7.2 A Simplified DES-Type Algorithm

The DES algorithm is rather unwieldy to use for examples, so in the present section we present an algorithm that has many of the same features, but is much smaller. Like DES, the present algorithm is a block cipher. Since the blocks are encrypted separately, we assume throughout the present discussion that the full message consists of only one block.

The message has 12 bits and is written in the form L_0R_0, where L_0 consists of the first six bits and R_0 consists of the last six bits. The key K has nine bits. The ith round of the algorithm transforms an input $L_{i-1}R_{i-1}$ to the output L_iR_i using an eight-bit key K_i derived from K.

The main part of the encryption process is a function $f(R_{i-1}, K_i)$ that takes a six-bit input R_{i-1} and an eight-bit input K_i and produces a six-bit output. This will be described later.

The output for the ith round is defined as follows:

$$L_i = R_{i-1} \text{ and } R_i = L_{i-1} \oplus f(R_{i-1}, K_i),$$

where $\oplus$ denotes XOR, namely bit-by-bit addition mod 2. This is depicted in Figure 7.1.

Figure 7.1 One Round of a Feistel System

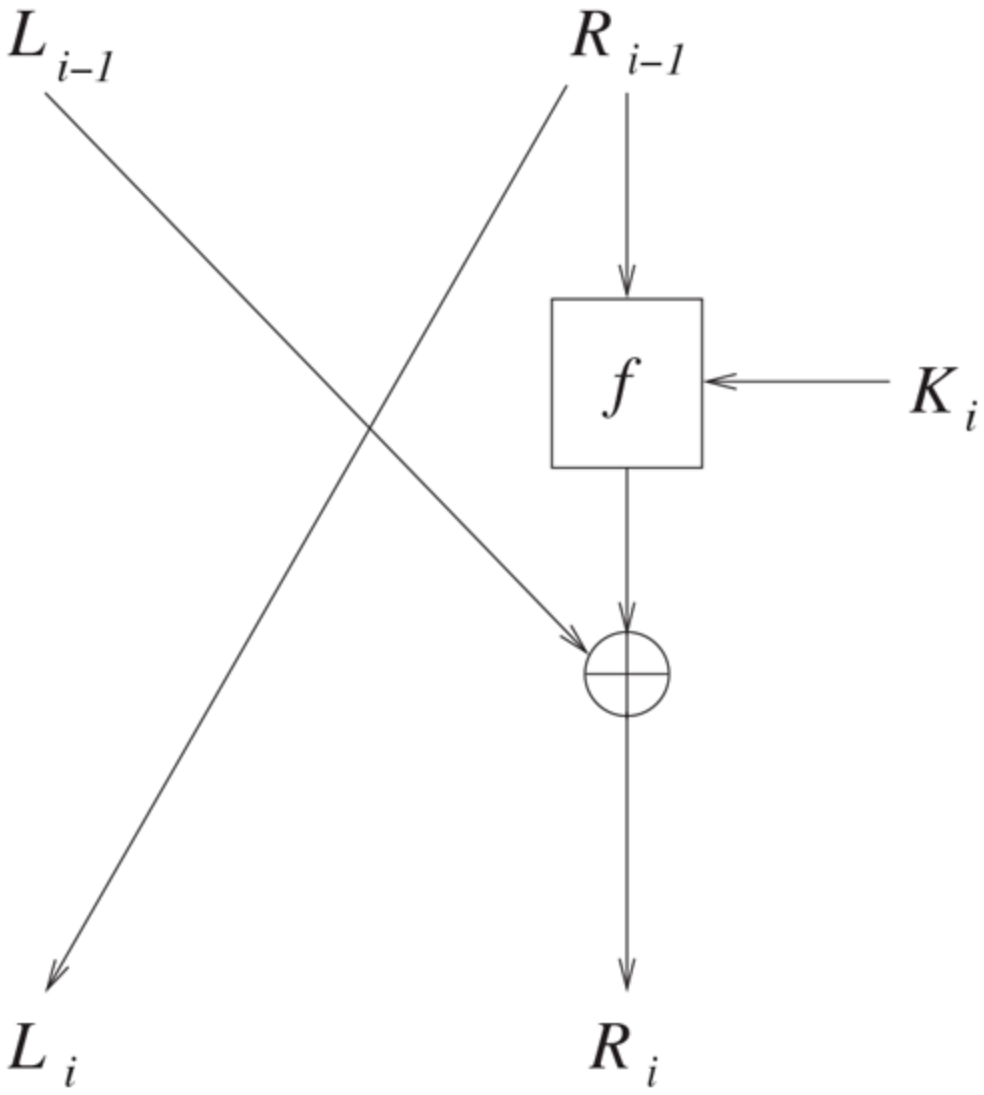

Figure 7.1 Full Alternative Text

This operation is performed for a certain number of rounds, say n, and produces the ciphertext L_nR_n.

How do we decrypt? Start with L_nR_n and switch left and right to obtain R_nL_n. (Note: *This switch is built into the DES*

encryption algorithm, so it is not needed when decrypting DES.) Now use the same procedure as before, but with the keys K_i used in reverse order $K_n,\ \ldots,\ K_1$. Let's see how this works. The first step takes R_nL_n and gives the output

