Chapter 3
The Data Encryption Standard (DES) and Alternatives

The *Data Encryption Standard (DES)* has been by far the most popular block cipher for most of the last 30 years. Even though it is nowadays not considered secure against a determined attacker because the DES key space is too small, it is still used in legacy applications. Furthermore, encrypting data three times in a row with DES — a process referred to as *3DES* or *triple DES* — yields a very secure cipher which is still widely used today (Section 3.5 deals with 3DES.) Perhaps what is more important, since DES is by far the best-studied symmetric algorithm, its design principles have inspired many current ciphers. Hence, studying DES helps us to understand many other symmetric algorithms.

In this chapter you will learn:

- The design process of DES, which is very helpful for understanding the technical and political evolution of modern cryptography
- Basic design ideas of block ciphers, including confusion and diffusion, which are important properties of all modern block ciphers
- The internal structure of DES, including Feistel networks, S-boxes and the key schedule
- Security analysis of DES
- Alternatives to DES, including 3DES

Christof Paar et al., *Understanding Cryptography*

3.1 Introduction to DES

In 1972 a mildly revolutionary act was performed by the US National Bureau of Standards (NBS), which is now called *National Institute of Standards and Technology (NIST)*: the NBS initiated a request for proposals for a standardized cipher in the USA. The idea was to find a single secure cryptographic algorithm which could be used for a variety of applications. Up to this point in time governments had always considered cryptography, and in particular cryptanalysis, so crucial for national security that it had to be kept secret. However, by the early 1970s the demand for encryption for commercial applications such as banking had become so pressing that it could not be ignored without economic consequences.

The NBS received the most promising candidate in 1974 from a team of cryptographers working at IBM. The algorithm IBM submitted was based on the cipher *Lucifer*. Lucifer was a family of ciphers developed by Horst Feistel in the late 1960s, and was one of the first instances of block ciphers operating on digital data. Lucifer is a Feistel cipher which encrypts blocks of 64 bits using a key size of 128 bits. In order to investigate the security of the submitted ciphers, the NBS requested the help of the *National Security Agency (NSA)*, which did not even admit its existence at that point in time. It seems certain that the NSA influenced changes to the cipher, which was rechristened DES. One of the changes that occurred was that DES is specifically designed to withstand differential cryptanalysis, an attack not known to the public until 1990. It is not clear whether the IBM team developed the knowledge about differential cryptanalysis by themselves or whether they were guided by the NSA. Allegedly, the NSA also convinced IBM to reduce the Lucifer key length of 128 bit to 56 bit, which made the cipher much more vulnerable to brute-force attacks.

The NSA involvement worried some people because it was feared that a secret trapdoor, i.e., a mathematical property with which DES could be broken but which is only known to NSA, might have been the real reason for the modifications. Another major complaint was the reduction of the key size. Some people conjectured that the NSA would be able to search through a key space of 2^{56}, thus breaking it by brute-force. In later decades, most of these concerns turned out to be unfounded. Section 3.5 provides more information about real and perceived security weaknesses of DES.

Despite of all the criticism and concerns, in 1977 the NBS finally released all specifications of the modified IBM cipher as the *Data Encryption Standard (FIPS PUB 46)* to the public. Even though the cipher is described down to the bit level in the standard, the motivation for parts of the DES design (the so-called design criteria), especially the choice of the substitution boxes, were never officially released.

With the rapid increase in personal computers in the early 1980s and all specifications of DES being publicly available, it become easier to analyze the inner structure of the cipher. During this period, the civilian cryptography research community also grew and DES underwent major scrutiny. However, no serious weaknesses were found until 1990. Originally, DES was only standardized for 10 years, until 1987. Due to the wide use of DES and the lack of security weaknesses, the NIST reaf-

firmed the federal use of the cipher until 1999, when it was finally replaced by the *Advanced Encryption Standard (AES)*.

3.1.1 Confusion and Diffusion

Before we start with the details of DES, it is instructive to look at primitive operations which can be applied in order to achieve strong encryption. According to the famous information theorist Claude Shannon, there are two primitive operations with which strong encryption algorithms can be built:

1. **Confusion** is an encryption operation where the relationship between key and ciphertext is obscured. Today, a common element for achieving confusion is substitution, which is found in both DES and AES.
2. **Diffusion** is an encryption operation where the influence of one plaintext symbol is spread over many ciphertext symbols with the goal of hiding statistical properties of the plaintext. A simple diffusion element is the bit permutation, which is used frequently within DES. AES uses the more advanced Mixcolumn operation.

Ciphers which only perform confusion, such as the Shift Cipher (cf. Sect. 1.4.3) or the World War II encryption machine Enigma, are not secure. Neither are ciphers which only perform diffusion. However, through the concatenation of such operations, a strong cipher can be built. The idea of concatenating several encryption operation was also proposed by Shannon. Such ciphers are known as *product ciphers*. All of today's block ciphers are product ciphers as they consist of rounds which are applied repeatedly to the data (Fig. 3.1).

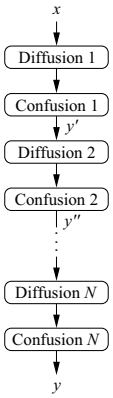

Fig. 3.1 Principle of an N round product cipher, where each round performs a confusion and diffusion operation

Modern block ciphers possess excellent diffusion properties. On a cipher level this means that changing of one bit of plaintext results *on average* in the change of

half the output bits, i.e., the second ciphertext looks statistically independent of the first one. This is an important property to keep in mind when dealing with block ciphers. We demonstrate this behavior with the following simple example.

Example 3.1. Let's assume a small block cipher with a block length of 8 bits. Encryption of two plaintexts x_1 and x_2, which differ only by one bit, should roughly result in something as shown in Fig. 3.2.

Fig. 3.2 Principle of diffusion of a block cipher

Note that modern block ciphers have block lengths of 64 or 128 bit but they show exactly the same behavior if one input bit is flipped.

◇

3.2 Overview of the DES Algorithm

DES is a cipher which encrypts blocks of length of 64 bits with a key of size of 56 bits (Fig. 3.3).

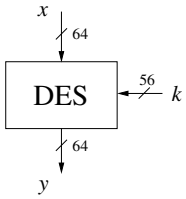

Fig. 3.3 DES block cipher

DES is a symmetric cipher, i.e., the same same key is used for encryption and decryption. DES is, like virtually all modern block ciphers, an iterative algorithm. For each block of plaintext, encryption is handled in 16 rounds which all perform the identical operation. Figure 3.4 shows the round structure of DES. In every round a different subkey is used and all subkeys k_i are derived from the main key k.

Let's now have a more detailed view on the internals of DES, as shown in Fig. 3.5.The structure in the figure is called a *Feistel network*. It can lead to very strong ciphers if carefully designed. Feistel networks are used in many, but certainly not in all, modern block ciphers. (In fact, AES is not a Feistel cipher.) In addition to its potential cryptographic strength, one advantage of Feistel networks is that encryption and decryption are almost the same operation. Decryption requires

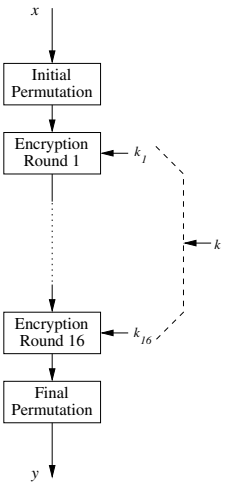

Fig. 3.4 Iterative structure of DES

only a reversed key schedule, which is an advantage in software and hardware implementations. We discuss the Feistel network in the following.

After the initial bitwise permutation IP of a 64-bit plaintext x, the plaintext is split into two halves L_0 and R_0. These two 32-bit halves are the input to the Feistel network, which consists of 16 rounds. The right half R_i is fed into the function f. The output of the f function is XORed (as usually denoted by the symbol $\oplus$) with the left 32-bit half L_i. Finally, the right and left half are swapped. This process repeats in the next round and can be expressed as:

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

where i= 1,. . . ,16. After round 16, the 32-bit halves L_{16} and R_{16} are swapped again, and the final permutation IP^{-1} is the last operation of DES. As the notation suggests, the final permutation IP^{-1} is the inverse of the initial permutation IP. In each round, a round key k_i is derived from the main 56-bit key using what is called the key schedule.

