NETWORK SECURITY

Data Encryption Standard

Dr. Faheem Masoodi
masoodifahim@uok.edu.in

Data Encryption Standard (DES)

DES is a cipher which encrypts blocks of length of 64 bits with a key of size of 56 bits (Fig 1)

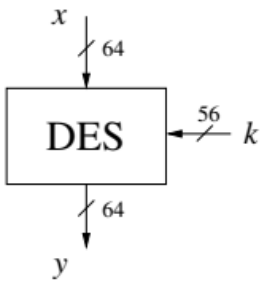

Fig. 1 DES Block cipher

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 *ki* are derived from the main key *k*.

Let's now have a more detailed view on the internals of DES, as shown in Fig. 2.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

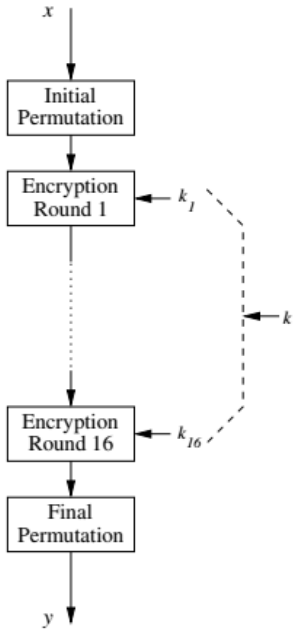

Fig. 2 Iterative structure of DES

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 *Ri* 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 *Li*. Finally, the right and left half are swapped. This process repeats in the next round and can be expressed as:

$$Li = Ri\text{-}1,$$
$$Ri = Li\text{-}1 \oplus f(Ri\text{-}1, ki)$$

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 *ki* 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 *Ri*-1 and *ki*. The output of the pseudorandom generator is then used to encrypt the left half *Li*-1 with an XOR operation. If the output of the *f* function is not predictable for an attacker, this results in a strong encryption method.

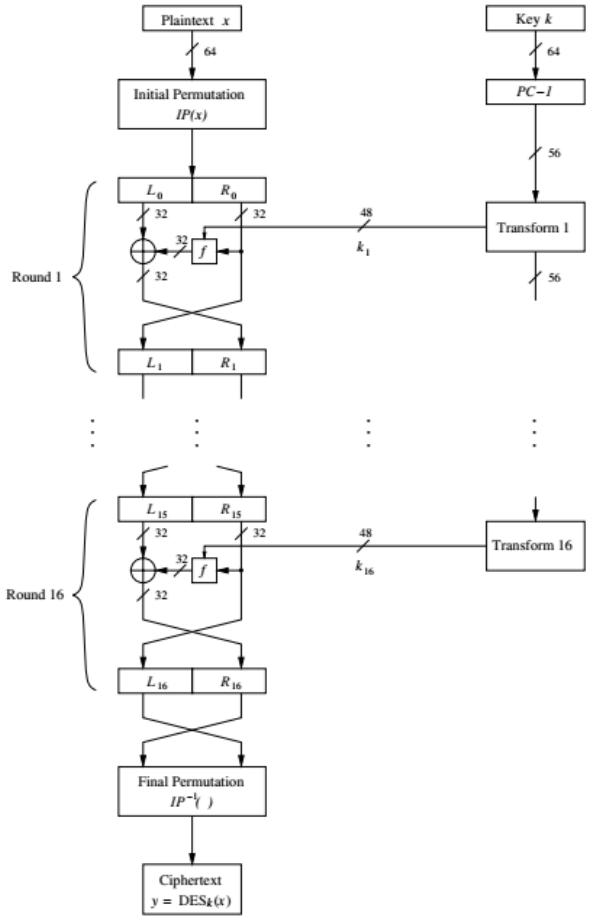

Fig. 3 The Feistel structure of DES

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

Internal Structure of DES

The structure of DES as depicted in Fig. 3 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.

Initial and Final Permutation

As shown in Figs. 4 and 5 , 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

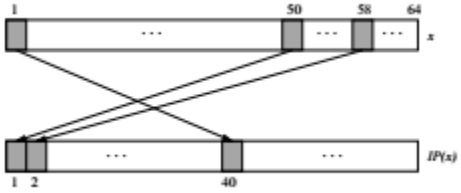

Fig. 4 Examples for the bit swaps of the initial permutation

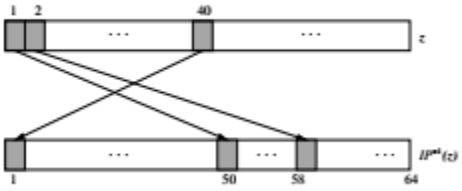

Fig. 5 Examples for the bit swaps of the final permutation

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}.

