Handout 2

Summary of this handout: *Symmetric Ciphers Overview — Block Ciphers — Feistel Ciphers — DES*

II. Symmetric Ciphers

18. Symmetric Ciphers

In a symmetric cipher the same key is used to both encrypt and decrypt a message. Therefore, both sender and receiver have to have knowledge of that key for encryption and decryption. Sometimes the keys are not exactly the same, but only trivially related. For instance, in the permutation cipher we can view the permutation as the encryption key and its inverse permutation as its trivially related decryption key.

Symmetric ciphers are the classic variant of cryptographic algorithms, as opposed to asymmetric ciphers, in which both sender and receiver use different keys. We will learn about asymmetric ciphers later.

Symmetric ciphers can be divided into two main types:

Block Cipher A symmetric key cipher, which operates on fixed-length groups of bits, named *blocks*.

Stream Cipher A symmetric cipher that encrypts plaintext continuously. Digits are enciphered one at a time and the transformation of successive digits varies during the encryption.

19. Problems with Symmetric Ciphers

There are a number of obvious problems with symmetric ciphers. Since all parties involved in the communication have to use the same key there need to be secure ways of distributing the key and keeping it secret. To guarantee continuous secure communication keys have to be changed often and therefore new, non-trivial keys have to be generated. All these problems are known as *key management* problems and we will touch on them at the end of this section.

Another drawback is that symmetric-key algorithms can not be used to authenticate the sender of a message. This is a problem we will get back to towards the end of this term.

Before we have a closer look at block ciphers we will first define some terminology that we will use throughout the course.

20. The Players

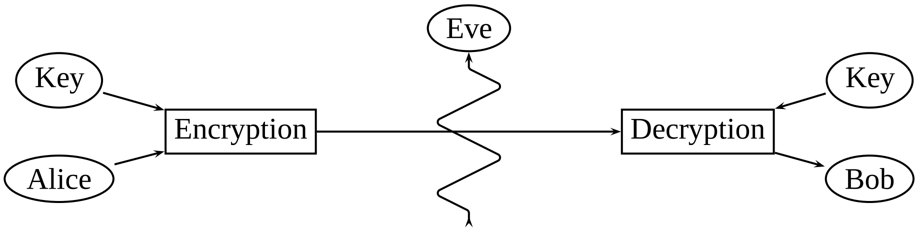
We will name our three main players in the game of cryptology:

Alice The sender of an encrypted message.

Bob The intended receiver of an encrypted message. Bob is assumed to have the key to decrypt it.

Eve The eavesdropper who tries to intercept and to cryptanalyse messages passed between Bob and Alice.

These three names are used throughout the literature to illustrate cryptographic algorithms and protocols. They are invariably supplemented by other players to mark additional participants in multi-party communication (names with ‘C’ and ‘D’) or additional attackers, arbitrators, trusted third parties, etc.

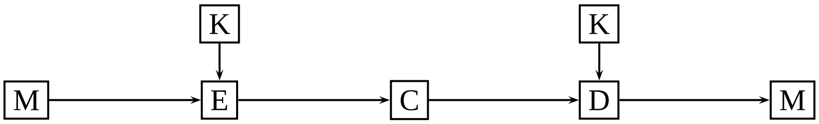


21. Mathematical Notation

For a more abstract depiction of the encryption and decryption process, let

- M be the plaintext,
- K be the secret key,
- E be the encryption function,
- D be the decryption function,
- C be the ciphertext.

We can then simplify the above diagram.



The encryption function E can be seen either as a binary function taking two arguments K and M or as a generic function which is customised by the key K. We will generally adopt that latter view both for E and D and express encryption and decryption as:

$$E_K(M) = C \qquad\qquad D_K(C) = M$$

The communication is performed under the constant thread that Eve is listening in! We have to assume that Eve is familiar with the particular cryptographic algorithm used by Alice and Bob, i.e., with the generic functions E and D. Thus the security of the communication depends on the cryptographic strength of the customised system E_K and D_K, such that it is impossible for Eve to (1) find the key K and to (2) find a function f such that $f(C) = M$.

II.1 Block Ciphers

We will first have a look at the basic building blocks for many modern block ciphers and then inspect two algorithms (DES and Rijndael) in detail.