It is crucial to note that the Feistel structure really only encrypts (decrypts) half of the input bits per each round, namely the left half of the input. The right half is copied to the next round unchanged. In particular, the right half is *not encrypted* with the f function. In order to get a better understanding of the working of Feistel cipher, the following interpretation is helpful: Think of the f function as a pseudorandom generator with the two input parameters R_{i-1} and k_i. The output of the pseudorandom generator is then used to encrypt the left half L_{i-1} with an XOR operation. As we saw in Chap. 2, if the output of the f function is not predictable for an attacker, this results in a strong encryption method.

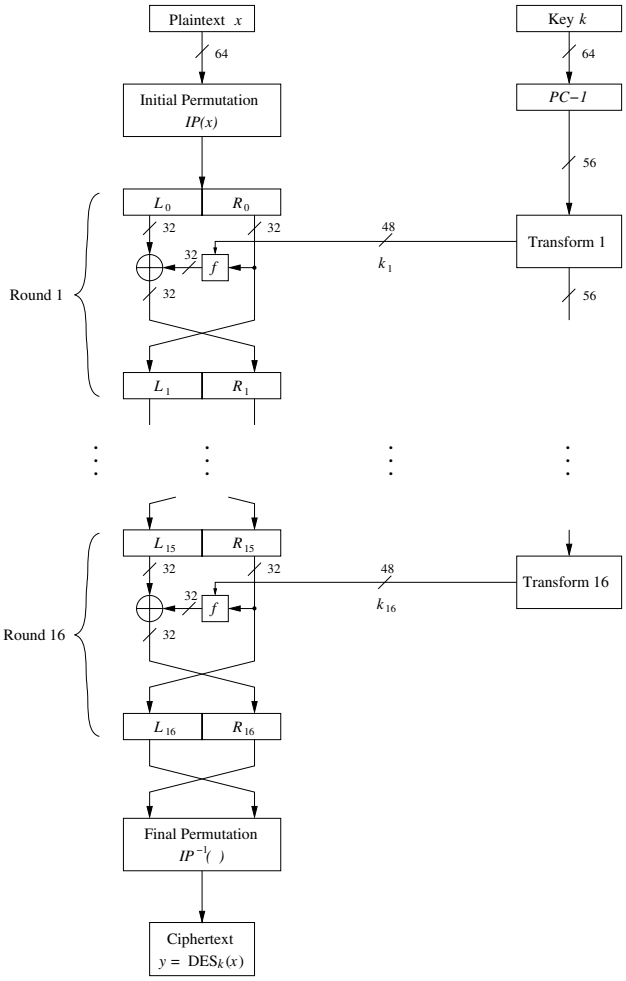

Fig. 3.5 The Feistel structure of DES

The two aforementioned basic properties of ciphers, i.e., confusion and diffusion, are realized within the f-function. In order to thwart advanced analytical attacks, the f-function must be designed extremely carefully. Once the f-function has been designed securely, the security of a Feistel cipher increases with the number of key bits used and the number of rounds.

Before we discuss all components of DES in detail, here is an algebraic description of the Feistel network for the mathematically inclined reader. The Feistel structure of each round bijectively maps a block of 64 input bits to 64 output bits (i.e., every possible input is mapped uniquely to exactly one output, and vice versa). This mapping remains bijective for some arbitrary function f, i.e., even if the embedded function f is not bijective itself. In the case of DES, the function f is in fact a sur-

jective (many-to-one) mapping. It uses nonlinear building blocks and maps 32 input bits to 32 output bits using a 48-bit round key k_i, with $1 \leq i \leq 16$.

3.3 Internal Structure of DES

The structure of DES as depicted in Fig. 3.5 shows the internal functions which we will discuss in this section. The building blocks are the initial and final permutation, the actual DES rounds with its core, the f-function, and the key schedule.

3.3.1 Initial and Final Permutation

As shown in Figs. 3.6 and 3.7, the *initial permutation IP* and the *final permutation* IP^{-1} are bitwise permutations. A bitwise permutation can be viewed as simple crosswiring. Interestingly, permutations can be very easily implemented in hardware but are not particularly fast in software. Note that both permutations do not increase the security of DES at all. The exact rationale for the existence of these two permutations is not known, but it seems likely that their original purpose was to arrange the plaintext, ciphertext and bits in a bytewise manner to make data fetches easier for 8-bit data busses, which were the state-of-the-art register size in the early 1970s.

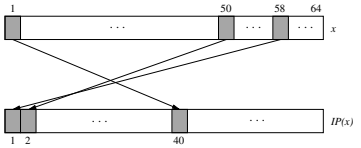

Fig. 3.6 Examples for the bit swaps of the initial permutation

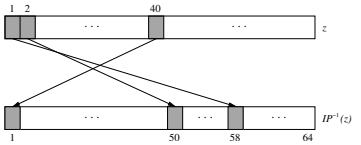

Fig. 3.7 Examples for the bit swaps of the final permutation

The details of the transformation *IP* are given in Table 3.1. This table, like all other tables in this chapter, should be read from left to right, top to bottom. The table indicates that input bit 58 is mapped to output position 1, input bit 50 is mapped to

the second output position, and so forth. The final permutation IP^{-1} performs the inverse operation of IP as shown in Table 3.2.

Table 3.1 Initial permutation IP

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Table 3.2 Final permutation IP^{-1}

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

3.3.2 The f-Function

As mentioned earlier, the f-function plays a crucial role for the security of DES. In round i it takes the right half R_{i-1} of the output of the previous round and the current round key k_i as input. The output of the f-function is used as an XOR-mask for encrypting the left half input bits L_{i-1}.

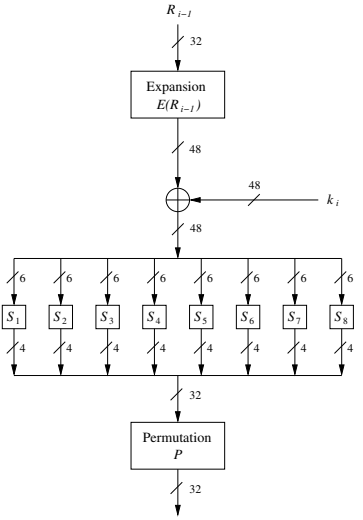

Fig. 3.8 Block diagram of the f-function

The structure of the f-function is shown in Fig. 3.8. First, the 32-bit input is expanded to 48 bits by partitioning the input into eight 4-bit blocks and by expanding each block to 6 bits. This happens in the E-box, which is a special type of permutation. The first block consists of the bits $(1,2,3,4)$, the second one of $(5,6,7,8)$, etc. The expansion to six bits can be seen in Fig. 3.9.

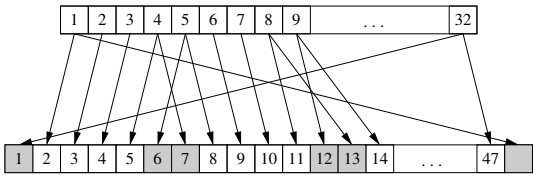

Fig. 3.9 Examples for the bit swaps of the expansion function E

As can be seen from the Table 3.3, exactly 16 of the 32 input bits appear twice in the output. However, an input bit never appears twice in the same 6-bit output block. The expansion box increases the diffusion behavior of DES since certain input bits influence two different output locations.