$$[L_n] \quad [R_n \oplus f(L_n, K_n)].$$

We know from the encryption procedure that $L_n = R_{n-1}$ and $R_n = L_{n-1} \oplus f(R_{n-1}, K_n)$. Therefore,

$$\begin{aligned}[L_n] \quad [R_n \oplus f(L_n, K_n)] = [R_{n-1}] \quad & [L_{n-1} \oplus f(R_{n-1}, K_n) \oplus f(L_n, K_n)] \\ & = [R_{n-1}] \quad [L_{n-1}].\end{aligned}$$

The last equality again uses $L_n = R_{n-1}$, so that $f(R_{n-1}, K_n) \oplus f(L_n, K_n)$ is 0. Similarly, the second step of decryption sends $R_{n-1}L_{n-1}$ to $R_{n-2}L_{n-2}$. Continuing, we see that the decryption process leads us back to R_0L_0. Switching the left and right halves, we obtain the original plaintext L_0R_0, as desired.

Note that the decryption process is essentially the same as the encryption process. We simply need to switch left and right and use the keys K_i in reverse order. Therefore, both the sender and receiver use a common key and they can use identical machines (though the receiver needs to reverse left and right inputs).

So far, we have said nothing about the function f. In fact, any f would work in the above procedures. But some choices of f yield much better security than others. The type of f used in DES is similar to that which we describe next. It is built up from a few components.

The first function is an expander. It takes an input of six bits and outputs eight bits. The one we use is given in Figure 7.2.

Figure 7.2 The Expander Function

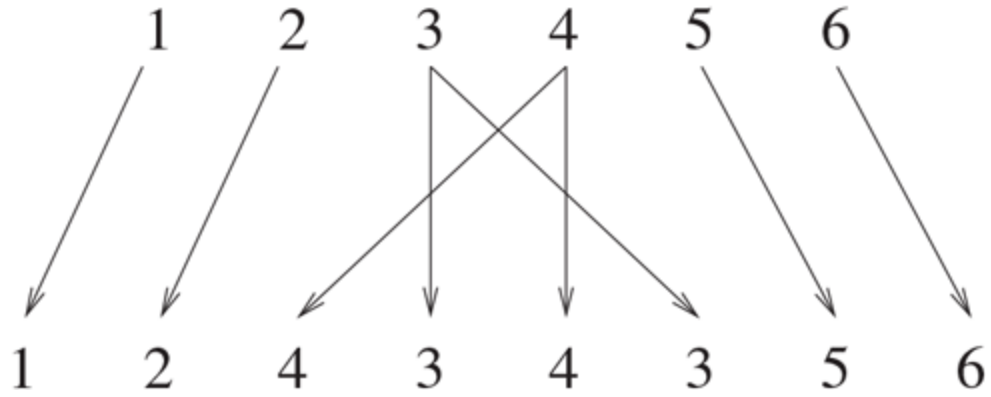

Figure 7.2 Full Alternative Text

This means that the first input bit yields the first output bit, the third input bit yields both the fourth and the sixth output bits, etc. For example, 011001 is expanded to 01010101.

