3.2.2 Algorithm Modes

An *algorithm mode* is a combination of a series of the basic algorithm steps on block cipher, and some kind of feedback from the previous step. We shall discuss it now, as it forms the basis for the computer-based security algorithms. There are four important algorithm modes, namely **Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB),** and **Output Feedback (OFB)**. This is shown in Fig. 3.5. The first two modes operate on block-cipher, whereas the latter two modes are block cipher modes, which can be used as if they are working on stream cipher.

Apart from this, we will also discuss a variation of the OFB mode, called **Counter (CTR)**.

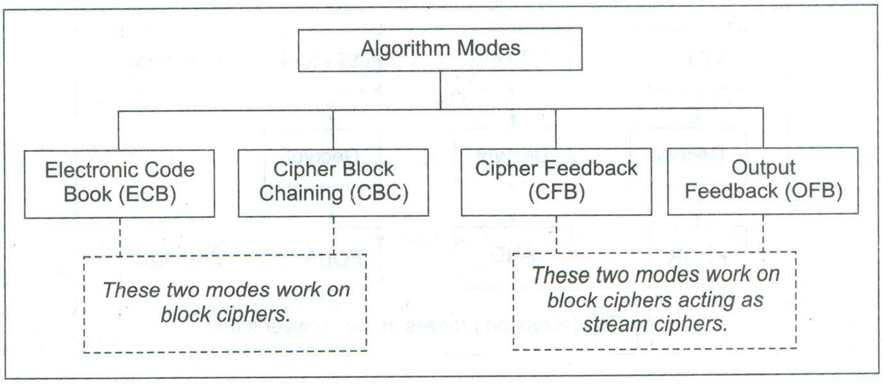
Algorithm Modes
Electronic Code Book (ECB)
Cipher Block Chaining (CBC)
Cipher Feedback (CFB)
Output Feedback (OFB)
These two modes work on block ciphers.
These two modes work on block ciphers acting as stream ciphers.

Fig. 3.5 Algorithm modes

We shall discuss the algorithm modes in brief now.

1. Electronic Code Book (ECB) Mode

Electronic Code Book (ECB) is the simplest mode of operation. Here, the incoming plain-text message is divided into blocks of 64 bits each. Each such block is then encrypted independently of the other blocks. For all blocks in a message, the same key is used for encryption. This process is shown in Fig. 3.6.

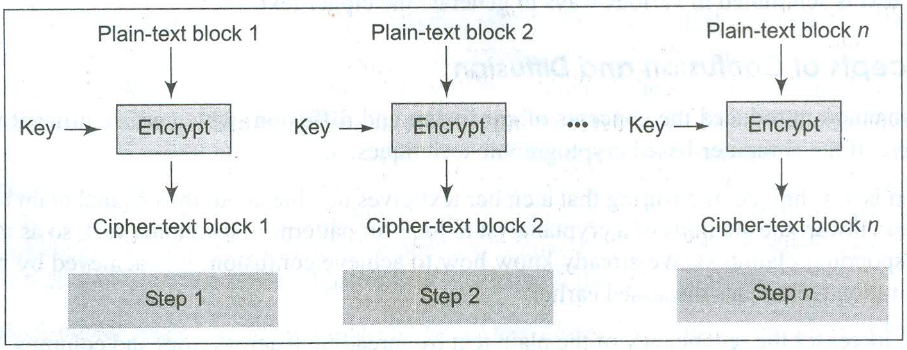

Fig. 3.6 ECB mode—the encryption process

At the receiver's end, the incoming data is divided into 64-bit blocks, and by using the same key as was used for encryption, each block is decrypted to produce the corresponding plain-text block. This process is shown in Fig. 3.7.

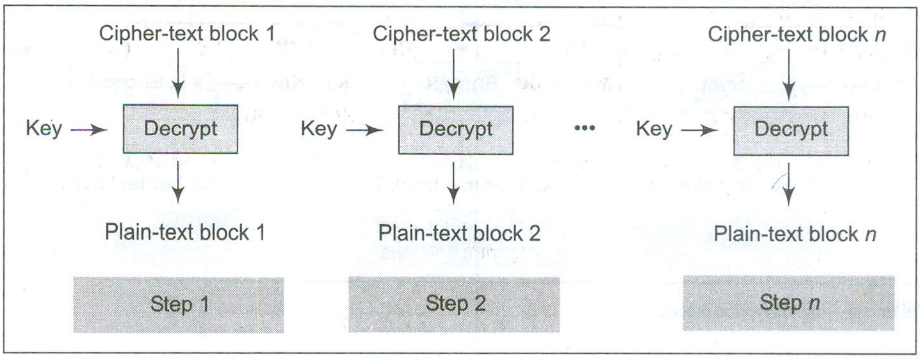

Fig. 3.7 ECB mode—the decryption process

In ECB, since a single key is used for encrypting all the blocks of a message, if a plain-text block repeats in the original message, the corresponding cipher-text block will also repeat in the encrypted message. Therefore, ECB is suitable only for encrypting small messages, where the scope for repeating the same plain-text blocks is quite less.

