or equal to the least common multiple of these cycle lengths, which turned out to be around 10^{277}. But if DES is closed under composition, we showed that $m \leq 2^{56}$. Therefore, DES is not closed under composition.

4.5 Modes of Operation

DES is an example of a block cipher algorithm. A block of plaintext, 64 bits in the case of DES, is encrypted to a block of ciphertext. There are many circumstances, however, where it is necessary to encrypt messages that are either longer or shorter than the cipher's block length. For example, we may have a long text message that is many times longer than 64 bits. A plaintext that is shorter than the block size might occur in situations where data are created in a bit-by-bit, or character-by-character manner, and we are required to produce ciphertext output as quickly as we receive plaintext input.

Block ciphers can be run in many different modes of operation, allowing users to choose appropriate modes to meet the requirements of their applications. There are five common modes of operation: electronic codebook (ECB), cipher block chaining (CBC), cipher feedback (CFB), output feedback (OFB), and counter (CTR) modes. We now discuss these modes.

4.5.1 Electronic Codebook (ECB)

The natural manner for using a block cipher is to break a long piece of plaintext into appropriate sized blocks of plaintext and process each block separately with the encryption function E_K. This is known as the electronic codebook (ECB) mode of operation. The plaintext P is broken into smaller chunks $P = [P_1, P_2, \cdots, P_L]$ and the ciphertext is

$$C = [C_1, C_2, \cdots, C_L]$$

where $C_j = E_K(P_j)$ is the encryption of P_j using the key K.

There is a natural weakness in the ECB mode of operation that becomes apparent when dealing with long pieces of plaintext. Say an adversary Eve has been observing communication between Alice and Bob for a long enough period of time. If Eve has managed to acquire some plaintext pieces corresponding to the ciphertext pieces that she has observed, she can start to build up a codebook with which she can decipher future communication between Alice and Bob. Eve never needs to calculate the key K; she just looks up a ciphertext message in her codebook and uses the corresponding plaintext (if available) to decipher the message.

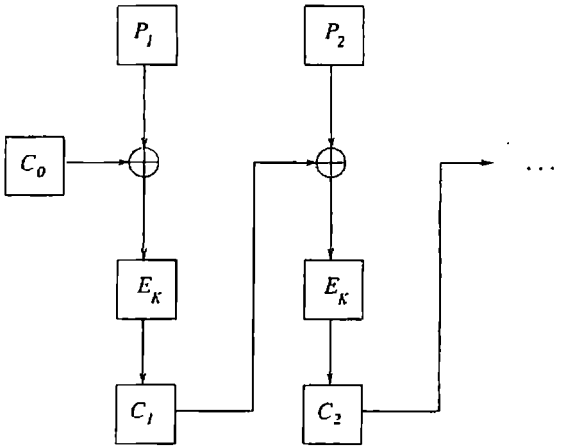

Figure 4.6: Cipher Block Chaining Mode.

This can be a serious problem since many real world messages consist of repeated fragments. E-mail is a prime example. An e-mail between Alice and Bob might start with the following header:

```
Date:  Tue, 29 Feb 2000 13:44:38 -0500 (EST)
```

The ciphertext starts with the encrypted version of "Date: Tu". If Eve finds that this piece of ciphertext often occurs on a Tuesday, she might be able to guess, without knowing any of the plaintext, that such messages are e-mail sent on Tuesdays. With patience and ingenuity, Eve might be able to piece together enough of the message's header and trailer to figure out the context of the message. With even greater patience and computer memory, she might be able to piece together important pieces of the message.

Another problem that arises in ECB mode occurs when Eve tries to modify the encrypted message being sent to Bob. She might be able to extract key portions of the message and use her codebook to construct a false ciphertext message that she can insert in the data stream.

4.5.2 Cipher Block Chaining (CBC)

One method for reducing the problems that occur in ECB mode is to use chaining. Chaining is a feedback mechanism where the encryption of a block depends on the encryption of previous blocks. In particular, encryption proceeds as

$$C_j = E_K(P_j \oplus C_{j-1}),$$

while decryption proceeds as

$$P_j = D_K(C_j) \oplus C_{j-1},$$

where C_0 is some chosen initial value. As usual, E_K and D_K denote the encryption and decryption functions for the block cipher.