The main components are called S-boxes. We use two:

$$S_1 \quad \begin{bmatrix} 101 & 010 & 001 & 110 & 011 & 100 & 111 & 000 \\ 001 & 100 & 110 & 010 & 000 & 111 & 101 & 011 \end{bmatrix}$$

$$S_2 \quad \begin{bmatrix} 100 & 000 & 110 & 101 & 111 & 001 & 011 & 010 \\ 101 & 011 & 000 & 111 & 110 & 010 & 001 & 100 \end{bmatrix}.$$

The input for an S-box has four bits. The first bit specifies which row will be used: 0 for the first row, 1 for the second. The other three bits represent a binary number that specifies

the column: 000 for the first column, 001 for the second, ..., 111 for the last column. The output for the S-box consists of the three bits in the specified location. For example, an input of 1010 for S_1 means we look at the second row, third column, which yields the output 110.

The key K consists of nine bits. The key K_i for the ith round of encryption is obtained by using eight bits of K, starting with the ith bit. For example, if $K = 010011001$, then $K_4 = 01100101$ (after five bits, we reached the end of K, so the last two bits were obtained from the beginning of K).

We can now describe $f(R_{i-1}, K_i)$. The input R_{i-1} consists of six bits. The expander function is used to expand it to eight bits. The result is XORed with K_i to produce another eight-bit number. The first four bits are sent to S_1, and the last four bits are sent to S_2. Each S-box outputs three bits, which are concatenated to form a six-bit number. This is $f(R_{i-1}, K_i)$. We present this in Figure 7.3.