Table 3.3 Expansion permutation E

E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Next, the 48-bit result of the expansion is XORed with the round key k_i, and the eight 6-bit blocks are fed into eight different *substition boxes*, which are often referred to as *S-boxes*. Each S-box is a lookup table that maps a 6-bit input to a 4-bit output. Larger tables would have been cryptographically better, but they also become much larger; eight 4-by-6 tables were probably close the maximum size which could be fit on a single integrated circuit in 1974. Each S-box contains $2^6 = 64$ entries, which are typically represented by a table with 16 columns and 4 rows. Each entry is a 4-bit value. All S-boxes are listed in Tables 3.4 to 3.11. Note that all S-boxes are different. The tables are to be read as indicated in Fig. 3.10: the most significant bit (MSB) and the least significant bit (LSB) of each 6-bit input select the row of the table, while the four inner bits select the column. The integers 0,1,...,15 of each entry in the table represent the decimal notation of a 4-bit value.

Example 3.2. The S-box input $b = (100101)_2$ indicates the row $11_2 = 3$ (i.e., fourth row, numbering starts with 00_2) and the column $0010_2 = 2$ (i.e., the third column). If the input b is fed into S-box 1, the output is $S_1(37 = 100101_2) = 8 = 1000_2$.

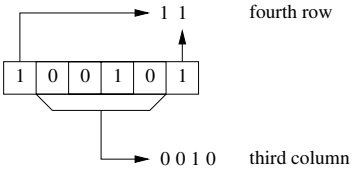

Fig. 3.10 Example of the decoding of the input 100101_2 by S-box 1

◇

Table 3.4 S-box S_1

S_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Table 3.5 S-box S_2

S_2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
1	03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
2	00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
3	13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09

Table 3.6 S-box S_3

S_3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	08
1	13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
2	13	06	04	09	08	15	03	00	11	01	02	12	05	10	14	07
3	01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12

The S-boxes are the core of DES in terms of cryptographic strength. They are the only nonlinear element in the algorithm and provide confusion. Even though the entire specification of DES was released by NBS/NIST in 1977, the motivation for the choice of the S-box tables was never completely revealed. This often gave rise

Table 3.7 S-box S_4

S_4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	07	13	14	03	00	06	09	10	01	02	08	05	11	12	04	15
1	13	08	11	05	06	15	00	03	04	07	02	12	01	10	14	09
2	10	06	09	00	12	11	07	13	15	01	03	14	05	02	08	04
3	03	15	00	06	10	01	13	08	09	04	05	11	12	07	02	14

Table 3.8 S-box S_5

S_5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
1	14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
2	04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
3	11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03

Table 3.9 S-box S_6

S_6	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	01	10	15	09	02	06	08	00	13	03	04	14	07	05	11
1	10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
2	09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
3	04	03	02	12	09	05	15	10	11	14	01	07	06	00	08	13

Table 3.10 S-box S_7

S_7	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	04	11	02	14	15	00	08	13	03	12	09	07	05	10	06	01
1	13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
2	01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
3	06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12

Table 3.11 S-box S_8

S_8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
1	01	15	13	08	10	03	07	04	12	05	06	11	00	14	09	02
2	07	11	04	01	09	12	14	02	00	06	10	13	15	03	05	08
3	02	01	14	07	04	10	08	13	15	12	09	00	03	05	06	11

to speculation, in particular with respect to the possible existence of a secret back door or some other intentionally constructed weakness, which could be exploited by the NSA. However, now we know that the S-boxes were designed according to the criteria listed below.

1. Each S-box has six input bits and four output bits.
2. No single output bit should be too close to a linear combination of the input bits.
3. If the lowest and the highest bits of the input are fixed and the four middle bits are varied, each of the possible 4-bit output values must occur exactly once.
4. If two inputs to an S-box differ in exactly one bit, their outputs must differ in at least two bits.

5. If two inputs to an S-box differ in the two middle bits, their outputs must differ in at least two bits.
6. If two inputs to an S-box differ in their first two bits and are identical in their last two bits, the two outputs must be different.
7. For any nonzero 6-bit difference between inputs, no more than 8 of the 32 pairs of inputs exhibiting that difference may result in the same output difference.
8. A collision (zero output difference) at the 32-bit output of the eight S-boxes is only possible for three adjacent S-boxes.

Note that some of these design criteria were not revealed until the 1990s. More information about the issue of the secrecy of the design criteria is found in Sect. 3.5.

The S-boxes are the most crucial elements of DES because they introduce a *nonlinearity* to the cipher, i.e.,

$$S(a) \oplus S(b) \neq S(a \oplus b).$$

Without a nonlinear building block, an attacker could express the DES input and output with a system of linear equations where the key bits are the unknowns. Such systems can easily be solved, a fact that was used in the LFSR attack in Sect. 2.3.2. However, the S-boxes were carefully designed to also thwart advanced mathematical attacks, in particular *differential cryptanalysis*. Interestingly, differential cryptanalysis was first discovered in the research community in 1990. At this point, the IBM team declared that the attack was known to the designers at least 16 years earlier, and that DES was especially designed to withstand differential cryptanalysis.

Finally, the 32-bit output is permuted bitwise according to the P permutation, which is given in Table 3.12. Unlike the initial permutation IP and its inverse IP^{-1}, the permutation P introduces diffusion because the four output bits of each S-box are permuted in such a way that they affect several different S-boxes in the following round. The diffusion caused by the expansion, S-boxes and the permutation P guarantees that every bit at the end of the fifth round is a function of every plaintext bit and every key bit. This behavior is known as the *avalanche effect*.

Table 3.12 The permutation P within the f-function

P							
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

3.3.3 Key Schedule

The *key schedule* derives 16 round keys k_i, each consisting of 48 bits, from the original 56-bit key. Another term for round key is subkey. First, note that the DES input key is often stated as 64-bit, where every eighth bit is used as an odd parity bit over the preceding seven bits. It is not quite clear why DES was specified that way. In any case, the eight parity bits are **not** actual key bits and do not increase the security. DES is a 56-bit cipher, not a 64-bit one.

As shown in Fig. 3.11, the 64-bit key is first reduced to 56 bits by ignoring every eighth bit, i.e., the parity bits are stripped in the initial $PC-1$ permutation. Again, the parity bits certainly do not increase the key space! The name $PC-1$ stands for "permuted choice one". The exact bit connections that are realized by $PC-1$ are given in Table 3.13.

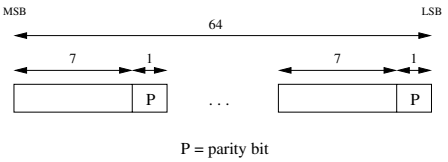

Fig. 3.11 Location of the eight parity bits for a 64-bit input key

Table 3.13 Initial key permutation $PC-1$

$PC-1$							
57	49	41	33	25	17	9	1
58	50	42	34	26	18	10	2
59	51	43	35	27	19	11	3
60	52	44	36	63	55	47	39
31	23	15	7	62	54	46	38
30	22	14	6	61	53	45	37
29	21	13	5	28	20	12	4

The resulting 56-bit key is split into two halves C_0 and D_0, and the actual key schedule starts as shown in Fig. 3.12. The two 28-bit halves are cyclically shifted, i.e., rotated, left by one or two bit positions depending on the round i according to the following rules:

- In rounds $i = 1, 2, 9, 16$, the two halves are rotated left by one bit.
- In the other rounds where $i \neq 1, 2, 9, 16$, the two halves are rotated left by two bits.

Note that the rotations only take place within either the left or the right half. The total number of rotation positions is $4 \cdot 1 + 12 \cdot 2 = 28$. This leads to the interesting property that $C_0 = C_{16}$ and $D_0 = D_{16}$. This is very useful for the decryption key

schedule where the subkeys have to be generated in reversed order, as we will see in Sect. 3.4.