Thus, in CBC mode, the plaintext is XORed with the previous ciphertext block and the result is encrypted. Figure 4.6 depicts CBC.

The initial value C_0 is often called the initialization vector, or the IV. If we use a fixed value for C_0, say $C_0 = 0$, and ever have the same plaintext message, the result will be that the resulting ciphertexts will be the same. This is undesirable since it allows the adversary to deduce that the same plaintext was created. This can be very valuable information, and can often be used by the adversary to infer the meaning of the original plaintext.

In practice, this problem is handled by always choosing the IV C_0 randomly and sending C_0 in the clear along with the first ciphertext C_1. By doing so, even if the same plaintext message is sent repeatedly, an observer will see a different ciphertext each time.

4.5.3 Cipher Feedback (CFB)

One of the problems with both the CBC and ECB methods is that encryption (and hence decryption) cannot begin until a complete block of 64 bits of plaintext data is available. The cipher feedback mode operates in a manner that is very similar to the way in which LFSR is used to encrypt plaintext. Rather than use linear recurrence to generate random bits, the cipher feedback mode is a stream mode of operation that produces pseudorandom bits using the block cipher E_K. In general, CFB operates in a k-bit mode, where each application produces k random bits for XORing with the plaintext. For our discussion, however, we focus on the 8-bit version of CFB. Using the 8-bit CFB allows one 8-bit piece of message (e.g., a single character) to be encrypted without having to wait for an entire block of data to be available. This is useful in interactive computer communications, for example.

The plaintext is broken into 8-bit pieces: $P = [P_1, P_2, \ldots]$, where each P_j has 8 bits, rather than the 64 bits used in ECB and CBC. Encryption proceeds as follows. An initial 64-bit X_1 is chosen. Then for $j = 1, 2, 3, \ldots$, the following is performed:

$$\begin{aligned} O_j &= L_8(E_K(X_j)) \\ C_j &= P_j \oplus O_j \\ X_{j+1} &= R_{56}(X_j) \,\|\, C_j, \end{aligned}$$

where $L_8(X)$ denotes the 8 leftmost bits of X, $R_{56}(X)$ denotes the rightmost 56 bits of X, and $X\|Y$ denotes the string obtained by writing X followed by Y. We present the CFB mode of operation in Figure 4.7.

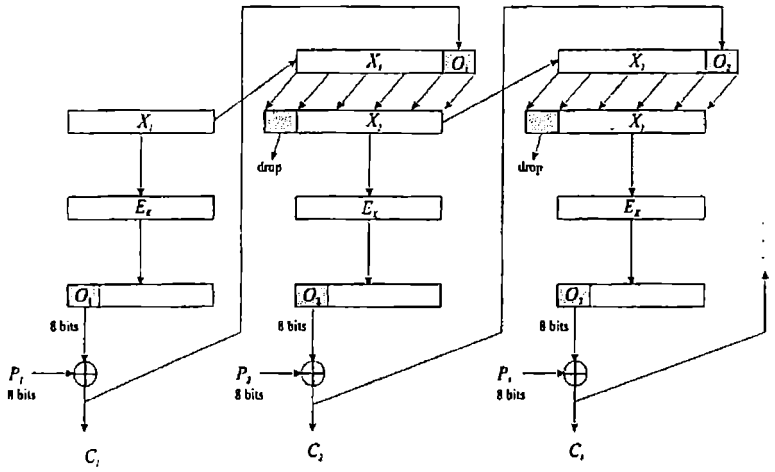

Figure 4.7: Cipher Feedback Mode.

Decryption is done with the following steps:

$$P_j = C_j \oplus L_8(E_K(X_j))$$

$$X_{j+1} = R_{56}(X_j) \parallel C_j.$$

We note that decryption does not involve calling the decryption function, D_K. This would be an advantage of running a block cipher in a stream mode in a case where the decryption function for the block cipher is slower than the encryption function.