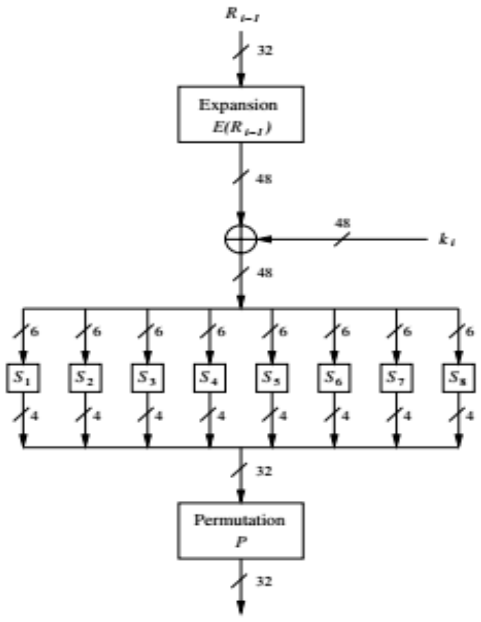

Fig. 6 Block Diagram of the *f function*

The structure of the *f* -function is shown in Fig. 6. 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 7.

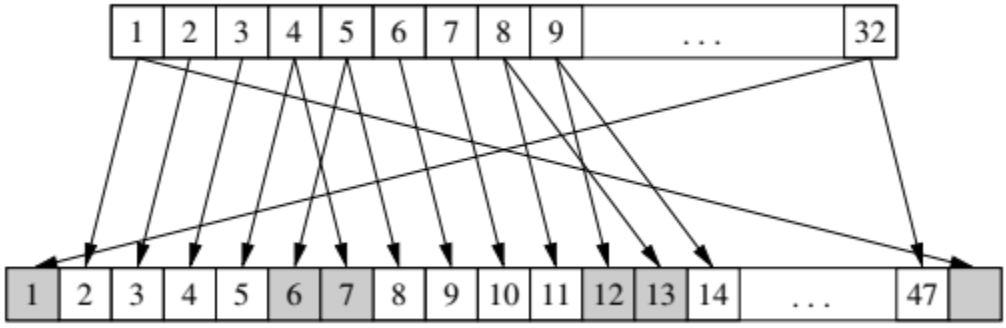

Fig 8 Examples for the bit swaps of the expansion function *E*

Next, the 48-bit result of the expansion is XORed with the round key *ki*, 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. Note that all S-boxes are different. The tables are to be read as indicated in Fig. 9: 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. 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).

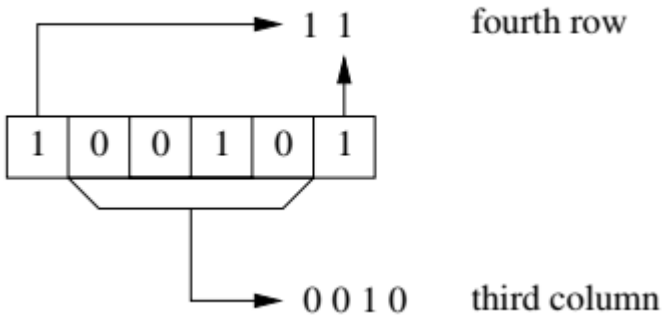

Fig. 9 Example of the decoding of the input 100101_2

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 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. 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. 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 3.10. 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*.

Key Schedule

The *key schedule* derives 16 round keys *ki*, 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.10, 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 1.

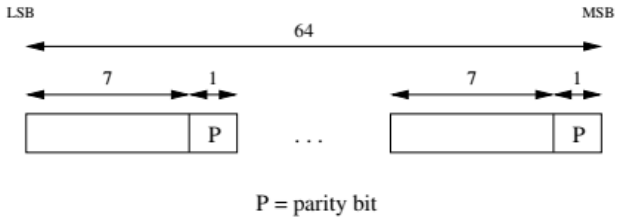

Fig. 10 Location of the eight parity bits for a 64-bit input key

Table 1. 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. 11. 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* = 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 4·1+12·2 = 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.