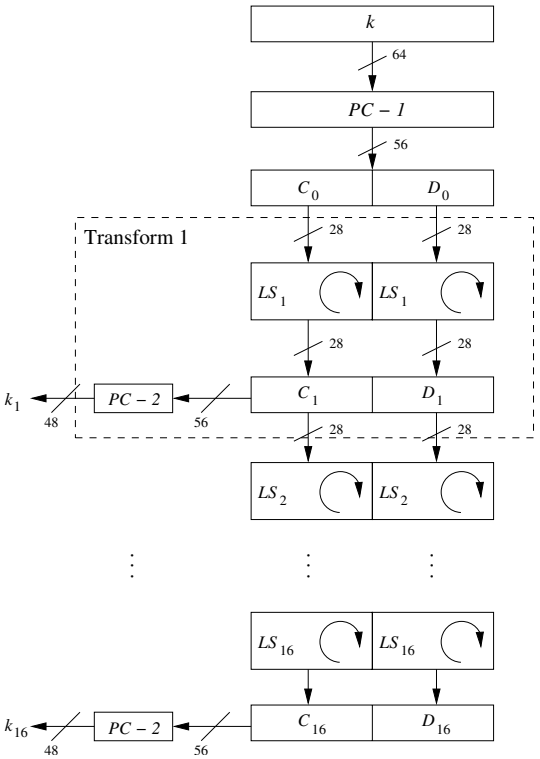

Fig. 3.12 Key schedule for DES encryption

To derive the 48-bit round keys k_i, the two halves are permuted bitwise again with $PC-2$, which stands for "permuted choice 2". $PC-2$ permutes the 56 input bits coming from C_i and D_i and ignores 8 of them. The exact bit-connections of $PC-2$ are given in Table 3.14.

Table 3.14 Round key permutation $PC-2$

$PC-2$							
14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Note that every round key is a selection of 48 permuted bits of the input key k. The key schedule is merely a method of realizing the 16 permutations systematically. Especially in hardware, the key schedule is very easy to implement. The key schedule is also designed so that each of the 56 key bits is used in different round keys; each bit is used in approximately 14 of the 16 round keys.

3.4 Decryption

One advantage of DES is that decryption is essentially the same function as encryption. This is because DES is based on a Feistel network. Figure 3.13 shows a block diagram for DES decryption. Compared to encryption, only the key schedule is reversed, i.e., in decryption round 1, subkey 16 is needed; in round 2, subkey 15; etc. Thus, when in decryption mode, the key schedule algorithm has to generate the round keys as the sequence $k_{16}, k_{15}, \ldots, k_1$.

Reversed Key Schedule

The first question that we have to clarify is how, given the initial DES key k, can we easily generate k_{16}? Note that we saw above that $C_0 = C_{16}$ and $D_0 = D_{16}$. Hence k_{16} can be directly derived after $PC-1$.

$$\begin{aligned} k_{16} &= PC-2(C_{16}, D_{16}) \\ &= PC-2(C_0, D_0) \\ &= PC-2(PC-1(k)) \end{aligned}$$

To compute k_{15} we need the intermediate variables C_{15} and D_{15}, which can be derived from C_{16}, D_{16} through cyclic *right shifts (RS)*:

$$\begin{aligned} k_{15} &= PC-2(C_{15}, D_{15}) \\ &= PC-2(RS_2(C_{16}), RS_2(D_{16})) \\ &= PC-2(RS_2(C_0), RS_2(D_0)) \end{aligned}$$

The subsequent round keys $k_{14}, k_{13}, \ldots, k_1$ are derived via right shifts in a similar fashion. The number of bits shifted right for each round key in decryption mode

- In decryption round 1, the key is not rotated.
- In decryption rounds 2, 9, and 16 the two halves are rotated right by one bit.
- In the other rounds 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14 and 15 the two halves are rotated right by two bits.

Figure 3.14 shows the reversed key schedule for decryption.

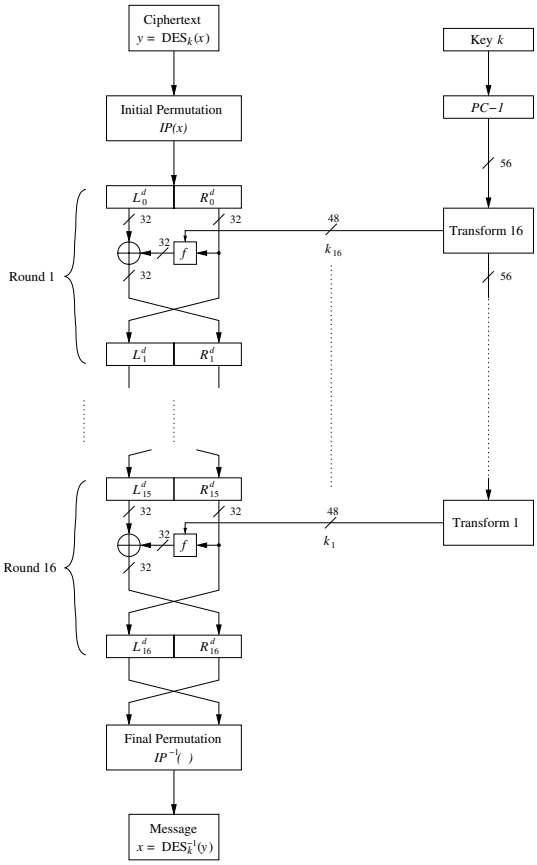

Fig. 3.13 DES decryption

Decryption in Feistel Networks

We have not addressed the core question: Why is the decryption function essentially the same as the encryption function? The basic idea is that the decryption function reverses the DES encryption in a round-by-round manner. That means that decryption round 1 reverses encryption round 16, decryption round 2 reverses encryption round 15, and so on. Let's first look at the initial stage of decryption by looking at Fig. 3.13. Note that the right and left halves are swapped in the last round of DES:

$$(L_0^d, R_0^d) = IP(Y) = IP(IP^{-1}(R_{16}, L_{16})) = (R_{16}, L_{16})$$

And thus:

$$L_0^d = R_{16}$$
$$R_0^d = L_{16} = R_{15}$$

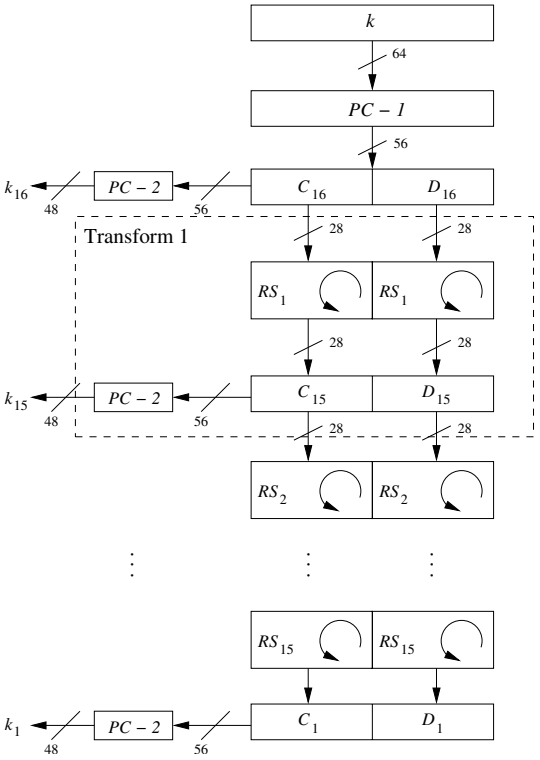

Fig. 3.14 Reversed key schedule for decryption of DES

Note that all variables in the decryption routine are marked with the superscript d, whereas the encryption variables do not have superscripts. The derived equation simply says that the input of the first round of decryption is the output of the last round of encryption because final and initial permutations cancel each other out. We will now show that the first decryption round reverses the last encryption round. For this, we have to express the output values (L_1^d, R_1^d) of the first decryption round 1 in terms of the input values of the last encryption round (L_{15}, R_{15}) . The first one is easy:

$$L_1^d = R_0^d = L_{16} = R_{15}$$

We now look at how R_1^d is computed:

$$\begin{aligned} R_1^d &= L_0^d \oplus f(R_0^d, k_{16}) = R_{16} \oplus f(L_{16}, k_{16}) \\ R_1^d &= [L_{15} \oplus f(R_{15}, k_{16})] \oplus f(R_{15}, k_{16}) \\ R_1^d &= L_{15} \oplus [f(R_{15}, k_{16}) \oplus f(R_{15}, k_{16})] = L_{15} \end{aligned}$$

The crucial step is shown in the last equation above: An identical output of the f-function is XORed twice to L_{15}. These operations cancel each other out, so that

$R_1^d = L_{15}$. Hence, after the first decryption round, we in fact have computed the same values we had *before* the last encryption round. Thus, the first decryption round reverses the last encryption round. This is an iterative process which continues in the next 15 decryption rounds and that can be expressed as:

$$\begin{aligned} L_i^d &= R_{16-i}, \\ R_i^d &= L_{16-i} \end{aligned}$$

where $i = 0, 1, \ldots, 16$. In particular, after the last decryption round:

$$\begin{aligned} L_{16}^d &= R_{16-16} = R_0 \\ R_{16}^d &= L_0 \end{aligned}$$

Finally, at the end of the decryption process, we have to reverse the initial permutation:

$$IP^{-1}(R_{16}^d, L_{16}^d) = IP^{-1}(L_0, R_0) = IP^{-1}(IP(x)) = x$$

where x is the plaintext that was the input to the DES encryption.