2. Cipher Block Chaining (CBC) Mode

We saw that in the case of ECB, within a given message (i.e. for a given key), a plain-text block always produces the same cipher-text block. Thus, if a block of plain text occurs more than once in the input, the corresponding cipher text block will also occur more than once in the output, thus providing some clues to a cryptanalyst. To overcome this problem, the **Cipher Block Chaining (CBC)** mode ensures that even if a block of plain text repeats in the input, these two (or more) identical plain-text blocks yield totally different cipher-text blocks in the output. For this, a feedback mechanism is used, as we shall learn now.

Chaining adds a feedback mechanism to a block cipher. In Cipher Block Chaining (CBC), the results of the encryption of the previous block are fed back into the encryption of the current block. That is, each block is used to modify the encryption of the next block. Thus, each block of cipher text is dependent on the corresponding current input plain-text block, as well as all the previous plain-text blocks.

The encryption process of CBC is depicted in Fig. 3.8 and described thereafter.

1. As shown in the figure, the first step receives two inputs: the first block of plain text and a random block of text, called **Initialization Vector (IV)**.
   (a) The IV has no special meaning: it is simply used to make each message unique. Since the value of IV is randomly generated, the likelihood of it repeating in two different messages is quite rare. Consequently, IV helps in making the cipher-text somewhat unique, or at least quite different from all the other cipher texts in a different message. Interestingly, it is not mandatory to keep IV secret—it can be known to everybody. This seems slightly concerning and confusing. However, if we take a re-look at the operation of CBC, we will realize that IV is simply one of the two inputs to the first encryption step. The output of step 1 is

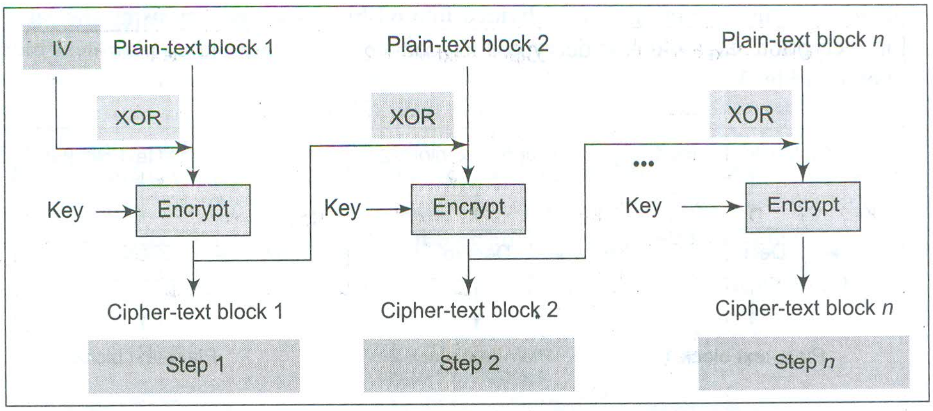

Fig. 3.8 CBC mode—the encryption process

cipher-text block 1, which is also one of the two inputs to the second encryption step. In other words, cipher-text block 1 is also an IV for step 2! Similarly, cipher-text block 2 is also an IV for step 3, and so on. Since all these cipher-text blocks will be sent to the receiver, we are anyway sending all IVs for step 2 onwards! Thus, there is no special reason why the IV for step 1 should be kept secret. It is the key used for encryption that needs to be kept secret. However, in practice, for maximum security, both the key and the IV are kept secret.

(b) The first block of cipher text and IV are combined using XOR and then encrypted using a key to produce the first cipher-text block. The first cipher-text block is then provided as a *feedback* to the next plain-text block, as explained below.

2. In the second step, the second plain-text block is XORed with the output of step 1, i.e. the first cipher-text block. It is then encrypted with the same key, as used in step 1. This produces cipher-text block 2.
3. In the third step, the third plain-text block is XORed with the output of step 2, i.e. the second cipher-text block. It is then encrypted with the same key, as used in step 1.
4. This process continues for all the remaining plain-text blocks of the original message.

Remember that the IV is used only in the first plain text block. However, the same key is used for the encryption of all plain text blocks.

The decryption process works as follows.

1. The cipher-text block 1 is passed through the decryption algorithm using the same key, which was used during the encryption process for all the plain-text blocks. The output of this step is then XORed with the IV. This process yields plain-text block 1.
2. In step 2, the cipher-text block 2 is decrypted, and its output is XORed with cipher-text block 1, which yields plain-text block 2.
3. This process continues for all the Cipher text blocks in the encrypted message.