Figure 7.3 The Function $f(R_{i-1}, K_i)$

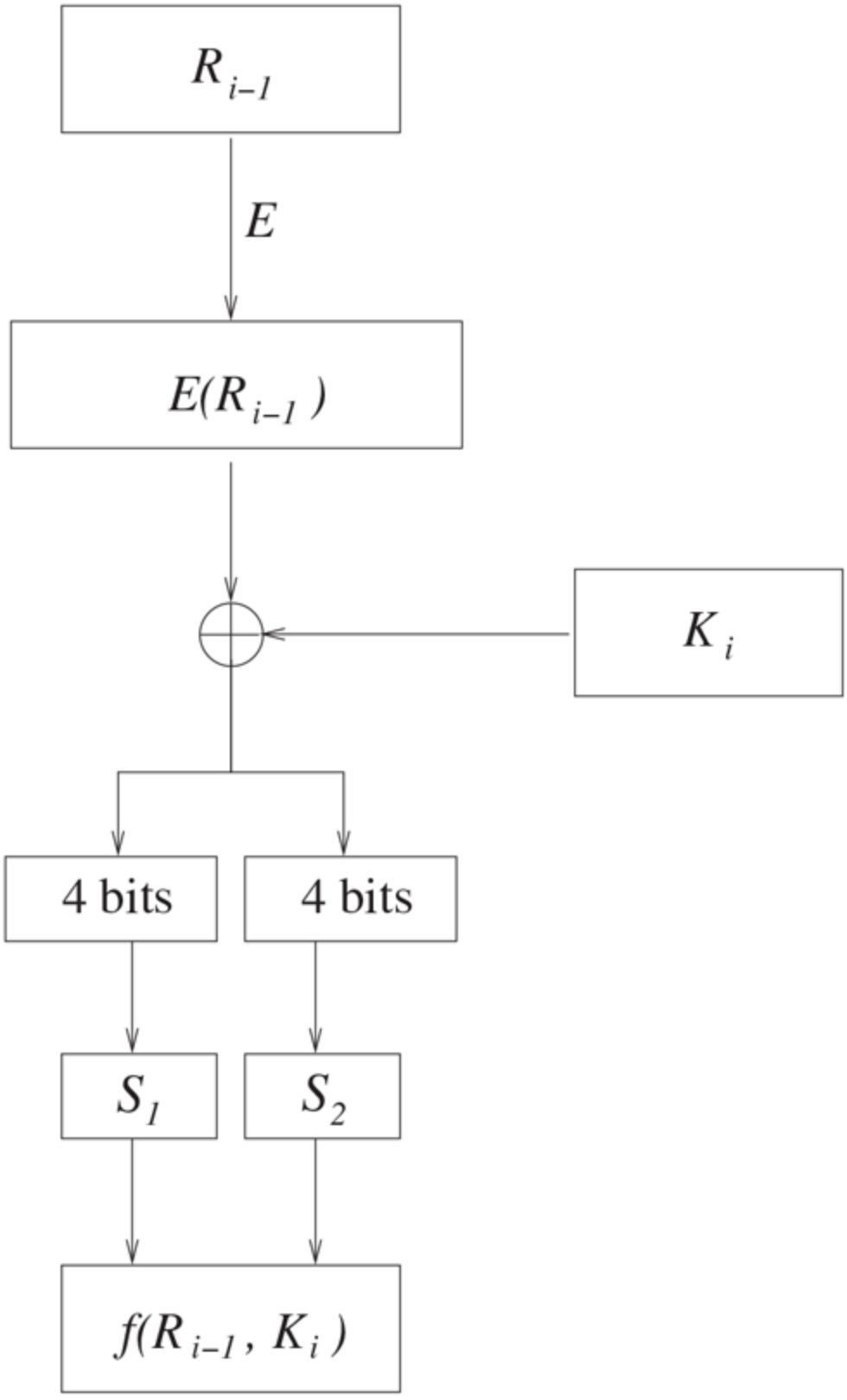

Figure 7.3 Full Alternative Text

For example, suppose $R_{i-1} = 100110$ and $K_i = 01100101$. We have

$$E(100110) \oplus K_i = 10101010 \oplus 01100101 = 11001111.$$

The first four bits are sent to S_1 and the last four bits are sent to S_2. The second row, fifth column of S_1 contains 000. The second row, last column of S_2 contains 100. Putting these outputs one after the other yields $f(R_{i-1}, K_i) = 000100$.

We can now describe what happens in one round. Suppose the input is

$$L_{i-1}R_{i-1} = 011100100110$$

and $K_i = 01100101$, as previously. This means that $R_{i-1} = 100110$, as in the example just discussed. Therefore, $f(R_{i-1}, K_i) = 000100$. This is XORed with $L_{i-1} = 011100$ to yield $R_i = 011000$. Since $L_i = R_{i-1}$, we obtain

$$L_iR_i = 100110011000.$$

The output becomes the input for the next round.

For work on this and another simplified DES algorithm and how they behave under multiple encryption, see [Konikoff-Toplosky].

7.3 Differential Cryptanalysis

This section is rather technical and can be skipped on a first reading.

Differential cryptanalysis was introduced by Biham and Shamir around 1990, though it was probably known much earlier to the designers of DES at IBM and NSA. The idea is to compare the differences in the ciphertexts for suitably chosen pairs of plaintexts and thereby deduce information about the key. Note that the difference of two strings of bits can be found by XORing them. Because the key is introduced by XORing with $E(R_{i-1})$, looking at the XOR of the inputs removes the effect of the key at this stage and hence removes some of the randomness introduced by the key. We'll see that this allows us to deduce information as to what the key could be.

7.3.1 Differential Cryptanalysis for Three Rounds

We eventually want to describe how to attack the above system when it uses four rounds, but we need to start by analyzing three rounds. Therefore, we temporarily start with L_1R_1 instead of L_0R_0.

The situation is now as follows. We have obtained access to a three-round encryption device that uses the preceding procedure. We know all the inner workings of the encryption algorithm such as the S-boxes, but we do not know the key. We want to find the key by a chosen plaintext attack. We use various inputs L_1R_1 and obtain outputs L_4R_4.