3.5 Security of DES

As we discussed in Sect. 1.2.2, ciphers can be attacked in several ways. With respect to cryptographic attacks, we distinguish between exhaustive key search or brute-force attacks, and analytical attacks. The latter was demonstrated with the LFSR attack in Sect. 2.3.2, where we could easily break a stream cipher by solving a system of linear equations. Shortly after DES was proposed, two major criticisms against the cryptographic strength of DES centered around two arguments:

1. The key space is too small, i.e., the algorithm is vulnerable against brute-force attacks.
2. The design criteria of the S-boxes was kept secret and there might have existed an analytical attack that exploits mathematical properties of the S-boxes, but which is only known to the DES designers.

We discuss both types of attacks below. However, we also state the main conclusion about DES security already here: Despite very intensive cryptanalysis over the lifetime of DES, current analytical attacks are not very efficient. However, DES can relatively easily be broken with an exhaustive key-search attack and, thus, plain DES is not suited for most applications any more.

3.5.1 Exhaustive Key Search

The first criticism is nowadays certainly justified. The original cipher proposed by IBM had a key length of 128 bits and it is suspicious that it was reduced to 56 bits. The official statement that a cipher with a shorter key length made it easier to implement the DES algorithm on a single chip in 1974 does not sound too convincing. For clarification, let's recall the principle of an exhaustive key search (or brute-force attack):

Definition 3.5.1 DES Exhaustive key search
Input*: at least one pair of plaintext–ciphertext* (x, y)
Output*: k, such that* $y = DES_k(x)$
Attack*: Test all* 2^{56} *possible keys until the following condition is fulfilled:*

$$DES_{k_i}^{-1}(y) \stackrel{?}{=} x \ , \ \ i = 0, 1, \ldots, 2^{56} - 1.$$

Note that there is a small chance of $1/2^{16}$ that an incorrect key is found, i.e., a key k which decrypts only the one ciphertext y correctly but not subsequent ciphertexts. If one wants to rule out this possibility, an attacker must check such a key candidate with a second plaintext–ciphertext pair. More about this is found in Sect. 5.2.

Regular computers are not particularly well suited to perform the 2^{56} key tests necessary, but special-purpose key-search machines are an option. It seems highly likely that large (government) institutions have long been able to build such *brute-force crackers*, which can break DES in a matter of days. In 1977, Whitfield Diffie and Martin Hellman [59] estimated that it was possible to build an exhaustive key-search machine for approximately $20,000,000. Even though they later stated that their cost estimate had been too optimistic, it was clear from the beginning that a cracker could be built with sufficient funding.

At the rump session of the CRYPTO 1993 conference, Michael Wiener proposed the design of a very efficient key-search machine which used pipelining techniques. An update of his proposal can be found in [174]. He estimated the cost of his design at approximately $1,000,000, and the time required to find the key at 1.5 days. This was a proposal only, and the machine was not built. In 1998, however, the EFF (Electronic Frontier Foundation) built the hardware machine *Deep Crack*, which performed a brute-force attack against DES in 56 hours. Figure 3.15 shows a photo of Deep Crack. The machine consisted of 1800 integrated circuits, where each had 24 key-test units. The average search time of Deep Crack was 15 days, and the machine was built for less than $250,000. The successful break with Deep Crack was considered the official demonstration that DES is no longer secure against determined attacks by many people. Please note that this break does not imply that a weak algorithm had been in use for more than 20 years. It was only possible to build Deep Crack at such a relatively low price because digital hardware had become

cheap. In the 1980s it would have been impossible to build a DES cracker without spending many millions of dollars. It can be speculated that only government agencies were willing to spend such an amount of money for code breaking.

Fig. 3.15 Deep Crack — the hardware exhaustive key-search machine that broke DES in 1998 (reproduced with permission from Paul Kocher)

DES brute-force attacks also provide an excellent case study for the continuing decrease in hardware costs. In 2006, the *COPACOBANA (Cost-Optimized Parallel Code-Breaker)* machine was built based on commercial integrated circuits by a team of researchers from the Universities of Bochum and Kiel in Germany (the authors of this book were heavily involved in this effort). COPACOBANA allows one to break DES with an average search time of less than 7 days. The interesting part of this undertaking is that the machine could be built with hardware costs in the $10,000 range. Figure 3.16 shows a picture of COPACOBANA.

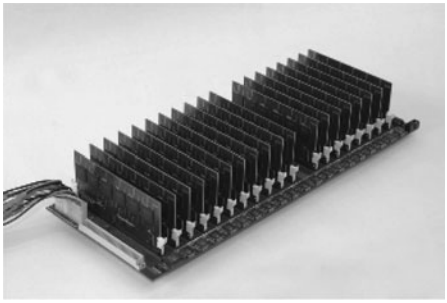

Fig. 3.16 COPACOBANA — A cost-optimized parallel code breaker

In summary, a key size of 56 bits is too short to encrypt confidential data nowadays. Hence, single DES should only be used for applications where only short-term security is needed — say, a few hours — or where the value of the encrypted data is very low. However, variants of DES, in particular 3DES, are still secure.

3.5.2 Analytical Attacks

As was shown in the first chapter, analytical attacks can be very powerful. Since the introduction of DES in the mid-1970s, many excellent researchers in academia (and without doubt many excellent researchers in intelligence agencies) tried to find weaknesses in the structure of DES which allowed them to break the cipher. It is a major triumph for the designers of DES that no weakness was found until 1990. In this year, Eli Biham and Adi Shamir discovered what is called *differential cryptanalysis (DC)*. This is a powerful attack which is *in principle* applicable to any block cipher. However, it turned out that the DES S-boxes are particularly resistant against this attack. In fact, one member of the original IBM design team declared after the discovery of DC that they had been aware of the attack at the time of design. Allegedly, the reason why the S-box design criteria were not made public was that the design team did not want to make such a powerful attack public. If this claim is true — and all circumstances support it — it means that the IBM and NSA team was 15 years ahead of the research community. It should be noted, however, that in the 1970s and 1980s relatively few people did active research in cryptography.

In 1993 a related but distinct analytical attack was published by Mitsuru Matsui, which was named *linear cryptanalysis (LC)*. Similar to differential cryptanalysis, the effectiveness of this attack also heavily depends on the structure of the S-boxes.

What is the practical relevance of these two analytical attacks against DES? It turns out that an attacker needs 2^{47} plaintext–ciphertext pairs for a successful DC attack. This assumes particularly chosen plaintext blocks; for random plaintext 2^{55} pairs are needed! In the case of LC, an attacker needs 2^{43} plaintext–ciphertext pairs. All these numbers seem highly impractical for several reasons. First, an attacker needs to know an extremely large number of plaintexts, i.e., pieces of data which are supposedly encrypted and thus hidden from the attacker. Second, collecting and storing such an amount of data takes a long time and requires considerable memory resources. Third, the attack only recovers one key. (This is actually one of many arguments for introducing key freshness in cryptographic applications.) As a result of all these arguments, it does not seem likely that DES can be broken with either DC or LC in real-world systems. However, both DC and LC are very powerful attacks which are applicable to many other block ciphers. Table 3.15 provides an overview of proposed and realized attacks against DES over the last three decades. Some entries refer to what is known as the DES Challenges. Starting in 1997, several DES-breaking challenges were organized by the company RSA Security.

