication between Web browsers and servers. It is also used in the WEP (Wired Equivalent Privacy) protocol and the newer WiFi Protected Access (WPA) protocol that are part of the IEEE 802.11 wireless LAN standard. RC4 was kept as a trade secret by RSA Security. In September 1994, the RC4 algorithm was anonymously posted on the Internet on the Cypherpunks anonymous remailers list.

The RC4 algorithm is remarkably simply and quite easy to explain. A variable-length key of from 1 to 256 bytes (8 to 2048 bits) is used to initialize a 256-byte state vector **S**, with elements **S**[0], **S**[1], . . . , **S**[255]. At all times, **S** contains a permutation of all 8-bit numbers from 0 through 255. For encryption and decryption, a byte *k* (see Figure 2.3b) is generated from **S** by selecting one of the 255 entries in a systematic fashion. As each value of *k* is generated, the entries in **S** are once again permuted.

Initialization of S To begin, the entries of **S** are set equal to the values from 0 through 255 in ascending order; that is, **S**[0] = 0, **S**[1] = 1, . . . , **S**[255] = 255. A temporary vector, T, is also created. If the length of the key K is 256 bytes, then K is transferred to T. Otherwise, for a key of length *keylen* bytes, the first *keylen* elements of T are copied from K and then K is repeated as many times as necessary to fill out T. These preliminary operations can be summarized as follows:

```
/* Initialization */
for i = 0 to 255 do
S[i] = i;
T[i] = K[i mod keylen];
```

Next we use T to produce the initial permutation of S. This involves starting with S[0] and going through to S[255], and, for each **S**[i], swapping **S**[i] with another byte in **S** according to a scheme dictated by T[i]:

```
/* Initial Permutation of S */
j = 0;
for i = 0 to 255 do
  j = (j + S[i] + T[i]) mod 256;
  Swap (S[i], S[j]);
```

Because the only operation on **S** is a swap, the only effect is a permutation. **S** still contains all the numbers from 0 through 255.

Stream Generation Once the **S** vector is initialized, the input key is no longer used. Stream generation involves cycling through all the elements of **S**[i], and, for each **S**[i], swapping **S**[i] with another byte in **S** according to a scheme dictated by the current configuration of S. After S[255] is reached, the process continues, starting over again at S[0]:

```
/* Stream Generation */
i, j = 0;
while (true)
   i = (i + 1) mod 256;
   j = (j + S[i]) mod 256;
   Swap (S[i], S[j]);
   t = (S[i] + S[j]) mod 256;
   k = S[t];
```

To encrypt, XOR the value *k* with the next byte of plaintext. To decrypt, XOR the value *k* with the next byte of ciphertext.

Figure 19.5 illustrates the RC4 logic.

Strength of RC4 A number of papers have been published analyzing methods of attacking RC4. None of these approaches is practical against RC4 with a reasonable key length, such as 128 bits. A more serious problem is reported in [FLUH01]. The authors demonstrate that the WEP protocol, intended to provide confidentiality on 802.11 wireless LAN networks, is vulnerable to a particular attack approach. In essence, the problem is not with RC4 itself but the way in which keys are generated for use as input to RC4. This particular problem does not appear to be relevant to other applications using RC4 and can be remedied in WEP by changing the way in which keys are generated. This problem points out the difficulty in designing a secure system that involves both cryptographic functions and protocols that make use of them.

19.5 CIPHER BLOCK MODES OF OPERATION

A symmetric block cipher processes one block of data at a time. In the case of DES and 3DES, the block length is 64 bits. For longer amounts of plaintext, it is necessary to break the plaintext into 64-bit blocks (padding the last block if necessary). To apply a block cipher in a variety of applications, five **modes of operation** have been