We have

$$\begin{gathered} R_2 = L_1 \oplus f(R_1, K_2) \\ L_3 = R_2 = L_1 \oplus f(R_1, K_2) \\ R_4 = L_3 \oplus f(R_3, K_4) = L_1 \oplus f(R_1, K_2) \oplus f(R_3, K_4). \end{gathered}$$

Suppose we have another message $L_1^*R_1^*$ with $R_1 = R_1^*$. For each i, let $R_i{}' = R_i \oplus R_i^*$ and $L_i{}' = L_i \oplus L_i^*$. Then $L_i{}'R_i{}'$ is the "difference" (or sum; we are working mod 2) of L_iR_i and $L_i^*R_i^*$. The preceding calculation applied to $L_1^*R_1^*$ yields a formula for R_4^*. Since we have assumed that $R_1 = R_1^*$, we have $f(R_1, K_2) = f(R_1^*, K_2)$. Therefore, $f(R_1, K_2) \oplus f(R_1^*, K_2) = 0$ and

$$R_4^{'} = R_4 \oplus R_4^* = L_1^{'} \oplus f(R_3, K_4) \oplus f(R_3^*, K_4).$$

This may be rearranged to

$$R_4^{'} \oplus L_1^{'} = f(R_3, K_4) \oplus f(R_3^*, K_4).$$

Finally, since $R_3 = L_4$ and $R_3^* = L_4^*$, we obtain

$$R_4' \oplus L_1' = f(L_4, K_4) \oplus f(L_4^*, K_4).$$

Note that if we know the input XOR, namely $L_1{}'R_1{}'$, and if we know the outputs L_4R_4 and $L_4^*R_4^*$, then we know everything in this last equation except K_4.

Now let's analyze the inputs to the S-boxes used to calculate $f(L_4, K_4)$ and $f(L_4^*, K_4)$. If we start with L_4, we first expand and then XOR with K_4 to obtain $E(L_4) \oplus K_4$, which are the bits sent to S_1 and S_2. Similarly, L_4^* yields $E(L_4^*) \oplus K_4$. The XOR of these is

$$E(L_4) \oplus E(L_4^*) = E(L_4 \oplus L_4^*) = E(L_4')$$

(the first equality follows easily from the bit-by-bit description of the expansion function). Therefore, we know

1. the XORs of the inputs to the two S-boxes (namely, the first four and the last four bits of $E(L_4')$);
2. the XORs of the two outputs (namely, the first three and the last three bits of $R_4' \oplus L_1'$).

Let's restrict our attention to S_1. The analysis for S_2 will be similar. It is fairly fast to run through all pairs of four-bit inputs with a given XOR (there are only 16 of them) and see which ones give a desired output XOR. These can be computed once for all and stored in a table.

For example, suppose we have input XOR equal to 1011 and we are looking for output XOR equal to 100. We can run through the input pairs (1011, 0000), (1010, 0001), (1001, 0010), ..., each of which has XOR equal to 1011, and look at

the output XORs. We find that the pairs (1010, 0001) and (0001, 1010) both produce output XORs 100. For example, 1010 means we look at the second row, third column of S_1, which is 110. Moreover, 0001 means we look at the first row, second column, which is 010. The output XOR is therefore $110 \oplus 010 = 100$.

We know L_4 and L_4^*. For example, suppose $L_4 = 101110$ and $L_4^* = 000010$. Therefore, $E(L_4) = 10111110$ and $E(L_4^*) = 00000010$, so the inputs to S_1 are $1011 \oplus K_4^L$ and $0000 \oplus K_4^L$, where K_4^L denotes the left four bits of K_4. If we know that the output XOR for S_1 is 100, then $(1011 \oplus K_4^L,\ 0000 \oplus K_4^L)$ must be one of the pairs on the list we just calculated, namely (1010, 0001) and (0001, 1010). This means that $K_4^L = 0001$ or 1010.

If we repeat this procedure a few more times, we should be able to eliminate one of the two choices for K_4 and hence determine four bits of K. Similarly, using S_2, we find four more bits of K. We therefore know eight of the nine bits of K. The last bit can be found by trying both possibilities and seeing which one produces the same encryptions as the machine we are attacking.

Here is a summary of the procedure (for notational convenience, we describe it with both S-boxes used simultaneously, though in the examples we work with the S-boxes separately):

1. Look at the list of pairs with input XOR $= E(L_4')$ and output XOR $= R_4' \oplus L_1'$.
2. The pair $\big(E(L_4) \oplus K_4,\ E(L_4^*) \oplus K_4\big)$ is on this list.
3. Deduce the possibilities for K_4.
4. Repeat until only one possibility for K_4 remains.