The decryption process is shown in Fig. 3.9.

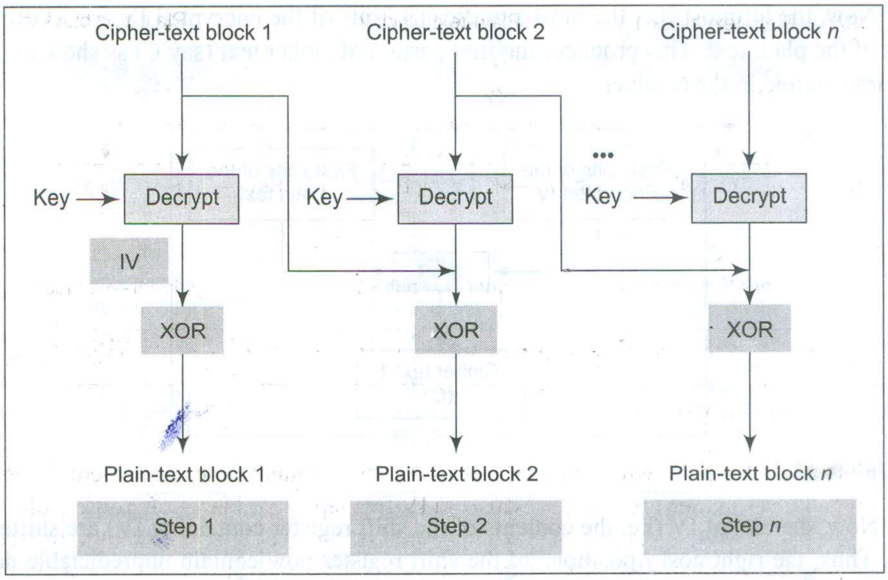

Fig. 3.9 CBC mode—the decryption process

Appendix A provides some more mathematical details about the CBC mode.

3. Cipher Feedback (CFB) Mode

Not all applications can work with blocks of data. Security is also required in applications that are character-oriented. For instance, an operator can be typing keystrokes at a terminal, which needs to be immediately transmitted across the communications link in a secure manner, i.e. by using encryption. In such situations, stream cipher must be used. The **Cipher Feedback (CFB)** mode is useful in such cases. In this mode, data is encrypted in units that are smaller (e.g. they could be of size 8 bits, i.e. the size of a character typed by an operator) than a defined block size (which is usually 64 bits).

Let us understand how CFB mode works, assuming that we are dealing with *j* bits at a time (as we have seen usually, but not always, *j* = 8). Since CFB is slightly more complicated as compared to the first two cryptography modes, we shall study CFB in a step-by-step fashion.

Step 1 Like CBC, a 64-bit Initialization Vector (IV) is used in the case of CFB mode. The IV is kept in a shift register. It is encrypted in the first step to produce a corresponding 64-bit IV cipher text. This is shown in Fig. 3.10.

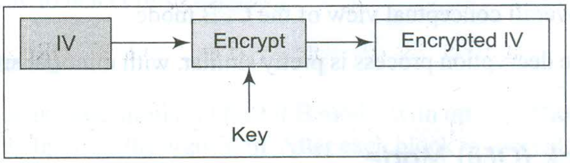

Fig. 3.10 CFB—Step 1

Step 2 Now, the leftmost (i.e. the most significant) *j* bits of the encrypted IV are XORed with the first *j* bits of the plain text. This produces the first portion of cipher text (say *C*) as shown in Fig. 3.11. *C* is then transmitted to the receiver.

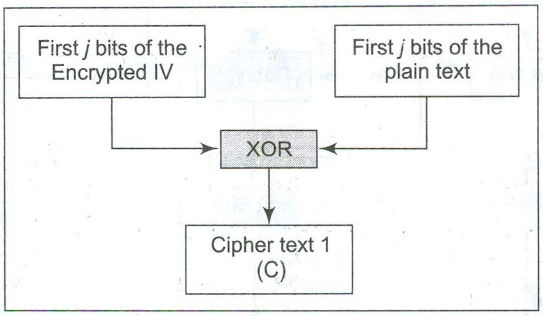
First *j* bits of the Encrypted IV
First *j* bits of the plain text
XOR
Cipher text 1 (C)

Fig. 3.11 CFB—Step 2

Step 3 Now, the bits of IV (i.e. the contents of the shift register containing IV) are shifted left by *j* positions. Thus, the rightmost *j* positions of the shift register now contain unpredictable data. These rightmost *j* positions are now filled with *C*. This is shown in Fig. 3.12.