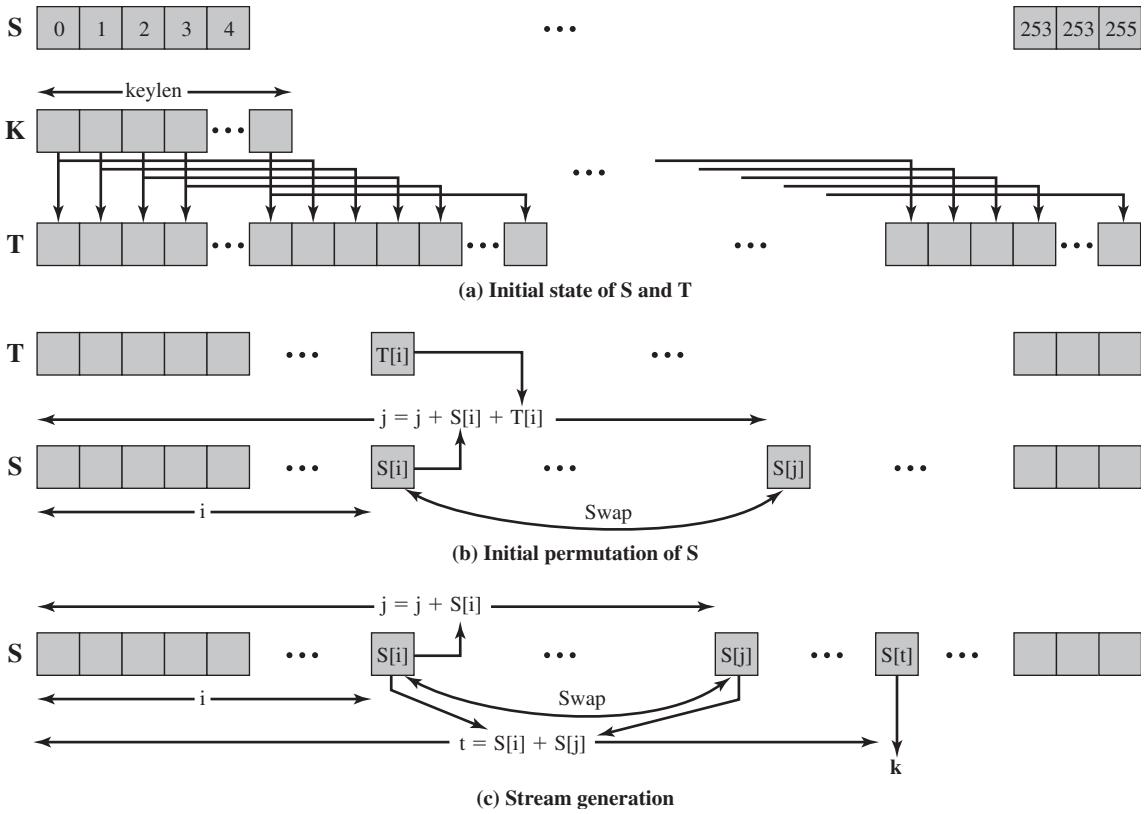


Figure 19.5 **RC4**

defined by NIST (Special Publication 800-38A). The five modes are intended to cover virtually all the possible 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 19.4, and the most important are described briefly in the remainder of this section.

Electronic Codebook Mode

The simplest way to proceed is what is known as electronic codebook (ECB) mode, in which plaintext is handled b bits at a time and each block of plaintext is encrypted using the same key (Figure 2.3a). The term *codebook* is used because, for a given key, there is a unique ciphertext for every b-bit block of plaintext. Therefore, one can imagine a gigantic codebook in which there is an entry for every possible b-bit plaintext pattern showing its corresponding ciphertext.

With ECB, if the same b-bit block of plaintext appears more than once in the message, it always produces the same ciphertext. Because of this, 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

Table 19.4 Block Cipher Modes of Operation

Mode	Description	Typical Application
Electronic Code book (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 transimission • 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 DES output.	• 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

elements can be identified by the analyst. This may help in the analysis or may provide an opportunity for substituting or rearranging blocks.

To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks.

Cipher Block Chaining Mode

In the cipher block chaining (CBC) mode (Figure 19.6), 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.

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

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

where $\mathrm{E}[K, X]$ is the encryption of plaintext X using key K, and $\oplus$ is the exclusive-OR operation. Then

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

which verifies Figure 19.6b.

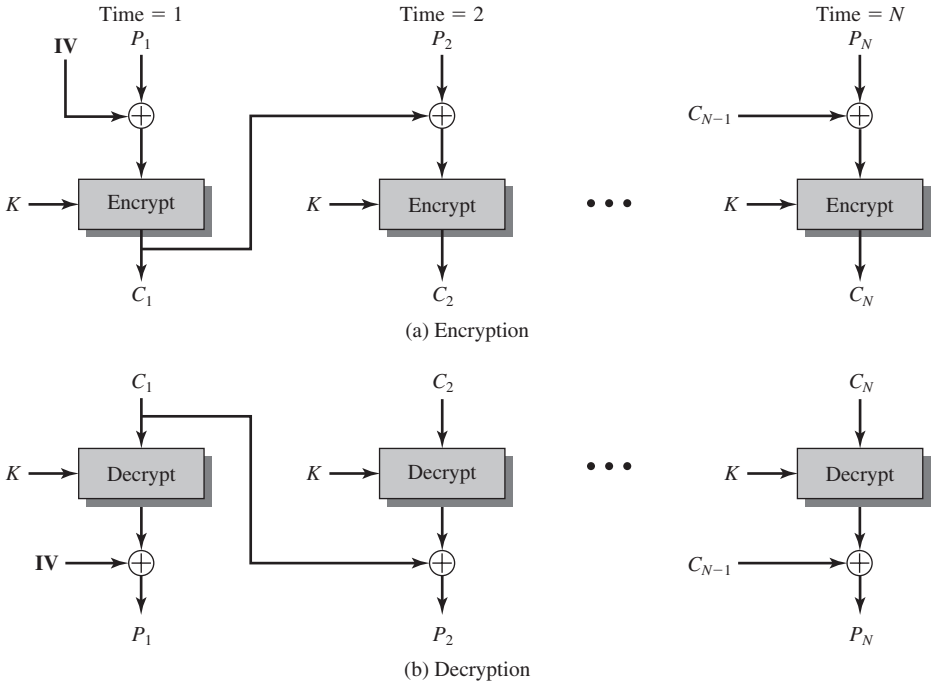


Figure 19.6 **Cipher Block Chaining (CBC) Mode**

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 must be known to both the sender and receiver. For maximum security, the IV should be protected as well as the key. 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 the following:

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

Now use the notation that $X[j]$ denotes the jth 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.