II.1.1 Feistel Ciphers

The Feistel cipher is a basic block cipher, which was developed by Horst Feistel at IBM. Its particular structure forms the bases of many modern block ciphers. The first Feistel cipher patented was the *Lucifer Cipher* in 1971.

A Feistel cipher is a *product cipher* in that it applies the same basic encryption scheme iteratively for several rounds. It works on a block of bits of a set size and applies in each iteration a so called *round function*, i.e. an encryption function parameterised by a round key. Round keys are often derived from a general key and therefore called *sub-keys*. They are invoked in the encryption scheme by some function called a *Feistel function*. Each round of encryption works then as follows:

(i) Split the input in half.

(ii) Apply the Feistel function parameterised by the key to the right half.

(iii) Compute the xor of the result with the old left half to be the new left half.

(iv) Swap the old right and new left half, unless we are in the last round, where we do not swap.

In the following we will denote the xor operation on two bit blocks by $\oplus$.

Example: Consider the following xor operation on two four-bit blocks: $1010 \oplus 1100 = 0110$.

Observe that this operation corresponds to a bit-wise addition modulo 2 and that it is self-inverse.

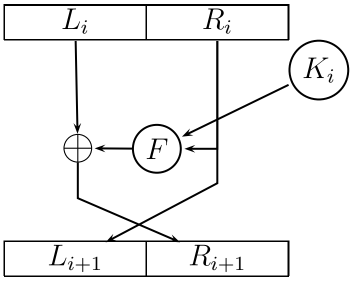
Example: $1010 \oplus 1100 = 0110$ and $0110 \oplus 1100 = 1010$.

22. Feistel Cipher Encryption Algorithm

We can formally define the encryption algorithm for an r-round Feistel cipher working on a plaintext M, with respect to a Feistel function F and round keys $K_0, \dots, K_{r-1}$ as

1. Split the plaintext block into two equal pieces, $M = (L_0, R_0)$
2. For each round $i = 0, 1, \dots, r-1$, compute
$$\begin{aligned} L_{i+1} &= R_i \\ R_{i+1} &= L_i \oplus F(K_i, R_i) \end{aligned}$$
3. Then the ciphertext is $C = (R_r, L_r)$. [Observe that this means we do not swap in the last round!]

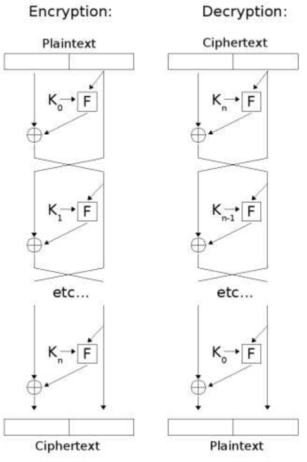
Step 2 of the algorithm (except for the last round where there is no swap) is graphically shown below:



The interesting property of the Feistel Cipher is that regardless of choice of the particular Feistel function F, the round function can be inverted. In fact the decryption algorithm works exactly as encryption, just with a reversed order of keys:

1. Split the ciphertext block into two equal pieces, $C = (R_r, L_r)$
   [Observe that we start with the ciphertext coming from the encryption, i.e. R and L are reversed!]
2. For each round $i = r, r-1, \dots, 1$, compute
$$\begin{aligned} R_{i-1} &= L_i \\ L_{i-1} &= R_i \oplus F(K_{i-1}, L_i) \end{aligned}$$
3. This results in the plaintext $M = (L_0, R_0)$. [Again, no swap in the last round!]

ere is an overview of the entire algorithm. For the decryption note that left and right hand sides of the ciphertext are swapped in from the beginning, i.e., except for the last round, the R_i parts are on the left and the L_i parts on the right. Observe in particular the last round of en-/decryption:



Feistel Cipher

Source: Wikipedia

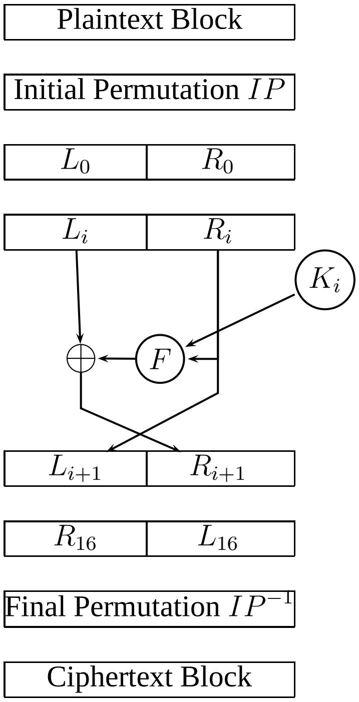
Ciphers can now be built from the basic Feistel cipher design (1) by specifying the generation of round keys, (2) by fixing the number of rounds, and (3) by defining the Feistel function F.

23. Some Feistel Ciphers are for instance Lucifer, Blowfish, Twofish, RC5, FEAL, DES, 3DES

II.1.2 DES

The Data Encryption Standard (DES) was one of the most widely applied block ciphers. It was designed by IBM in collaboration with the NSA and adopted as an official Federal Information Processing Standard (FIPS) for the United States in 1976 (FIPS PUB 46-3). There were rumours about backdoors the NSA had built into it, but until now no evidence was found for this.
DES has a fairly small key size and is therefore considered too weak today. Indeed the world record for breaking DES encryption is currently 10 hours. Nevertheless, we will study it here since it provides the basis for several variants of DES that still provide good security. Some examples of variants are Triple-DES (TDES), DES-X, or ICE.

24. Overview of the DES Algorithm
DES is a slightly modified Feistel cipher, in that it adds an initial permutation of the plaintext and a final permutation of the ciphertext to 16 rounds of Feistel encryption. The overview of the DES procedure is therefore:



The design parameters of the DES cipher are:

- Block length n is 64 bits.
- Number of rounds r is 16.
- Key lengths is 56 bits.
- Round keys length is 48 bits for each sub-key $K_0, \ldots, K_{15}$.
  The sub-keys are derived from the 56 bit key with a special *key schedule*.

The most important part of DES is of course its specialist Feistel Function F.

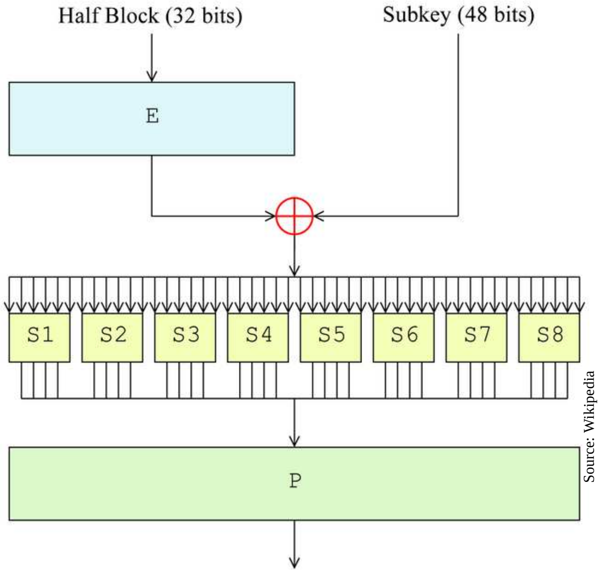
25. The DES Feistel Function
The Feistel function consists of four stage procedure:

1. **Expansion Permutation:** Expand the 32-bit message half block to a 48-bit block by doubling 16 bits and permuting them.
   [Observe that this permutation is different from the initial permutation IP!]
2. **Round Key Addition:** Compute the xor of the resulting 48-bit block with the round-key K_i.

3. **S-Box:** Split the 48-bit into eight 6-bit blocks. Each of those is then given as input to eight *Substitution Boxes* (*S-Boxes*), which substitute the 6-bit blocks by 4-bit blocks.

4. **P-Box:** The eight 4-bit output blocks from the S-Boxes are combined to a 32-bit block and permuted in the *Permutation Box* (*P-Box*) to result in the final output of the function F.