3.6 Implementation in Software and Hardware

In the following, we provide a brief assessment of DES implementation properties in software and hardware. When we talk about software, we refer to DES implementations running on desktop CPUs or embedded microprocessors like smart cards

Table 3.15 History of full-round DES attacks

Date	Proposed or implemented attacks
1977	W. Diffie and M. Hellman propose cost estimate for key-search machine
1990	E. Biham and A. Shamir propose differential cryptanalysis, which requires 2^{47} chosen plaintexts
1993	M. Wiener proposes detailed hardware design for key-search machine with an average search time of 36 h and estimated cost of $1,000,000
1993	M. Matsui proposes linear cryptanalysis, which requires 2^{43} chosen ciphertexts
Jun. 1997	DES Challenge I broken through brute-force; distributed effort on the Internet took 4.5 months
Feb. 1998	DES Challenge II–1 broken through brute-force; distributed effort on the Internet took 39 days
Jul. 1998	DES Challenge II–2 broken through brute-force; Electronic Frontier Foundation built the Deep Crack key-search machine for about $250,000. The attack took 56 h (15 days average)
Jan. 1999	DES Challenge III broken through brute-force by distributed Internet effort combined with Deep Crack and a total search time of 22 hours
Apr. 2006	Universities of Bochum and Kiel built COPACOBANA key-search machine based on low-cost FPGAs for approximately $10,000. Average search time is 7 days

or cell phones. Hardware refers to DES implementations running on ICs such as application-specific integrated circuits (ASICs) or field programmable gate arrays (FPGAs).

Software

A straightforward software implementation which follows the data flow of most DES descriptions, such as the one presented in this chapter, results in a very poor performance. This is due to the fact that many of the atomic DES operations involve bit permutation, in particular the E and P permutation, which are slow in software. Similarly, small S-boxes such as used in DES are efficient in hardware but only moderately efficient on modern CPUs. There have been numerous methods proposed for accelerating DES software implementations. The general idea is to use tables with precomputed values of several DES operations, e.g., of several S-boxes and the permutation. Optimized implementations require about 240 cycles for encrypting one block on a 32-bit CPU. On a 2-GHz CPU this translates into a theoretical throughput of about 533 Mbits/s. 3DES, which is considerably more secure than single DES, runs at almost exactly 1/3 of the DES speed. Note that nonoptimized implementations are considerably slower, often below 100 Mbit/s.

A notable method for accelerating software implementations of DES is *bit-slicing*, developed by Eli Biham [20]. On a 300-MHz DEC Alpha workstation an encryption rate of 137 Mbit/sec has been reported, which was much faster than a standard DES implementation at that time. The limitation of bit-slicing, however, is that several blocks are encrypted in parallel, which can be a drawback for certain

modes of operation such as Cipher Block Chaining (CBC) and Output Feedback (OFB) mode (cf. Chap. 5).

Hardware

One design criterion for DES was its efficiency in hardware. Permutations such as the E, P, IP and IP^{-1} permutations are very easy to implement in hardware, as they only require wiring but no logic. The small 6-by-4 S-boxes are also relatively easily realizable in hardware. Typically, they are implemented with Boolean logic, i.e., logic gates. On average, one S-box requires about 100 gates.

An area-efficient implementation of a single DES round can be done with less than 3000 gates. If a high throughput is desired, DES can be implemented extremely fast by fitting multiple rounds in one circuit, e.g., by using pipelining. On modern ASICs and FPGAs throughput rates of several 100 Gbit/sec are possible. On the other end of the performance spectrum, very small implementations with fewer than 3000 gates even fit onto lowcost radio frequency identification (RFID) chips.

3.7 DES Alternatives

There exist a wealth of other block ciphers. Even though there are many ciphers which have security weaknesses or which are not well investigated, there are also many block ciphers which appear very strong. In the following a brief list of ciphers is given which can be of interest depending on the application needs.

3.7.1 The Advanced Encryption Standard (AES) and the AES Finalist Ciphers

By now, the algorithm of choice for many, many applications has become the Advanced Encryption Standard (AES), which will be introduced in detail in the following chapter. AES is with its three key lengths of 128, 192 and 256 bit secure against brute-force attacks for several decades, and there are no analytical attacks with any reasonable chance of success known.

AES was the result of an open competition, and in the last stage of the selection process there were four other finalist algorithms. These are the block ciphers *Mars*, *RC6*, *Serpent* and *Twofish*. All of them are cryptographically strong and quite fast, especially in software. Based on today's knowledge, they can all be recommended. Mars, Serpent and Twofish can be used royalty-free.

3.7.2 Triple DES (3DES) and DESX

An alternative to AES or the AES finalist algorithms is *triple DES*, often denoted as *3DES*. 3DES consists of three subsequent DES encryptions

$$y = DES_{k_3}(DES_{k_2}(DES_{k_1}(x)))$$

with different keys, as shown in Fig. 3.17.

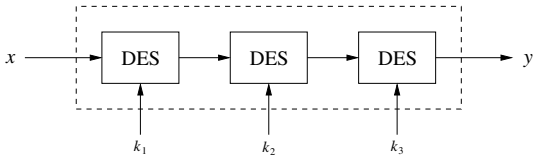

Fig. 3.17 Triple DES (3DES)

3DES seems resistant to both brute-force attacks and any analytical attack imaginable at the moment. See Chap. 5 for more information on double and triple encryption. Another version of 3DES is

$$y = DES_{k_3}(DES_{k_2}^{-1}(DES_{k_1}(x))).$$

The advantage here is that 3DES performs single DES encryption if $k_3 = k_2 = k_1$, which is sometimes desired in implementations that should also support single DES for legacy reasons. 3DES is very efficient in hardware but not particularly in software. It is popular in financial applications as well as for protecting biometric information in electronic passports.

A different approach for strengthening DES is to use key whitening. For this, two additional 64-bit keys k_1 and k_2 are XORed to the plaintext and ciphertext, respectively, prior to and after the DES algorithm. This yields the following encryption scheme:

$$y = DES_{k,k_1,k_2}(x) = DES_k(x \oplus k_1) \oplus k_2$$

This surprisingly simple modification makes DES much more resistant against exhaustive key searches. More about key whitening is said in Sect. 5.3.3.

3.7.3 Lightweight Cipher PRESENT

Over the last few years, several new block algorithms which are classified as "lightweight ciphers" have been proposed. Lightweight commonly refers to algorithms with a very low implementation complexity, especially in hardware. Trivium (Sect. 2.3.3) is an example of a lightweight stream cipher. A promising block cipher candidate is *PRESENT*, which was designed specifically for applications such as

RFID tags or other pervasive computing applications that are extremely power or cost constrained. (One of the book authors participated in the design of PRESENT.)