Cipher Feedback Mode

It is possible to convert any block cipher into a stream cipher by using the cipher feedback (CFB) 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 using 8 bits. If more than 8 bits are used, transmission capacity is wasted.

Figure 19.7 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

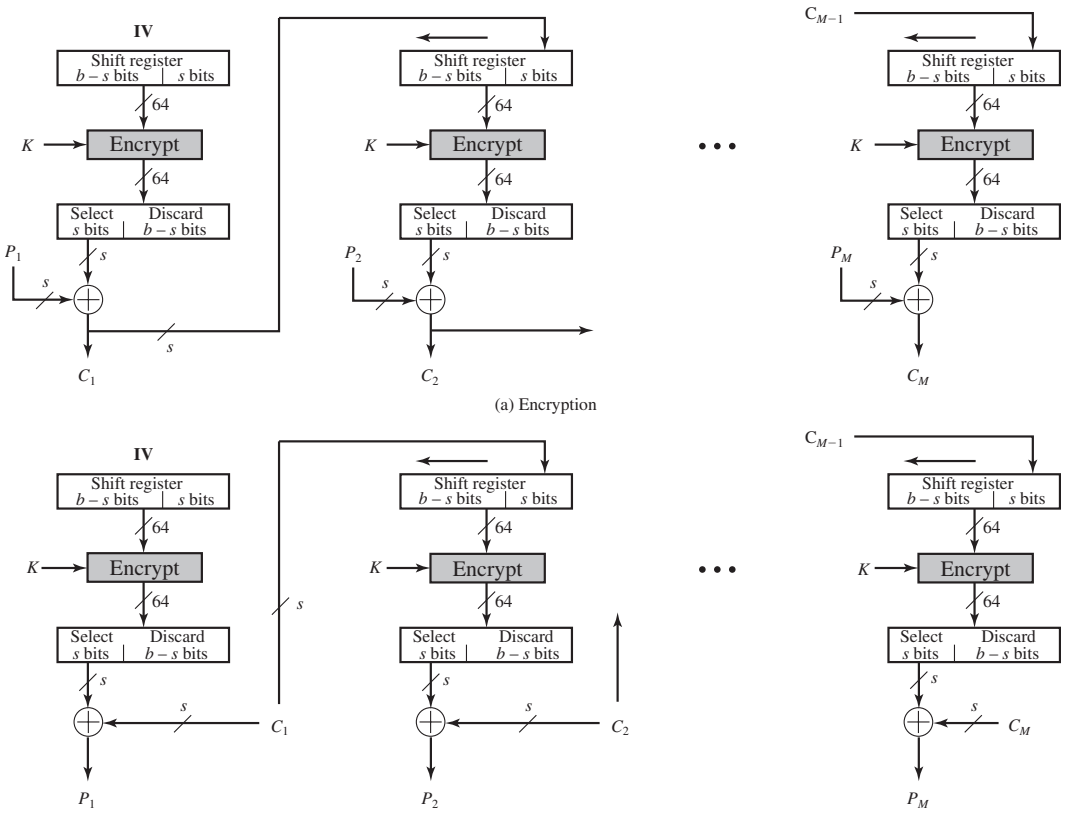


Figure 19.7 ***s*-bit Cipher Feedback (CFB) Mode**

are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext.

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 unit 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 $S_s(X)$ be defined as the most significant s bits of X. Then

$$C_1 = P_1 \oplus S_s[E(K, IV)]$$

Therefore,

$$P_1 = C_1 \oplus S_s[E(K, IV)]$$

The same reasoning holds for subsequent steps in the process.

Counter Mode

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

Figure 19.8 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.