The following is an overview of the DES Feistel function:



Source: Wikipedia

26. Operations of DES

Before we examine DES step-by-step, there are three new operations we have to get familiar with:

1. Cyclic shifts on bit strings blocks.

2. Permutations in DES

3. S-Box substitutions.

1. Cyclic left shifts The idea of a cyclic left shift on a bit block is to move the bits of a block left by a constant number of positions and to add every bit that would have "fallen out" of the block on the left side on the right of the block. We denote the cyclic left shift operation by '$\lll$'.
More formally we can define $\lll$ as: Suppose B is a block of n bits, $B = b_1 b_2 \dots b_n$ and $0 \leq k \leq n$ then $B \lll k = b_{k+1} b_{k+2} \dots b_n b_1 \dots b_k$.
Example: Let $B = 01101001$ and $k = 3$ then $B \lll k = 01101001 \lll 3 = 01001011$.
Similarly we can define cyclic right shifts $\ggg$.

2. Permutations Permutations in DES are not necessarily permutations in the strictly mathematical sense as they might drop or duplicate bits. Moreover, they use a particular notation one has to get familiar with. Instead of using, for example, a cycle notation, DES permutations are denoted in a form that specifies the output order of the input bits. For example, the permutation $\boxed{4\ 1\ 2\ 3}$ means that

- the fourth input bit becomes the first output bit,
- the first input bit becomes the second output bit,
- the second input bit becomes the third output bit, and
- the third input bit becomes the fourth output bit.

Example: Suppose we apply $\boxed{4\ 1\ 2\ 3}$ to the bit block 0101 we then get the result 1010.
As mentioned before DES permutations are not necessarily one-to-one mappings. The can duplicate bits or they can drop bits or they can do both at the same time. Most importantly, the size of a bit block before a permutation is applied is not necessarily equal to the size of the resulting bit block. (This is also the reason why the standard mathematical notation is not appropriate for DES "permutations").
Example: Suppose we apply

- $\boxed{4\ 1\ 2\ 3}$ to the bit block 010101 we still get the result 1010. Notice that bits 5 and 6 have been dropped.
- $\boxed{4\ 1\ 2\ 3\ 1\ 4}$ to the bit block 0101 we get 101001. Notice that bits 1 and 4 have been duplicated.
- $\boxed{4\ 1\ 2\ 3\ 1\ 4}$ to the bit block 010101 we still get 101001. Notice that bits 5 and 6 have been dropped, bits 1 and 4 have been duplicated, and that we therefore get an output block of the same length as the input block.

3. S-Box Substitutions An S-Box substitution is essentially a table look-up. In DES S-Box substitutions work on 6 bit input blocks, yielding 4 bit output blocks. The basic idea is to use the input block to compute the row and column of the S-Box to look-up the output block.
In detail this works as follows: First the outer bits of the 6 bit input are stripped of and joined. This results in a two bit number, which as the row number for the table look-up, while the four inner bits, i.e. bits $2, 3, 4, 5$, are used as column number. The entry in the corresponding S-Box cell is then the resulting substitution.
As an example S-Box we first have a look at the table below (which corresponds DES's S-box S_5) in bit notation:

		Middle 4 bits of input															
	S_5	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Outer bits	00	0010	1100	0100	0001	0111	1100	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1100	0011	1001	1000	0110
	10	0100	0010	0001	1011	1100	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1100	0100	0101	0011

Example: Suppose we feed 101100 into S-Box S_5. We first take bits 1 and 6 and get 10 as row number. This leaves the inner bits 0110 as column number. The correct substitution is then 0111.
To preserve space S-Boxes are generally given in integer notation. For example the S-Box S_5 from above can also be written as:

S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Observe that we have also omitted to enumerate rows and columns as they are implicit. Note, however, that similar to the binary representation of the S-Box we enumerate rows from 0 to 3 and columns from 0 to 15, i.e., column 0 consists of entries $2, 14, 4, 11$ and row 0 consists of $2, 12, 3, 1, \ldots, 9$. We can now perform S-Box substitutions of 6 bit integers from 0 to 63 by 4 bit integers 0 to 15.
Example: Suppose we feed 47 into S_5 we first translate 47 into its binary representation 101111. We then compute the row and column for our look-up, which corresponds to 11 and 0111, i.e. row 3 and column 7. Thus the result of our substitution is 13.

27. Steps of DES

We now examine the single steps of DES.