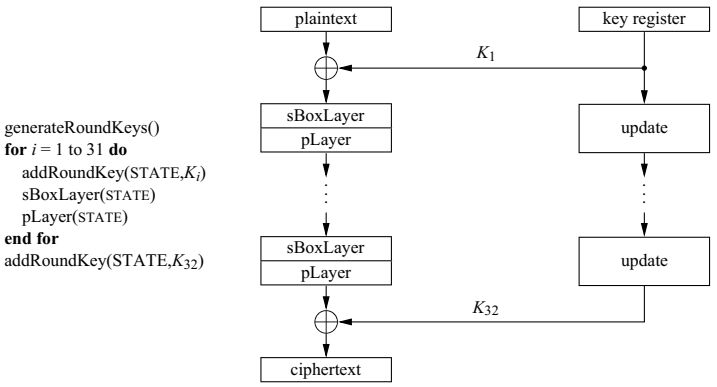

Fig. 3.18 Internal structure and pseudocode of the block cipher PRESENT

Unlike DES, PRESENT is not based on a Feistel network. Instead it is a substitution-permutation network (SP-network) and consists of 31 rounds. The block length is 64 bits, and two key lengths of 80 and 128 bits are supported. Each of the 31 rounds consists of an XOR operation to introduce a round key K_i for $1 \leq i \leq 32$, where K_{32} is used after round 31, a nonlinear substitution layer (sBoxLayer) and a linear bitwise permutation (pLayer). The nonlinear layer uses a single 4-bit S-box S, which is applied 16 times in parallel in each round. The key schedule generates 32 round keys from the user supplied key. The encryption routine of the cipher is described in pseudocode in Fig. 3.18, and each stage is now specified in turn.

addRoundKey At the beginning of each round, the round key K_i is XORed to the current STATE.

sBoxLayer PRESENT uses a single 4-bit to 4-bit S-box. This is a direct consequence of the pursuit of hardware efficiency, since such an S-Box allows a much more compact implementation than, e.g., an 8-bit S-box. The S-box entries in hexadecimal notation are given in Table 3.16.

Table 3.16 The PRESENT S-box in hexadecimal notation

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

The 64 bit data path $b_{63} \ldots b_0$ is referred to as *state*. For the sBoxLayer the current state is considered as sixteen 4-bit words $w_{15} \ldots w_0$, where $w_i = b_{4*i+3}||b_{4*i+2}||b_{4*i+1}||b_{4*i}$ for $0 \leq i \leq 15$, and the output are the 16 words $S[w_i]$.

pLayer Just like DES, the mixing layer was chosen as a bit permutation, which can be implemented extremely compactly in hardware. The bit permutation used in PRESENT is given by Table 3.17. Bit i of STATE is moved to bit position $P(i)$.

Table 3.17 The permutation layer of PRESENT

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

The bit permutation is quite regular and can in fact be expressed in the following way:

$$P(i) = \begin{cases} i \cdot 16 \mod 63, & i \in \{0,\ldots,62\} \\ 63, & i = 63. \end{cases}$$

Key Schedule We describe in the following the key schedule for PRESENT with an 80-bit key. Since the main applications of PRESENT are low-cost systems, this key length is in most cases appropriate. (Details of the key schedule for PRESENT-128 can be found in [29].) The user-supplied key is stored in a key register K and is represented as $k_{79}k_{78}\ldots k_0$. At round i the 64-bit round key $K_i = \kappa_{63}\kappa_{62}\ldots\kappa_0$ consists of the 64 leftmost bits of the current contents of register K. Thus at round i we have:

$$K_i = \kappa_{63}\kappa_{62}\ldots\kappa_0 = k_{79}k_{78}\ldots k_{16}$$

The first subkey K_1 is a direct copy of 64 bit of the user supplied key. For the following subkeys $K_2,\ldots,K_{32}$ the key register $K = k_{79}k_{78}\ldots k_0$ is updated as follows:

1. $[\mathtt{k}_{79}\mathtt{k}_{78}\ldots\mathtt{k}_1\mathtt{k}_0] = [\mathtt{k}_{18}\mathtt{k}_{17}\ldots\mathtt{k}_{20}\mathtt{k}_{19}]$
2. $[\mathtt{k}_{79}\mathtt{k}_{78}\mathtt{k}_{77}\mathtt{k}_{76}] = S[\mathtt{k}_{79}\mathtt{k}_{78}\mathtt{k}_{77}\mathtt{k}_{76}]$
3. $[\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] = [\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] \oplus \mathtt{round\_counter}$

Thus, the key schedule consists of three operations: (1) the key register is rotated by 61 bit positions to the left, (2) the leftmost four bits are passed through the PRESENT S-box, and (3) the `round_counter` value i is XORed with bits $\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}$ of K, where the least significant bit of `round_counter` is on the right. This counter is a simple integer which takes the values $(00001, 00010, \ldots, 11111)$. Note that for the derivation of K_2 the counter value 00001 is used; for K_3, the counter value 00010; and so on.

Implementation As a result of the aggressively hardware-optimized design of PRESENT, its software performance is not very competitive relative to modern ciphers like AES. An optimized software implementation on a Pentium III CPU in

C achieves a throughput of about 60 Mbit/s at a frequency of 1 GHz. However, it performs quite well on small microprocessors, which are common in inexpensive consumer products.

PRESENT-80 can be implemented in hardware with area requirements of approximately 1600 gate equivalences [147], where the encryption of one 64-bit plaintext block requires 32 clock cycles. As an example, at a clock rate of 1 MHz, which is quite typical on low-cost devices, a throughput of 2 Mbit/s is achieved, which is sufficient for most such applications. It is possible to realize the cipher with as few as approximately 1000 gate equivalences, where the encryption of one 64-bit plaintext requires 547 clock cycles. A fully pipelined implementation of PRESENT with 31 encryption stages achieves a throughput of 64 bit per clock cycle, which can be tranlsated into encryption throughputs of more than 50 Gbit/s.

Even though no attacks against PRESENT are known at the time of writing, it should be noted that it is a relatively new block cipher.

3.8 Discussion and Further Reading

DES History and Attacks Even though plain DES (i.e., non-3DES) is today mainly used in legacy applications, its history helps us understand the evolution of cryptography since the mid-1970s from an obscure discipline almost solely studied in government organizations towards an open discipline with many players in industry and academia. A summary of the DES history can be found in [165]. The two main analytical attacks developed against DES, differential and linear cryptanalysis, are today among the most powerful general methods for breaking block ciphers. Readers interested in the theory of block ciphers are encouraged to study these attacks. Good descriptions are given in [21, 114].

As we have seen in this chapter, DES should no longer be used since a brute-force attack can be accomplished at low cost in little time with cryptanalytical hardware. The two machines built outside governments, Deep Crack and COPACOBANA, are instructive examples of how to build low-cost "supercomputers" for very narrowly defined computational tasks. More information about Deep Crack can be found on the Internet [78] and about COPACOBANA in the articles [105, 88] and online at [47]. Readers interested in the fascinating area of cryptanalytical computers in general should take a look at the SHARCS (Special-purpose Hardware for Attacking Cryptographic Systems) workshop series, which started in 2005 and has information online [170].

DES Alternatives It should be noted that hundreds of block ciphers have been proposed over the last three decades, especially in the late 1980s and in the 1990s. DES has influenced the design of many other encryption algorithms. It is probably fair to say that the majority of today's successful block ciphers have borrowed ideas from DES. Some of the popular block ciphers are also based on Feistel networks as is DES. Examples of Feistel ciphers include Blowfish, CAST, KASUMI, Mars,

MISTY1, Twofish and RC6. One cipher which is well known and markedly different from DES is IDEA; it uses arithmetic in three different algebraic structures as atomic operations.

DES is a good example of a block cipher which is very efficient in hardware. The recent advent of pervasive computing has created a need for extremely small ciphers for applications such as RFID tags or low-cost smart cards, e.g., for high-volume public transportation payment tickets. Good references for PRESENT are [29, 147]. In addition to PRESENT, other recently proposed very small block ciphers include Clefia [48], HIGHT [93] and mCrypton [111]. A good overview of the new field of lightweight cryptography is given in the surveys [71, 98]. A more in-depth treatment of lightweight algorithms can be found in the Ph.D. dissertation [135].

Implementation With respect to software implementation of DES, an early reference is [20]. More advanced techniques are described in [106]. The powerful method of bit-slicing is applicable not only to DES but to most other ciphers.

Regarding DES hardware implementation, an early but still very interesting reference is [169]. There are many descriptions of high-performance implementations of DES on a variety of hardware platforms, including FPGAs [163], standard ASICs as well as more exotic semiconductor technology [67].