[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

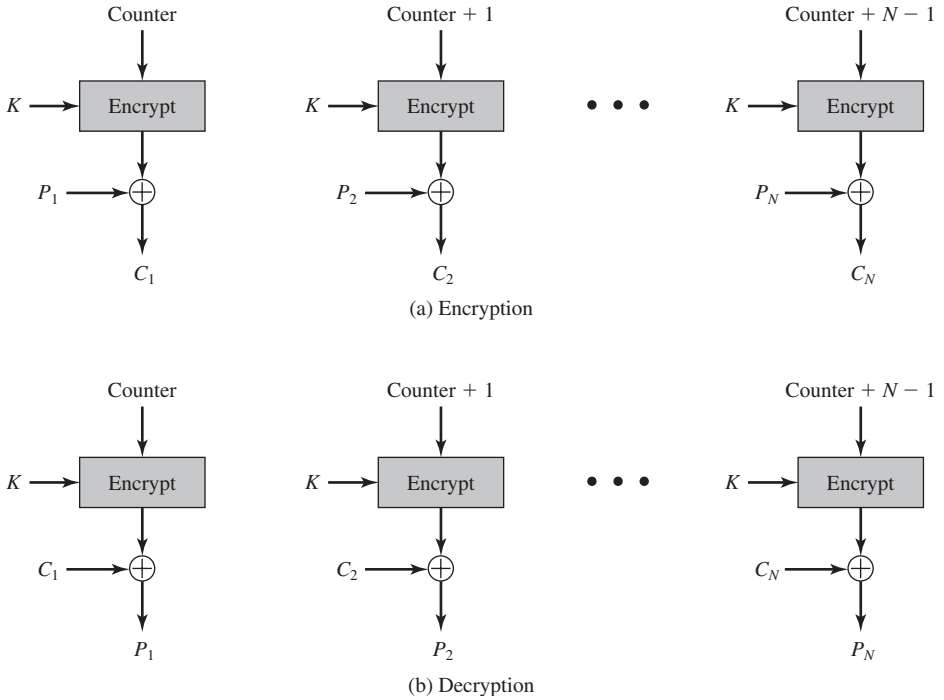


Figure 19.8 Counter (CTR) Mode

available and security is maintained, preprocessing can be used to prepare the output of the encryption boxes that feed into the XOR functions in Figure 19.8. 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.

19.6 LOCATION OF SYMMETRIC ENCRYPTION DEVICES

The most powerful, and most common, approach to countering the threats to network security is encryption. In using encryption, we need to decide what to encrypt and where the encryption gear should be located. There are two fundamental alternatives:

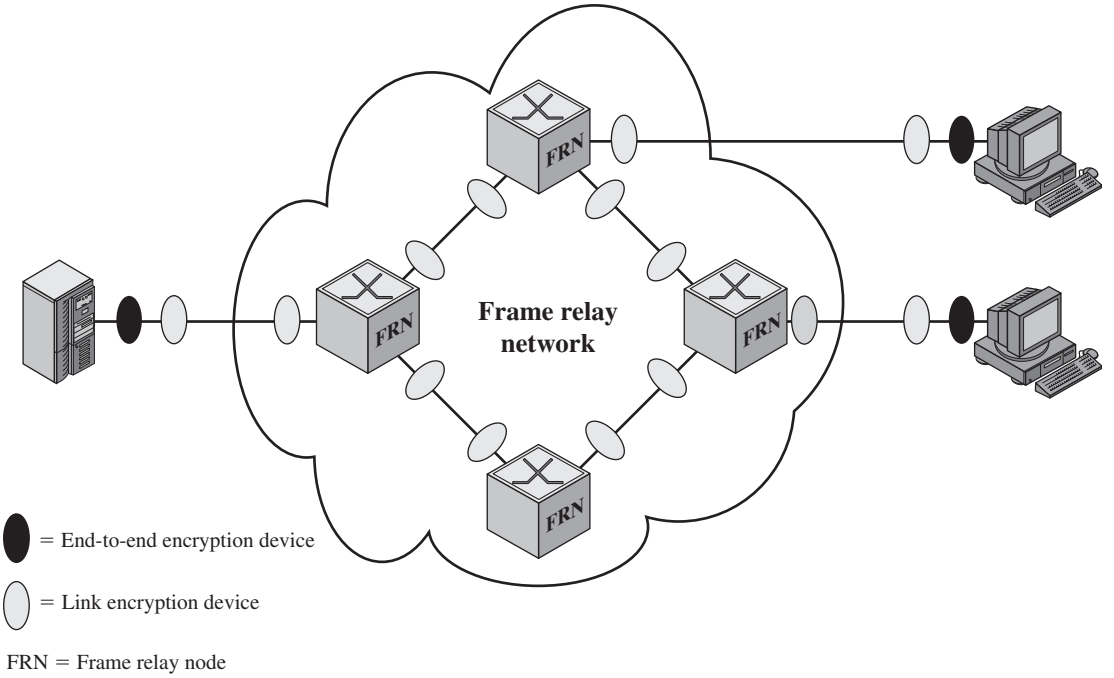


Figure 19.9 **Encryption Across a Frame Relay Network**

link encryption and end-to-end encryption; these are illustrated in use over a frame network in Figure 19.9.

With link encryption, each vulnerable communications link is equipped on both ends with an encryption device. Thus, all traffic over all communications links is secured. Although this requires a lot of encryption devices in a large network, it provides a high level of security. One disadvantage of this approach is that the message must be decrypted each time it enters a frame switch; this is necessary because the switch must read the address (connection identifier) in the frame header to route the frame. Thus, the message is vulnerable at each switch. If this is a public frame-relay network, the user has no control over the security of the nodes.

With end-to-end encryption, the encryption process is carried out at the two end systems. The source host or terminal encrypts the data. The data, in encrypted form, are then transmitted unaltered across the network to the destination terminal or host. The destination shares a key with the source and so is able to decrypt the data. This approach would seem to secure the transmission against attacks on the network links or switches. There is, however, still a weak spot.

Consider the following situation. A host connects to a frame relay network, sets up a logical data link connection to another host, and is prepared to transfer data to that other host using end-to-end encryption. Data are transmitted over such a network in the form of frames, consisting of a header and some user data. What part of each frame will the host encrypt? Suppose that the host encrypts the entire frame, including the header. This will not work because, remember, only the other host can perform the decryption. The frame relay node will receive an encrypted frame and be unable to read the header. Therefore, it will not be able to route the

frame. It follows that the host may only encrypt the user data portion of the frame and must leave the header in the clear, so that it can be read by the network.