Let's step through one round of the CFB algorithm. First, we have a 64-bit register that is initialized with X_1. These 64 bits are encrypted using E_K and the leftmost 8 bits of $E_K(X_1)$ are extracted and XORed with the 8-bit P_1 to form C_1. Then C_1 is sent to the recipient. Before working with P_2, the 64-bit register X_1 is updated by extracting the rightmost 56 bits. The 8 bits of C_1 are appended on the right to form $X_2 = R_{56}(X_1)\|C_1$. Then P_2 is encrypted by the same process, but using X_2 in place of X_1. After P_2 is encrypted to C_2, the 64-bit register is updated to form

$$X_3 = L_{56}(X_2)\|C_2 = L_{48}(X_1)\|C_1\|C_2.$$

By the end of the 8th round, the initial X_1 has disappeared from the 64-bit register and $X_9 = C_1\|C_2\|\cdots\|C_8$. The C_j continue to pass through the register, so for example $X_{20} = C_{12}\|C_{13}\|\cdots\|C_{19}$.

Note that CFB encrypts the plaintext in a manner similar to one-time pads or LFSRs. The key K and the numbers X_j are used to produce binary strings that are XORed with the plaintext to produce the ciphertext. This is a much different type of encryption than the ECB and CBC, where the ciphertext is the output of DES.

In practical applications, CFB is useful because it can recover from errors in transmission of the ciphertext. Suppose that the transmitter sends the ciphertext blocks $C_1, C_2, \ldots, C_k, \ldots$, and C_1 is corrupted during transmission, so that the receiver observes $\tilde{C}_1, C_2, \ldots$. Decryption takes $\tilde{C}_1$ and produces a garbled version of P_1 with bit errors in the locations that $\tilde{C}_1$ had bit errors. Now, after decrypting this ciphertext block, the receiver forms an incorrect X_2, which we denote $\tilde{X}_2$. If X_1 was $(*,*,*,*,*,*,*,*)$, then $\tilde{X}_2 = (*,*,*,*,*,*,*,*,\tilde{C}_1)$. When the receiver gets an uncorrupted C_2 and decrypts, then a completely garbled version of P_2 is produced. When forming X_3, the decrypter actually forms $\tilde{X}_3 = (*,*,*,*,*,*,*,\tilde{C}_1, C_2)$. The decrypter repeats this process, ultimately getting bad versions of $P_1, P_2, \cdots, P_9$. When the decrypter calculates X_9, the error block has moved to the leftmost block of $\tilde{X}_9$ as $\tilde{X}_9 = (\tilde{C}_1, C_2, \cdots, C_8)$. At the next step, the error will have been flushed from the X_{10} register, and X_{10} and subsequent registers will be uncorrupted. For a simplified version of these ideas, see Exercise 9.

4.5.4 Output Feedback (OFB)

The CBC and CFB modes of operation exhibit a drawback in that errors propagate for a duration of time corresponding to the block size of the cipher. The output feedback mode (OFB) is another example of a stream mode of operation for a block cipher where encryption is performed by XORing the message with a pseudo-random bit stream generated by the block cipher. One advantage of the OFB mode is that it avoids error propagation.

Much like CFB, OFB may work on chunks of different sizes. For our discussion, we will focus on the 8-bit version of OFB, where OFB is used to encrypt 8-bit chunks of plaintext in a streaming mode. Just as for CFB, we break our plaintext into 8-bit pieces, with $P = [P_1, P_2, \cdots,]$. We start with an initial value X_1, which has a length equal to the block length of the cipher, for example, 64 bits. X_1 is often called the IV, and should be chosen to be random. X_1 is encrypted using the key K to produce 64 bits of output, and the leftmost 8 bits O_1 of the ciphertext are extracted. These are then XORed with the first 8 bits P_1 of the plaintext to produce 8 bits of ciphertext, C_1.

So far, this is the same as what we were doing in CFB. But OFB differs from CFB in what happens next. In order to iterate, CFB updates the register X_2 by extracting the right 56 bits of X_1 and appending C_1 to the right side. Rather than use the ciphertext, OFB uses the output of the