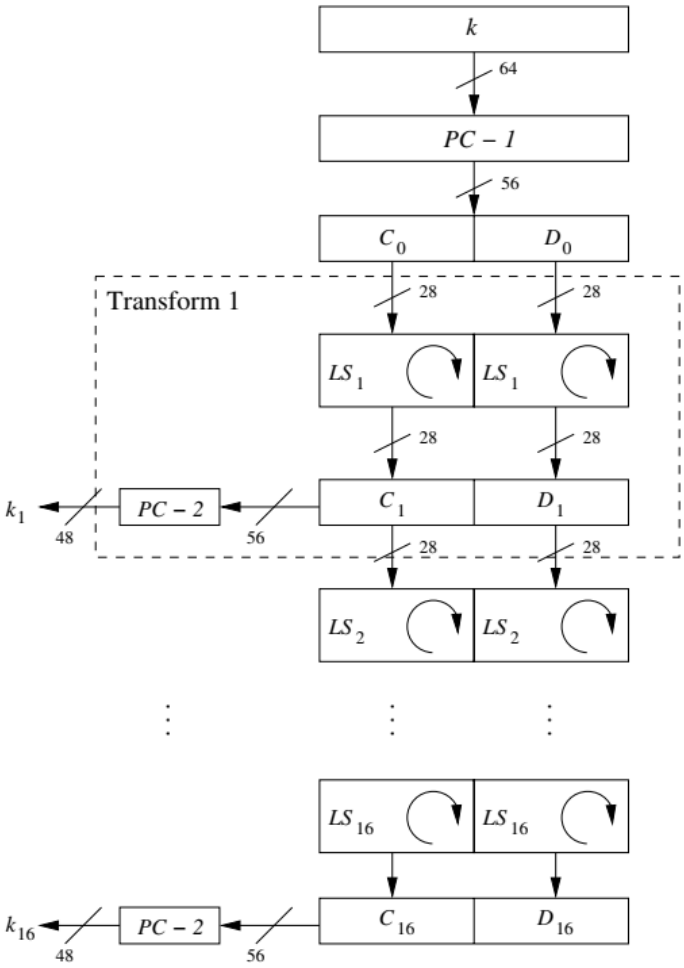

Fig. 11: Key schedule for DES encryption

To derive the 48-bit round keys *ki*, 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 *Ci* and *Di* and ignores 8 of them. The exact bit-connections of *PC* -2 are given in Table 2

Table 2 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.

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 12 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 $k16, k15, \ldots, k1$.

Reversed Key Schedule

The first question that we have to clarify is how, given the initial DES key *k*, can we easily generate $k16$? 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 $k14, k_{13}, \ldots, k_1$ 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