Thus, with end-to-end encryption, the user data are secure. However, the traffic pattern is not, because frame headers are transmitted in the clear. To achieve greater security, both link and end-to-end encryption are needed, as shown in Figure 19.9.

To summarize, when both forms are employed, the host encrypts the user data portion of a frame using an end-to-end encryption key. The entire frame is then encrypted using a link encryption key. As the frame traverses the network, each switch decrypts the frame using a link encryption key to read the header and then encrypts the entire frame again for sending it out on the next link. Now the entire frame is secure except for the time that the frame is actually in the memory of a frame switch, at which time the frame header is in the clear.

19.7 KEY DISTRIBUTION

For symmetric encryption to work, the two parties to an exchange must share the same key, and that key must be protected from access by others. Furthermore, frequent key changes are usually desirable to limit the amount of data compromised if an attacker learns the key. Therefore, the strength of any cryptographic system rests with the key distribution technique, a term that refers to the means of delivering a key to two parties that wish to exchange data, without allowing others to see the key. Key distribution can be achieved in a number of ways. For two parties A and B,

1. A key could be selected by A and physically delivered to B.
2. A third party could select the key and physically deliver it to A and B.
3. If A and B have previously and recently used a key, one party could transmit the new key to the other, encrypted using the old key.
4. If A and B each have an encrypted connection to a third party C, C could deliver a key on the encrypted links to A and B.

Options 1 and 2 call for manual delivery of a key. For link encryption, this is a reasonable requirement, because each link encryption device is only going to be exchanging data with its partner on the other end of the link. However, for end-to-end encryption, manual delivery is awkward. In a distributed system, any given host or terminal may need to engage in exchanges with many other hosts and terminals over time. Thus, each device needs a number of keys, supplied dynamically. The problem is especially difficult in a wide area distributed system.

Option 3 is a possibility for either link encryption or end-to-end encryption, but if an attacker ever succeeds in gaining access to one key, then all subsequent keys are revealed. Even if frequent changes are made to the link encryption keys, these should be done manually. To provide keys for end-to-end encryption, option 4 is preferable.

Figure 19.10 illustrates an implementation that satisfies option 4 for end-to-end encryption. In the figure, link encryption is ignored. This can be added, or not, as required. For this scheme, two kinds of keys are identified:

- **Session key:** When two end systems (hosts, terminals, etc.) wish to communicate, they establish a logical connection (e.g., virtual circuit). For the duration of

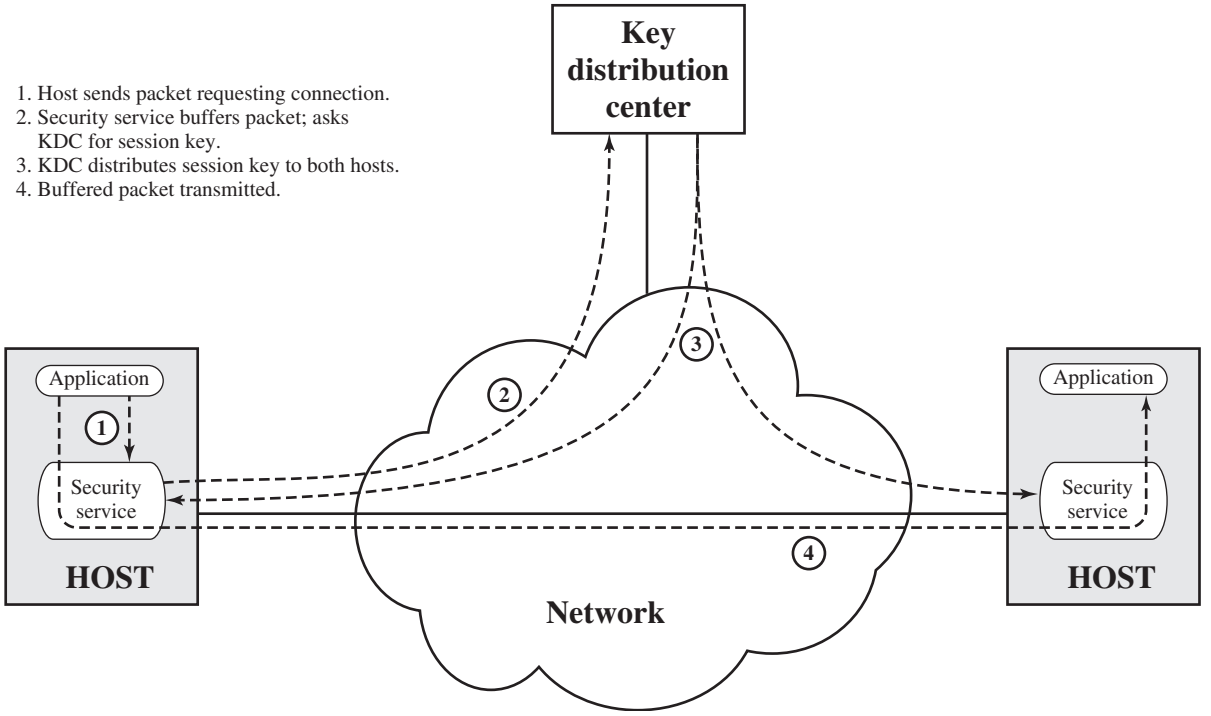


Figure 19.10 **Automatic Key Distribution for Connection-Oriented Protocd**

that logical connection, all user data are encrypted with a one-time session key. At the conclusion of the session, or connection, the session key is destroyed.