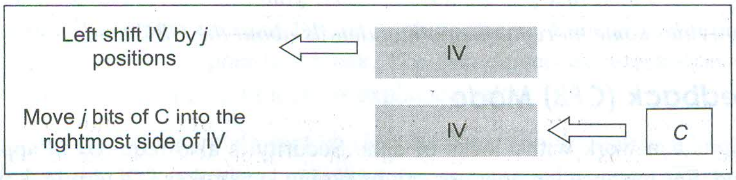
Left shift IV by *j* positions
IV
Move *j* bits of C into the rightmost side of IV
IV
C

Fig. 3.12 CFB—Step 3

Step 4 Now, steps 1 through 3 continue until all the plain-text units are encrypted. That is, the fol lowing steps are repeated:

- IV is encrypted.
- The leftmost *j* bits resulting from this encryption process are XORed with the next *j* bits of th plain text.
- The resulting cipher-text portion (i.e. the next *j* bits of cipher text) is sent to the receiver.
- The shift register containing the IV is left-shifted by *j* bits.
- The *j* bits of the cipher text are inserted from right into the shift register containing the IV.

Figure 3.13 shows the overall conceptual view of the CFB mode.

At the receiver's end, the decryption process is pretty similar, with minor changes. We shall not disc it.

4. Output Feedback (OFB) Mode

The **Output Feedback (OFB)** mode is extremely similar to the CFB. The only difference is that in case of CFB, the cipher text is fed into the next stage of encryption process. But in the case of C

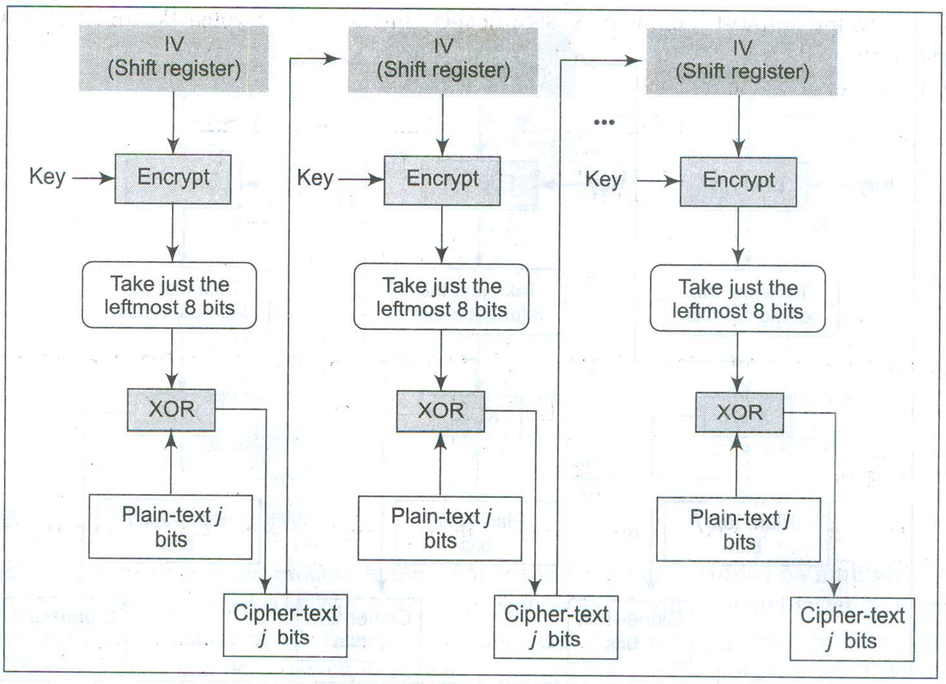

Fig. 3.13 CFB—the overall encryption process

the output of the IV encryption process is fed into the next stage of encryption process. Therefore, we shall not describe the details of OFB, and instead, shall simply draw the block diagram of the OFB process, as shown in Fig. 3.14. The same details as discussed in CFB apply here, except the change, as pointed out above.

Let us summarize the key advantage of the OFB mode. In simple terms, we can state that in this mode, if there are errors in individual bits, they remain errors in individual bits and do not corrupt the whole message. That is, bit errors do not get propagated. If a cipher-text bit C_i is in error, only the decrypted value corresponding to this bit, i.e. P_i is wrong. Other bits are not affected. Remember that in contrast to this, in the CFB mode, the cipher text bit C_i is fed back as input to the shift register, and would corrupt the other bits in the message!

The possible drawback with OFB is that an attacker can make necessary changes to the cipher text and the checksum of the message in a controlled fashion. This causes changes in the cipher text without it getting detected. In other words, the attacker changes both the cipher text and the checksum at the same time, hence there is no way to detect this change.

5. Counter (CTR) Mode

The **Counter (CTR)** mode is quite similar to the OFB mode, with one variation. It uses *sequence numbers* called *counters* as the inputs to the algorithm. After each block is encrypted, to fill the register, the next counter value is used. Usually, a constant is used as the initial counter value, and is incremented (usually by 1) for every iteration. The size of the counter block is the same as that of the plain-text block.