Example

We start with

$$L_1R_1 = 000111011011$$

and the machine encrypts in three rounds using the key $K = 001001101$, though we do not yet know K. We obtain (note that since we are starting with L_1R_1, we start with the shifted key $K_2 = 01001101$)

$$L_4R_4 = 000011100101.$$

If we start with

$$L_1^*R_1^* = 101110011011$$

(note that $R_1 = R_1^*$), then

$$L_4^*R_4^* = 100100011000.$$

We have $E(L_4) = 00000011$ and $E(L_4^*) = 10101000$. The inputs to S_1 have XOR equal to 1010 and the inputs to S_2 have XOR equal to 1011. The S-boxes have output XOR $R_4' \oplus L_1'1 = 111101 \oplus 101001 = 010100$, so the output XOR from S_1 is 010 and that from S_2 is 100.

For the pairs $(1001, 0011)$, $(0011, 1001)$, S_1 produces output XOR equal to 010. Since the first member of one of these pairs should be the left four bits of $E(L_4) \oplus K_4 = 0000 \oplus K_4$, the first four bits of K_4 are in $\{1001, 0011\}$. For the pairs $(1100, 0111)$, $(0111, 1100)$, S_2 produces output XOR equal to 100. Since the first member of one of these pairs should be the right four bits of $E(L_4) \oplus K_4 = 0011 \oplus K_4$, the last four bits of K_4 are in $\{1111, 0100\}$.

Now repeat (with the same machine and the same key K) and with

$$L_1R_1 = 010111011011 \text{ and } L_1^*R_1^* = 101110011011.$$

A similar analysis shows that the first four bits of K_4 are in $\{0011, 1000\}$ and the last four bits are in $\{0100, 1011\}$. Combining this with the previous information, we see that the first four bits of K_4 are 0011 and the last four bits are 0100. Therefore, $K = 00 * 001101$ (recall that K_4 starts with the fourth bit of K.

It remains to find the third bit of K. If we use $K = 000001101$, it encrypts L_1R_1 to 001011101010, which is not L_4R_4, while $K = 001001101$ yields the correct encryption. Therefore, the key is $K = 001001101$.

7.3.2 Differential Cryptanalysis for Four

Rounds

Suppose now that we have obtained access to a four-round device. Again, we know all the inner workings of the algorithm except the key, and we want to determine the key. The analysis we used for three rounds still applies, but to extend it to four rounds we need to use more probabilistic techniques.

There is a weakness in the box S_1. If we look at the 16 input pairs with XOR equal to 0011, we discover that 12 of them have output XOR equal to 011. Of course, we expect on the average that two pairs should yield a given output XOR, so the present case is rather extreme. A little variation is to be expected; we'll see that this large variation makes it easy to find the key.

There is a similar weakness in S_2, though not quite as extreme. Among the 16 input pairs with XOR equal to 1100, there are eight with output XOR equal to 010.

Suppose now that we start with randomly chosen R_0 and R_0^* such that $R_0' = R_0 \oplus R_0^* = 001100$. This is expanded to $E(001100) = 00111100$. Therefore the input XOR for S_1 is 0011 and the input XOR for S_2 is 1100. With probability 12/16 the output XOR for S_1 will be 011, and with probability 8/16 the output XOR for S_2 will be 010. If we assume the outputs of the two S-boxes are independent, we see that the combined output XOR will be 011010 with probability (12/16)(8/16) = 3/8. Because the expansion function sends bits 3 and 4 to both

S_1 and S_2, the two boxes cannot be assumed to have independent outputs, but 3/8 should still be a reasonable estimate for what happens.

Now suppose we choose L_0 and L_0^* so that $L_0{}' = L_0 \oplus L_0^* = 011010$. Recall that in the encryption algorithm the output of the S-boxes is XORed with L_0 to obtain R_1. Suppose the output XOR of the S-boxes is 011010. Then $R_1{}' = 011010 \oplus L_0{}' = 000000$. Since $R_1{}' = R_1 \oplus R_1^*$, it follows that $R_1 = R_1^*$.