- **Permanent key:** A permanent key is a key used between entities for the purpose of distributing session keys.

The configuration consists of the following elements:

- **Key distribution center:** The key distribution center (KDC) determines which systems are allowed to communicate with each other. When permission is granted for two systems to establish a connection, the key distribution center provides a one-time session key for that connection.
- **Security service module (SSM):** This module, which may consist of functionality at one protocol layer, performs end-to-end encryption and obtains session keys on behalf of users.

The steps involved in establishing a connection are shown in Figure 19.10. When one host wishes to set up a connection to another host, it transmits a connection-request packet (step 1). The SSM saves that packet and applies to the KDC for permission to establish the connection (step 2). The communication between the SSM and the KDC is encrypted using a master key shared only by this SSM and the KDC. If the KDC approves the connection request, it generates the session key and delivers it to the two appropriate SSMs, using a unique permanent key for each SSM (step 3). The requesting SSM can now release the connection request packet, and a connection is set

up between the two end systems (step 4). All user data exchanged between the two end systems are encrypted by their respective SSMs using the one-time session key.

The automated key distribution approach provides the flexibility and dynamic characteristics needed to allow a number of terminal users to access a number of hosts and for the hosts to exchange data with each other.

Another approach to key distribution uses public-key encryption, which is discussed in Chapter 3.

19.8 RECOMMENDED READING AND WEB SITES

The topics in this chapter are covered in greater detail in [STAL06a]. A worthwhile and detailed survey is [MENE97]. A more in-depth treatment, with rigorous mathematical discussion, is [STIN06].

MENE97 Menezes, A.; van Oorschot, P.; and Vanstone, S. *Handbook of Applied Cryptography.* Boca Raton, FL: CRC Press, 1997.

STAL06a Stallings, W. *Cryptography and Network Security: Principles and Practice, Fourth Edition.* Upper Saddle River, NJ: Prentice Hall, 2006.

STIN06 Stinson, D. *Cryptography: Theory and Practice.* Boca Raton, FL: CRC Press, 2006.

Recommended Web sites:

- **AES home page:** NIST's page on AES. Contains the standard plus a number of other relevant documents.
- **AES Lounge:** Contains a comprehensive bibliography of documents and papers on AES, with access to electronic copies.
- **Block Cipher Modes of Operation:** NIST page with full information on NIST-approved modes of operation.

19.9 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

Advanced Encryption Standard (AES)	counter mode	key distribution
block cipher	Cryptanalysis	link encryption
brute-force attack	cryptography Data Encryption Standard (DES)	plaintext
computationally secure	decryption	RC4
cipher block chaining (CBC) mode	electronic codebook (ECB) mode	session key
cipher feedback (CFB) mode	encryption	stream cipher
ciphertext	end-to-end encryption	subkey
	Feistel cipher	symmetric encryption
		triple DES (3DES)

Review Questions

19.1 What are the essential ingredients of a symmetric cipher?
19.2 What are the two basic functions used in encryption algorithms?
19.3 How many keys are required for two people to communicate via a symmetric cipher?
19.4 What is the difference between a block cipher and a stream cipher?
19.5 What are the two general approaches to attacking a cipher?
19.6 Why do some block cipher modes of operation only use encryption while others use both encryption and decryption?
19.7 What is triple encryption?
19.8 Why is the middle portion of 3DES a decryption rather than an encryption?
19.9 What is the difference between link and end-to-end encryption?
19.10 List ways in which secret keys can be distributed to two communicating parties.
19.11 What is the difference between a session key and a master key?
19.12 What is a key distribution center?

Problems

19.1 Show that Feistel decryption is the inverse of Feistel encryption.

19.2 Consider a Feistel cipher composed of 16 rounds with block length 128 bits and key length 128 bits. Suppose that, for a given k, the key scheduling algorithm determines values for the first 8 round keys, $k_1, k_2, \ldots k_8$, and then sets

$$k_9 = k_8, k_{10} = k_7, k_{11} = k_6, \ldots, k_{16} = k_1$$

Suppose you have a ciphertext c. Explain how, with access to an encryption oracle, you can decrypt c and determine m using just a single oracle query. This shows that such a cipher is vulnerable to a chosen plaintext attack. (An encryption oracle can be thought of as a device that, when given a plaintext, returns the corresponding ciphertext. The internal details of the device are not known to you and you cannot break open the device. You can only gain information from the oracle by making queries to it and observing its responses.)

19.3 For any block cipher, the fact that it is a nonlinear function is crucial to its security. To see this, suppose that we have a linear block cipher EL that encrypts 128-bit blocks of plaintext into 128-bit blocks of ciphertext. Let $\text{EL}(k, m)$ denote the encryption of a 128-bit message m under a key k (the actual bit length of k is irrelevant). Thus

$$\text{EL}(k, [m_1 \oplus m_2]) = \text{EL}(k, m_1) \oplus \text{EL}(k, m_1) \text{ for all 128-bit patterns } m_1, m_2$$

Describe how, with 128 chosen ciphertexts, an adversary can decrypt any ciphertext without knowledge of the secret key k. (A "chosen ciphertext" means that an adversary has the ability to choose a ciphertext and then obtain its decryption. Here, you have 128 plaintext/ciphertext pairs to work with and you have the ability to chose the value of the ciphertexts.)