1. The Initial Permutation IP is given in the following table. It permutes the 64 bits of the input, i.e. the plaintext. IP is given in a form that specifies the output order of the 64 input bits, e.g., the 58 in the first position means that the first bit of the output is the 58^{th} bit of the input.

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

After the application of IP the resulting 64 bit are split into the initial left and right half L_0 and R_0. R_0 is then passed on into the Feistel function, starting with the expansion permutation E.

2. The Expansion Permutation E is, strictly speaking, not a permutation in the mathematical sense, since it does not only permute bits but also duplicates them (thus it is not a one-to-one mapping). The table below specifies E in the same notation as IP above. E moves all bits and duplicates half of them. For example, input bit 1 is mapped to output bit 2 and output bit 48, whereas input bit 2 is only mapped to output bit 3.

E :	32	1	2	3	4	5	$\longrightarrow S_1$
	4	5	6	7	8	9	$\longrightarrow S_2$
	8	9	10	11	12	13	$\longrightarrow S_3$
	12	13	14	15	16	17	$\longrightarrow S_4$
	16	17	18	19	20	21	$\longrightarrow S_5$
	20	21	22	23	24	25	$\longrightarrow S_6$
	24	25	26	27	28	29	$\longrightarrow S_7$
	28	29	30	31	32	1	$\longrightarrow S_8$

We can observe that every row in E overlaps in two (input) bits with the row directly above and directly below it. This has the effect that one bit of input affects two substitutions performed in the S-Boxes, which means that only a small change in the plaintext produces a large difference in the ciphertext. Each row in the output of E corresponds to an input into the indicated S-boxes. However, before the substitutions are performed in step 4, the 48-bit round key K_i is xor-ed with the result of E.

3. Xor-ing the round key K_i At this point we have to take a closer look at how the round keys are computed from the original 56-bit key. The function to determine the sub-keys is called a *key schedule*.
The key K is actually given as a 64-bit key, where each 8^{th} bit is a parity bit. In a first step the parity bits are stripped and the remaining 56-bits are permuted with respect to the permutation PC-1 below.

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

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

The result of PC-1 is divided into a 28-bit left half C_0 and a 28-bit right half D_0. Now for each round we compute

$$\begin{aligned} C_i &= C_{i-1} \lll p_i \\ D_i &= D_{i-1} \lll p_i \end{aligned}$$

where $x \lll p_i$ means the cyclic shift on x to the left by p_i positions with $p_i = \begin{cases} 1 & \text{if } i = 1, 2, 9, 16 \\ 2 & \text{otherwise} \end{cases}$

For example 10101101 $\lll$ 2 $=$ 10110110.
C_i and D_i are then joined together again and permuted with PC-2 above. Observe, that PC-2 is again not a permutation in the strict sense since it drops some of the input bits, for instance bit 9 and 18 and thereby produces the final 48 bit round key.

4. S-Boxes Once the round key has been xor-ed to the output of the expansion permutation each 6 bit row is fed into an S-Box to be substituted. DES uses eight different S-Boxes, which are given below in decimal notation:

S_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

The S-Box substitution is the important part that provides security of DES. In fact, the composition of the S-Boxes is crucial and even only small changes to the substitution schemes can reduce the security of DES drastically. The correct eight S-Boxes are given on the next page.

The output of the eight S-Box substitution are eight 4-bit numbers, which are passed on to the P-Box.

5. P-Box The P-Box takes the eight 4-bit pieces and combines them using the following permutation scheme.

$P:$	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

This results again in 32 bits, which are xor-ed with the 32 bits of the original left half of the input text. Unless we are in the last round of the algorithm the new right half and the original left half are swapped and concatenated.

<u>*6. Final Permutation* IP^{-1}</u> Once steps 2–5 have been iterated for 16 rounds the resulting 64 bits are again permuted with respect to the inverse of the initial permutation:

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

28. An Example Round

Let's have a look at an example round of the DES cipher. We take a very simple message and a nearly trivial key:
$M = 0000000000000000000000000000000100000000000000000000000000000001$ and
$K = 1000000000000000000000000000000010000000000000000000000000000000$
The first round key is the computed as follows:
$PC\text{-}1(K) = PC\text{-}1(1000000000000000000000000000000010000000000000000000000000000000) =$
$00010001000000000000000000000000000000000000000000000000$
$C_1 = C_0 \lll 1 = 0001000100000000000000000000 \lll 1 = 0010001000000000000000000000$
$D_1 = D_0 \lll 1 = 0000000000000000000000000000 \lll 1 = 0000000000000000000000000000$
$PC\text{-}2(C_1\|D_1) = PC\text{-}2(00100010000000000000000000000000000000000000000000000000) =$
$000000100000000000100000000000000000000000000000$