3.9 Lessons Learned

- DES was the dominant symmetric encryption algorithm from the mid-1970s to the mid-1990s. Since 56-bit keys are no longer secure, the Advanced Encryption Standard (AES) was created.
- Standard DES with 56-bit key length can be broken relatively easily nowadays through an exhaustive key search.
- DES is quite robust against known analytical attacks: In practice it is very difficult to break the cipher with differential or linear cryptanalysis.
- DES is reasonably efficient in software and very fast and small in hardware.
- By encrypting with DES three times in a row, triple DES (3DES) is created, against which no practical attack is currently known.
- The "default" symmetric cipher is nowadays often AES. In addition, the other four AES finalist ciphers all seem very secure and efficient.
- Since about 2005 several proposals for lightweight ciphers have been made. They are suited for resource-constrained applications.

Problems

3.1. As stated in Sect. 3.5.2, one important property which makes DES secure is that the S-boxes are nonlinear. In this problem we verify this property by computing the output of S_1 for several pairs of inputs.

Show that $S_1(x_1) \oplus S_1(x_2) \neq S_1(x_1 \oplus x_2)$, where "$\oplus$" denotes bitwise XOR, for:

1. $x_1 = 000000, x_2 = 000001$
2. $x_1 = 111111, x_2 = 100000$
3. $x_1 = 101010, x_2 = 010101$

3.2. We want to verify that $IP(\cdot)$ and $IP^{-1}(\cdot)$ are truly inverse operations. We consider a vector $x = (x_1, x_2, \ldots, x_{64})$ of 64 bit. Show that $IP^{-1}(IP(x)) = x$ for the first five bits of x, i.e. for x_i, $i = 1, 2, 3, 4, 5$.

3.3. What is the output of the first round of the DES algorithm when the plaintext and the key are both all zeros?

3.4. What is the output of the first round of the DES algorithm when the plaintext and the key are both all ones?

3.5. Remember that it is desirable for good block ciphers that a change in one input bit affects many output bits, a property that is called diffusion or the avalanche effect. We try now to get a feeling for the avalanche property of DES. We apply an input word that has a "1" at bit position 57 and all other bits as well as the key are zero. (Note that the input word has to run through the initial permutation.)

1. How many S-boxes get different inputs compared to the case when an all-zero plaintext is provided?
2. What is the minimum number of output bits of the S-boxes that will change according to the S-box design criteria?
3. What is the output after the first round?
4. How many output bit after the first round have actually changed compared to the case when the plaintext is all zero? (Observe that we only consider a single round here. There will be more and more output differences after every new round. Hence the term *avalanche effect*.)

3.6. An avalanche effect is also desirable for the key: A one-bit change in a key should result in a dramatically different ciphertext if the plaintext is unchanged.

1. Assume an encryption with a given key. Now assume the key bit at position 1 (prior to $PC-1$) is being flipped. Which S-boxes in which rounds are affected by the bit flip during DES encryption?
2. Which S-boxes in which DES rounds are affected by this bit flip during DES decryption?

3.7. A DES key K_w is called a *weak key* if encryption and decryption are identical operations:

$$\mathrm{DES}_{K_w}(x) = \mathrm{DES}^{-1}_{K_w}(x), \text{ for all } x \tag{3.1}$$

1. Describe the relationship of the subkeys in the encryption and decryption algorithm that is required so that Eq. (3.1) is fulfilled.
2. There are four weak DES keys. What are they?
3. What is the likelihood that a randomly selected key is weak?

3.8. DES has a somewhat surprising property related to bitwise complements of its inputs and outputs. We investigate the property in this problem.

We denote the bitwise complement of a number A (that is, all bits are flipped) by A'. Let $\oplus$ denote bitwise XOR. We want to show that if

$$y = \mathrm{DES}_k(x)$$

then

$$y' = \mathrm{DES}_{k'}(x'). \tag{3.2}$$

This states that if we complement the plaintext and the key, then the ciphertext output will also be the complement of the original ciphertext. Your task is to *prove* this property.

Try to prove this property using the following steps:

1. Show that for any bit strings A, B of equal length,

$$A' \oplus B' = A \oplus B$$

and

$$A' \oplus B = (A \oplus B)'.$$

(These two operations are needed for some of the following steps.)

2. Show that $PC-1(k') = (PC-1(k))'$.
3. Show that $LS_i(C'_{i-1}) = (LS_i(C_{i-1}))'$.
4. Using the two results from above, show that if k_i are the keys generated from k, then k'_i are the keys generated from k', where $i = 1, 2, \ldots, 16$.
5. Show that $IP(x') = (IP(x))'$.
6. Show that $E(R'_i) = (E(R_i))'$.
7. Using all previous results, show that if R_{i-1}, L_{i-1}, k_i generate R_i, then R'_{i-1}, L'_{i-1}, k'_i generate R'_i.
8. Show that Eq. (3.2) is true.

3.9. Assume we perform a known-plaintext attack against DES with one pair of plaintext and ciphertext. How many keys do we have to test in a worst-case scenario if we apply an exhaustive key search in a straightforward way? How many on average?

3.10. In this problem we want to study the clock frequency requirements for a hardware implementation of DES in real-world applications. The speed of a DES implementation is mainly determined by the time required to do one core iteration. This hardware kernel is then used 16 consecutive times in order to generate the encrypted output. (An alternative approach would be to build a hardware pipeline with 16 stages, resulting in 16-fold increased hardware costs.)

1. Let's assume that one core iteration can be performed in one clock cycle. Develop an expression for the required clock frequency for encrypting a stream of data with a data rate r [bit/sec]. Ignore the time needed for the initial and final permutation.
2. What clock frequency is required for encrypting a fast network link running at a speed of 1 Gb/sec? What is the clock frequency if we want to support a speed of 8 Gb/sec?

3.11. As the example of COPACOBANA [105] shows, key-search machines need not be prohibitive from a monetary point of view. We now consider a simple brute-force attack on DES which runs on COPACOBANA.

1. Compute the runtime of an average exhaustive key-search on DES assuming the following implementational details:
   - COPACOBANA platform with 20 FPGA modules
   - 6 FPGAs per FPGA module
   - 4 DES engines per FPGA
   - Each DES engine is fully pipelined and is capable of performing one encryption per clock cycle
   - 100 MHz clock frequency
2. How many COPACOBANA machines do we need in the case of an average search time of one hour?
3. Why does any design of a key-search machine constitute only an upper security threshold? By *upper security threshold* we mean a (complexity) measure which describes the maximum security that is provided by a given cryptographic algorithm.

3.12. We study a real-world case in this problem. A commercial file encryption program from the early 1990s used standard DES with 56 key bits. In those days, performing an exhaustive key search was considerably harder than nowadays, and thus the key length was sufficient for some applications. Unfortunately, the implementation of the key generation was flawed, which we are going to analyze. Assume that we can test 10^6 keys per second on a conventional PC.

The key is generated from a password consisting of 8 characters. The key is a simple concatenation of the 8 ASCII characters, yielding $64 = 8 \cdot 8$ key bits. With the permutation $PC-1$ in the key schedule, the least significant bit (LSB) of each 8-bit character is ignored, yielding 56 key bits.

1. What is the size of the key space if all 8 characters are randomly chosen 8-bit ASCII characters? How long does an average key search take with a single PC?
2. How many key bits are used, if the 8 characters are randomly chosen 7-bit ASCII characters (i.e., the most significant bit is always zero)? How long does an average key search take with a single PC?
3. How large is the key space if, in addition to the restriction in Part 2, only letters are used as characters. Furthermore, unfortunately, all letters are converted

to capital letters before generating the key in the software. How long does an average key search take with a single PC?

3.13. This problem deals with the lightweight cipher PRESENT.

1. Calculate the state of PRESENT-80 after the execution of one round. You can use the following table to solve this problem with paper and pencil. Use the following values (in hexadecimal notation):
   plaintext = `0000 0000 0000 0000`,
   key = `BBBB 5555 5555 EEEE FFFF`.

Plaintext	`0000 0000 0000 0000`
Round key	
State after KeyAdd	
State after S-Layer	
State after P-Layer	

2. Now calculate the round key for the second round using the following table.

Key	`BBBB 5555 5555 EEEE FFFF`
Key state after rotation	
Key state after S-box	
Key state after CounterAdd	
Round key for Round 2	