19.4 What RC4 key value will leave S unchanged during initialization? That is, after the initial permutation of S, the entries of S will be equal to the values from 0 through 255 in ascending order.

19.5 RC4 has a secret internal state which is a permutation of all the possible values of the vector **S** and the two indices i and j.

a. Using a straightforward scheme to store the internal state, how many bits are used?
b. Suppose we think of it from the point of view of how much information is represented by the state. In that case, we need to determine how may different states there are, then take the log to the base 2 to find out how many bits of information this represents. Using this approach, how many bits would be needed to represent the state?

19.6 With the ECB mode, if there is an error in a block of the transmitted ciphertext, only the corresponding plaintext block is affected. However, in the CBC mode, this error propagates. For example, an error in the transmitted C_1 (Figure 19.6) obviously corrupts P_1 and P_{21}.

a. Are any blocks beyond P_2 affected?

b. Suppose that there is a bit error in the source version of P_1. Through how many ciphertext blocks is this error propagated? What is the effect at the receiver?

19.7 Suppose an error occurs in a block of ciphertext on transmission using CBC. What effect is produced on the recovered plaintext blocks?

19.8 You want to build a hardware device to do block encryption in the cipher block chaining (CBC) mode using an algorithm stronger than DES. 3DES is a good candidate. Figure 19.11 shows two possibilities, both of which follow from the definition of CBC. Which of the two would you choose

a. For security?

b. For performance?

19.9 Can you suggest a security improvement to either option in Figure 19.11, using only three DES chips and some number of XOR functions? Assume you are still limited to two keys.

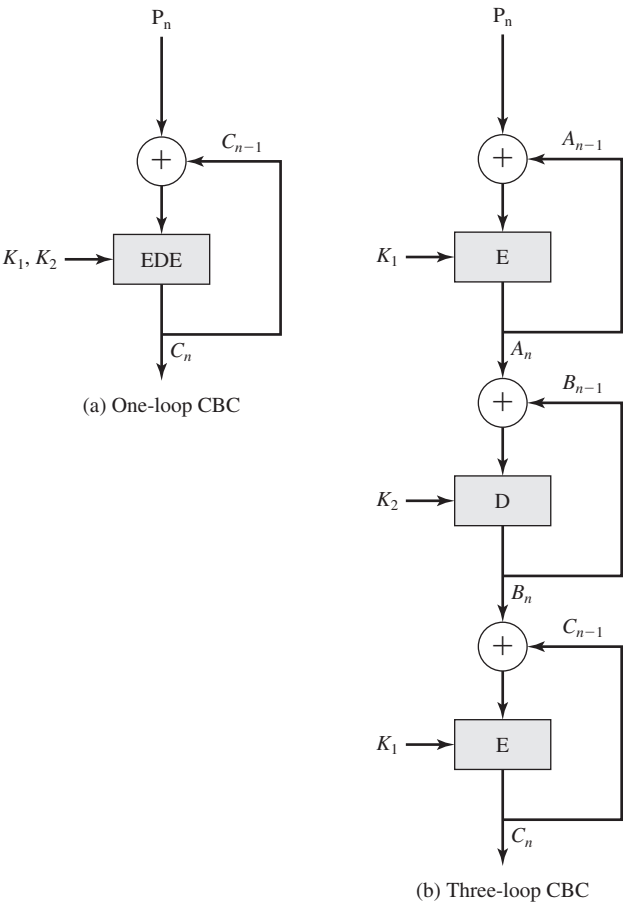


Figure 19.11 Use of Triple DES in CBC Mode

19.10 Fill in the remainder of this table:

Mode	Encrypt	Decrypt
ECB	$C_j = E(K, P_j) \quad j = 1, \ldots, N$	$P_j = D(K, C_j) \quad j = 1, \ldots, N$
CBC	$C_1 = E(K, [P_1 \oplus IV])$ $C_j = E(K, [P_j \oplus C_{j-1}]) \quad j = 2, \ldots, N$	$P_1 = D(K, C_1) \oplus IV$ $P_j = D(K, C_j) \oplus C_{j-1} \quad j = 2, \ldots, N$
CFB		
CTR		

19.11 CBC-Pad is a block cipher mode of operation used in the RC5 block cipher, but it could be used in any block cipher. CBC-Pad handles plaintext of any length. The ciphertext is longer then the plaintext by at most the size of a single block. Padding is used to assure that the plaintext input is a multiple of the block length. It is assumed that the original plaintext is an integer number of bytes. This plaintext is padded at the end by from 1 to *bb* bytes, where *bb* equals the block size in bytes. The pad bytes are all the same and set to a byte that represents the number of bytes of padding. For example, if there are 8 bytes of padding, each byte has the bit pattern 00001000. Why not allow zero bytes of padding? That is, if the original plaintext is an integer multiple of the block size, why not refrain from padding?