The following is the first round of DES (observe that we start with round 1 as opposed to round 0 in the usual Feistel ciphers):

Round 1: $L_1 = 00000000000000000000000000000001$ $R_1 = 00000000000000000000000000000001$
Apply E: $100000000000000000000000000000000000000000000010$
Xor K_1: $000000100000000000100000000000000000000000000000 \oplus$
$100000000000000000000000000000000000000000000010 =$
$100000\|100000\|000000\|010000\|000000\|000000\|000000\|000010$
S-Box S_1: 0100 S-Box S_2: 0000 S-Box S_3: 1010 S-Box S_4: 0001
S-Box S_5: 0010 S-Box S_6: 1100 S-Box S_7: 0100 S-Box S_8: 0010
P-Box: $10010000000100101000000110001100$
Xor L_1: $10010000000100101000000110001100 \oplus 00000000000000000000000000000001 =$
$10010000000100101000000110001101$
$R_1\|L_1 = 00000000000000000000000000000001\|10010000000100101000000110001101$

The remaining 15 rounds are left as an exercise. The final resulting ciphertext is:

$$1010010101001001101011110101011101010110111010110010100010000111$$

We can observe the *avalanche effect*, i.e., a small change of the plaintext triggers a big change in the ciphertext, already after the first round. Suppose we change the message ever so slightly to
$M = 0000000000000000000000000000000100000000000000000000000000000000$ and retain the key as
$K = 1000000000000000000000000000000010000000000000000000000000000000$
Then result after the first round is
$R_1\|L_1 = 00000000000000000000000000000000\|10011000100100001000101110101101$
which is already (at least the right half) significantly different from the first round result of the previous encryption.
This is even more apparent when comparing the final result of the encryption:

$$1011110101001011010000110011101010010000001101111000011011110100$$

with the previous one above. It is obvious that despite having chosen fairly trivial messages and trivial keys, the difference of only one bit in the original message leads to significant a difference in the ciphertexts, which makes it not obvious that the original plaintexts are actually very similar.

Cryptography Glossary 2

Alice	The sender of an encrypted message.	12
Asymmetric Cipher	Cipher that uses different (not trivially related) keys for encryption and decryption.	12
Avalanche Effect	The property of a cipher where a small change of the plaintext triggers a big change in the ciphertext.	20
Block	A fixed-length group of bits.	12
Block Cipher	A symmetric key cipher, which operates on fixed-length groups of bits, named *blocks*.	12
Bob	The intended receiver of an encrypted message. Bob is assumed to have the key to decrypt it.	12
Eve	The eavesdropper who tries to intercept and to cryptanalyse messages passed between Bob and Alice.	12
Feistel Cipher	A basic block cipher developed by Horst Feistel at IBM. It forms the bases of many modern block ciphers.	13
Feistel function	The function applying the round key in the Feistel cipher, thereby effectively parameterising the round function.	13
Key management	Problems related to choosing, distributing, and securely storing keys for symmetric ciphers.	12
Key Schedule	A function to generate round keys from one input key.	15, 18
Lucifer Cipher	First simple Feistel cipher.	13
P-Box	Short for Permutation Box.	16
Permutation Box	An operation that takes a set of input blocks, combines them and applies a permutation.	16
Product cipher	A cipher that applies the same basic encryption scheme iteratively for several rounds.	13
Round function	The parameterised encryption function applied during one round of a Feistel cipher.	13
S-Box	Short for Substitution Box.	15
Stream Cipher	A symmetric cipher that encrypts plaintext continuously.	12
Sub-key	A round key derived from one key valid for the entire algorithm.	13
Substitution Box	A look-up table that substitutes a 4-bit block for a 6-bit block.	15
Symmetric Cipher	Ciphers that use the same or trivially related keys for encryption and decryption.	12