Putting everything together, we see that if we start with two randomly chosen messages with XOR equal to $L_0^{'}R_0^{'} = 011010001100$, then there is a probability of around 3/8 that $L_1{}'R_1{}' = 001100000000$.

Here's the strategy for finding the key. Try several randomly chosen pairs of inputs with XOR equal to 011010001100. Look at the outputs L_4R_4 and $L_4^*R_4^*$. Assume that $L_1^{'}R_1^{'} = 001100000000$. Then use three-round differential cryptanalysis with $L_1^{'} = 001100$ and the known outputs to deduce a set of possible keys K_4. When $L_1^{'}R_1^{'} = 001100000000$, which should happen around 3/8 of the time, this list of keys will contain K_4, along with some other random keys. The remaining 5/8 of the time, the list should contain random keys. Since there seems to be no reason that any incorrect key should appear frequently, the correct key K_4 will probably appear in the lists of keys more often than the other keys.

Here is an example. Suppose we are attacking a four-round device. We try one hundred random pairs of inputs L_0R_0 and $L_0^*R_0^* = L_0R_0 \oplus 011010001100$. The frequencies of possible keys we obtain are in the following table. We find it easier to look at the first four bits and the last four bits of K_4 separately.

First four bits	Frequency	First four bits	Frequency
0000	12	1000	33
0001	7	1001	40
0010	8	1010	35
0011	15	1011	35
0100	4	1100	59
0101	3	1101	32
0110	4	1110	28
0111	6	1111	39

Last four bits	Frequency	Last four bits	Frequency
0000	14	1000	8
0001	6	1001	16
0010	42	1010	8
0011	10	1011	18
0100	27	1100	8
0101	10	1101	23
0110	8	1110	6
0111	11	1111	17

7.3-1 Full Alternative Text

It is therefore likely that $K_4 = 11000010$. Therefore, the key K is 10*110000.

To determine the remaining bit, we proceed as before. We can compute that 000000000000 is encrypted to 100011001011 using $K = 101110000$ and is encrypted to 001011011010 using $K = 100110000$. If the machine we are attacking encrypts 000000000000 to 100011001011, we conclude that

the second key cannot be correct, so the correct key is probably $K = 101110000$.

The preceding attack can be extended to more rounds by extensions of these methods. It might be noticed that we could have obtained the key at least as quickly by simply running through all possibilities for the key. That is certainly true in this simple model. However, in more elaborate systems such as DES, differential cryptanalytic techniques are much more efficient than exhaustive searching through all keys, at least until the number of rounds becomes fairly large. In particular, the reason that DES uses 16 rounds appears to be because differential cryptanalysis is more efficient than exhaustive search until 16 rounds are used.

There is another attack on DES, called **linear cryptanalysis**, that was developed by Mitsuru Matsui [Matsui]. The main ingredient is an approximation of DES by a linear function of the input bits. It is theoretically faster than an exhaustive search for the key and requires around 2^{43} plaintext–ciphertext pairs to find the key. It seems that the designers of DES had not anticipated linear cryptanalysis. For details of the method, see [Matsui].

7.4 DES

A block of plaintext consists of 64 bits. The key has 56 bits, but is expressed as a 64-bit string. The 8th, 16th, 24th, ..., bits are parity bits, arranged so that each block of eight bits has an odd number of 1s. This is for error detection purposes. The output of the encryption is a 64-bit ciphertext.

The DES algorithm, depicted in Figure 7.4, starts with a plaintext m of 64 bits, and consists of three stages:

1. The bits of m are permuted by a fixed initial permutation to obtain $m_0 = IP(m)$. Write $m_0 = L_0R_0$, where L_0 is the first 32 bits of m_0 and R_0 is the last 32 bits.
2. For $1 \leq i \leq 16$, perform the following:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i),$$

   where K_i is a string of 48 bits obtained from the key K and f is a function to be described later.
3. Switch left and right to obtain $R_{16}L_{16}$, then apply the inverse of the initial permutation to get the ciphertext $c = IP^{-1}(R_{16}L_{16})$.

Figure 7.4 The DES Algorithm