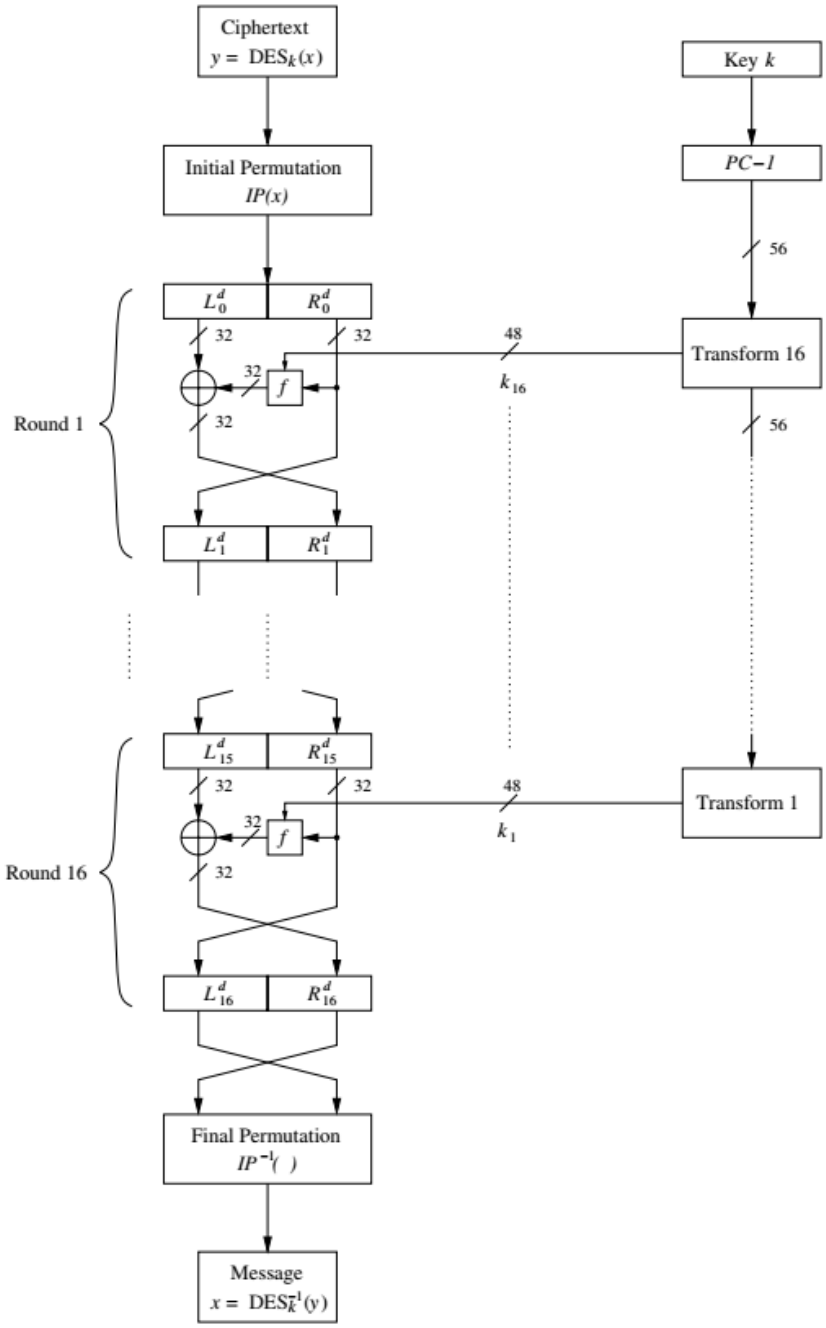

Fig. 12: DES decryption

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.** 12. 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:

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

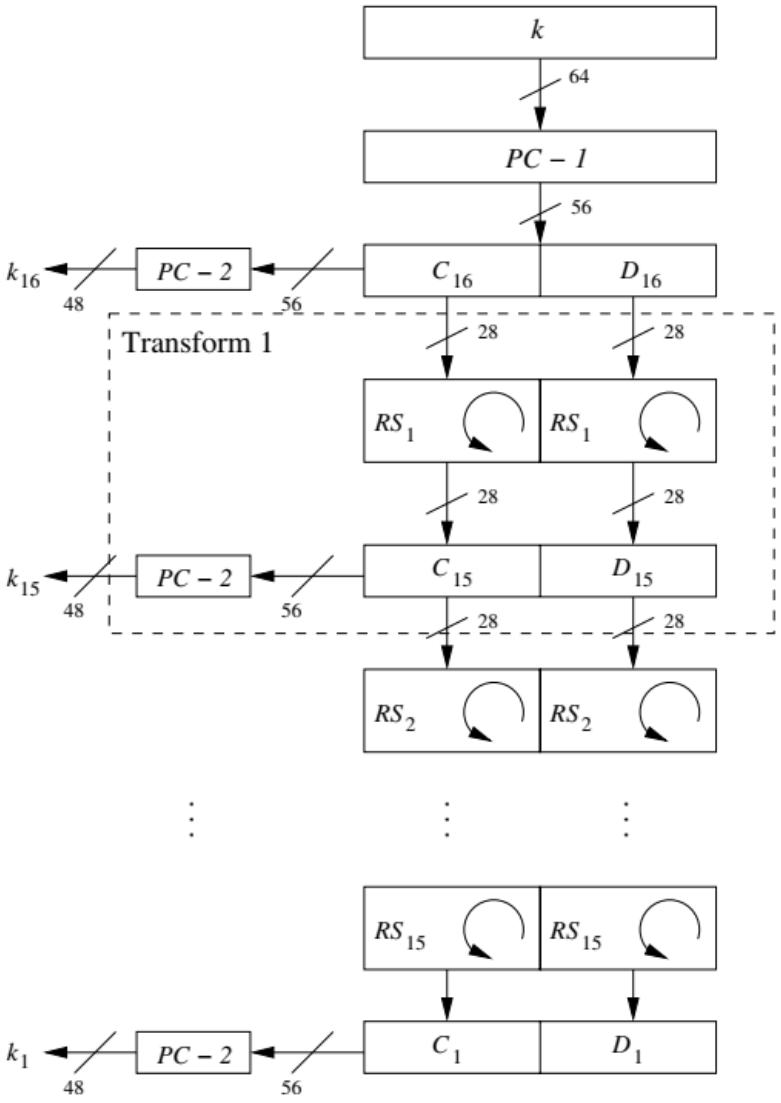

Fig. 12: Reversed Key schedule for DES decryption

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.