19.12 Padding may not always be appropriate. For example, one might wish to store the encrypted data in the same memory buffer that originally contained the plaintext. In that case, the ciphertext must be the same length as the original plaintext. A mode for that purpose is the ciphertext stealing (CTS) mode. Figure 19.12a shows an implementation of this mode.

a. Explain how it works.

b. Describe how to decrypt C_{n-1} and C_n.

19.13 Figure 19.12b shows an alternative to CTS for producing ciphertext of equal length to the plaintext when the plaintext is not an integer multiple of the block size.

a. Explain the algorithm.

b. Explain why CTS is preferable to this approach illustrated in Figure 19.12b.

19.14 If a bit error occurs in the transmission of a ciphertext character in 8-bit CFB mode, how far does the error propagate?

19.15 One of the most widely used message authentication codes (MACs), referred to as the Data Authentication Algorithm, is based on DES. The algorithm is both a FIPS publication (FIPS PUB 113) and an ANSI standard (X9.17). The algorithm can be defined as using the cipher block chaining (CBC) mode of operation of DES with an initialization vector of zero (Figure 19.6). The data (e.g., message, record, file, or program) to be authenticated are grouped into contiguous 64-bit blocks: $P_1, P_2, \ldots, P_N$. If necessary, the final block is padded on the right with 0s to form a full 64-bit block. The MAC consists of either the entire ciphertext block C_N or the leftmost M bits of the block, with $16 \leq M \leq 64$. Show that the same result can be produced using the cipher feedback mode.

19.16 Key distribution schemes using an access control center and/or a key distribution center have central points vulnerable to attack. Discuss the security implications of such centralization.

19.17 Suppose that someone suggests the following way to confirm that the two of you are both in possession of the same secret key. You create a random bit string the length of

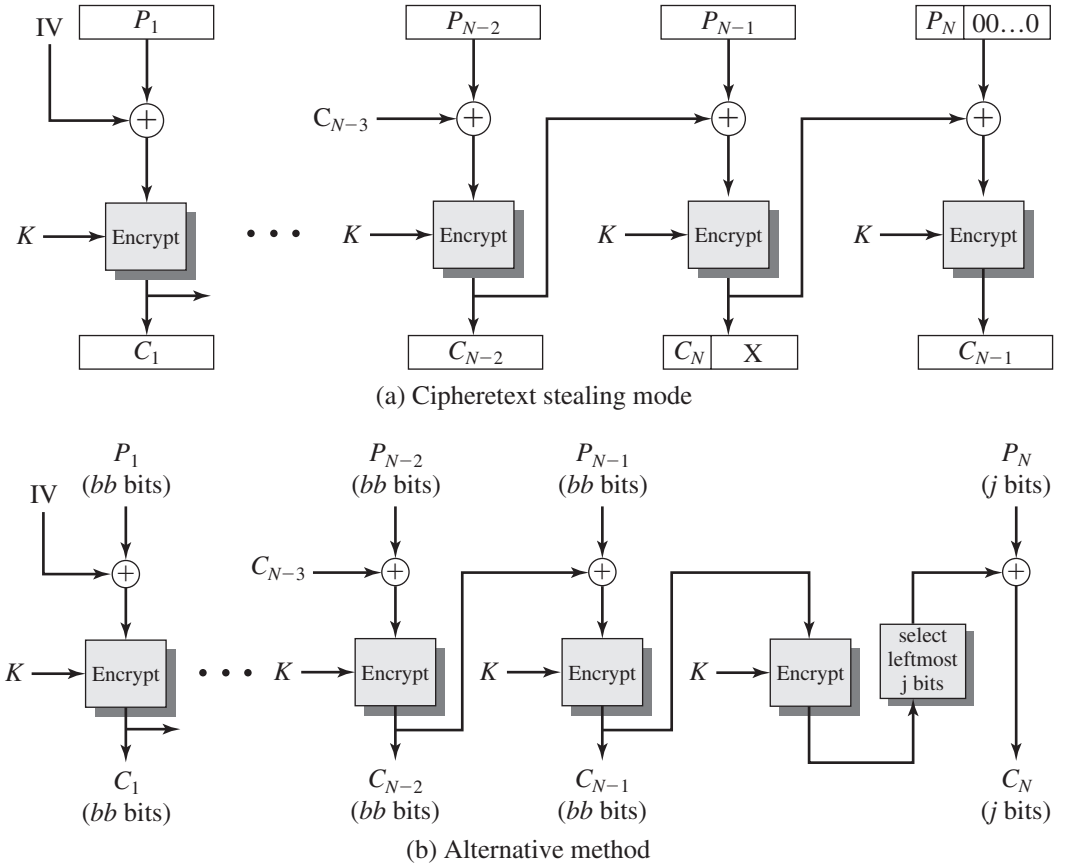


Figure 19.12 Block Cipher Modes for Plaintext Not a Multiple of Block Size

the key, XOR it with the key, and send the result over the channel. Your partner XORs the incoming block with the key (which should be the same as your key) and sends it back. You check, and if what you receive is your original random string, you have verified that your partner has the same secret key, yet neither of you has ever transmitted the key. Is there a flaw in this scheme?