Systems and Network Security (NETW-1002)

Dr. Mohamed Abdelwahab Saleh
GUC, IET-Networks Department

Spring 2017

1 Data Encryption Standard

The Data Encryption Standard (DES) is a standard encryption system that uses a combination of substitutions and permutations to encrypt data. The key in DES is XOR'ed with some of the data before the substitutions. The DES encryption system has a fixed-size key of 56 bits. Actually, the key is 64 bits and 8 of these are used for parity check. Also, DES operates on blocks of data whose size is fixed at 64 bits. The general view of the algorithm of DES is depiced in Figure 1.

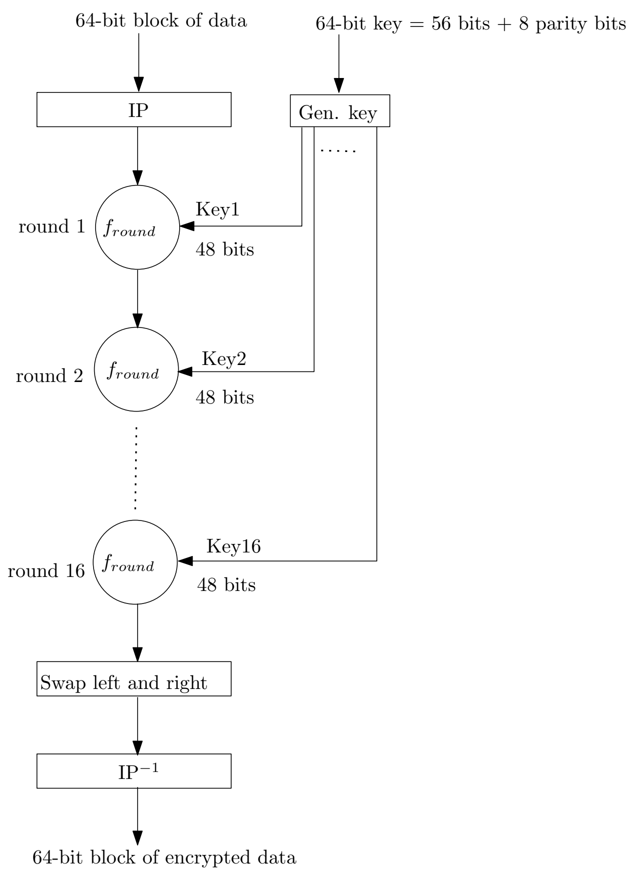


Figure 1: An overview of DES.

From Figure 1, we notice the following main blocks:

- Gen. key: This an algorithm that generates 16 keys, each one 48-bit long, from the original 56-bit key.
- IP: This a permutation block, it permutes the bits according to a standard fixed table [2].
- f_{round}: This the encryption function that is run 16 times, it accepts a 64-bit block of input data and a 48-bit key and generates a 64-bit block of data.
- Swap left and right: This is a permutation that splits the 64-bit data block into two halves: Left and right, and swaps these two, i.e., the input is LR and the output is RL. Both L and R are 32-bit blocks.
- IP^{-1}: This a permutation block, it permutes the bits according to a standard fixed table [2].

The tables for both IP and IP^{-1} can be found in any standard book on cryptography [2]. They are given here for completeness. For instance, in the permutation of table "IP", bit 58 replaces bit 1 and bit 50 repalces bit 2, and so on.

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

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

In the following, we explain the key generation and encryption function blocks.

1.1 Genaration of Keys

The encryption key K in DES is 64-bits long (56 bits + 8 parity bits), and it is used to generate 16 keys, each is 48-bits long. This is done by the following steps:

- The key K is permuted, while discarding parity bits, to get K', we call this permutation P_K.
- K' is split into in two halves C_0 and D_0, each half is 28-bits long.
- Both C_0 and D_0 are rotated one bit to the left to get C_1 and D_1, respectively.
- We combine C_1 and D_1 and get 56 bits out of which we choose the 48-bit key K_1. This choice is done using a table defined in the DES standrard [2] and given below. We call it the key choice table.
- Each key K_i, $i > 1$ of the remaining 15 keys is obtained by repeating the last two steps starting with C_{i-1} and D_{i-1}. However, we note the following:
  - In the generation of keys K_1, K_2, K_9, and K_{16}, we apply a one-bit left rotation. In all other keys, we apply a rotation by *two* bits to the left.

So, For instance, the key K_2 is obtained by starting with C_1 and D_1, rotating each one one bit to the left to get C_2 and D_2 and finally choosing 48 bits from C_2 and D_2 after we combine them. These operations are depicted in Figure 2.

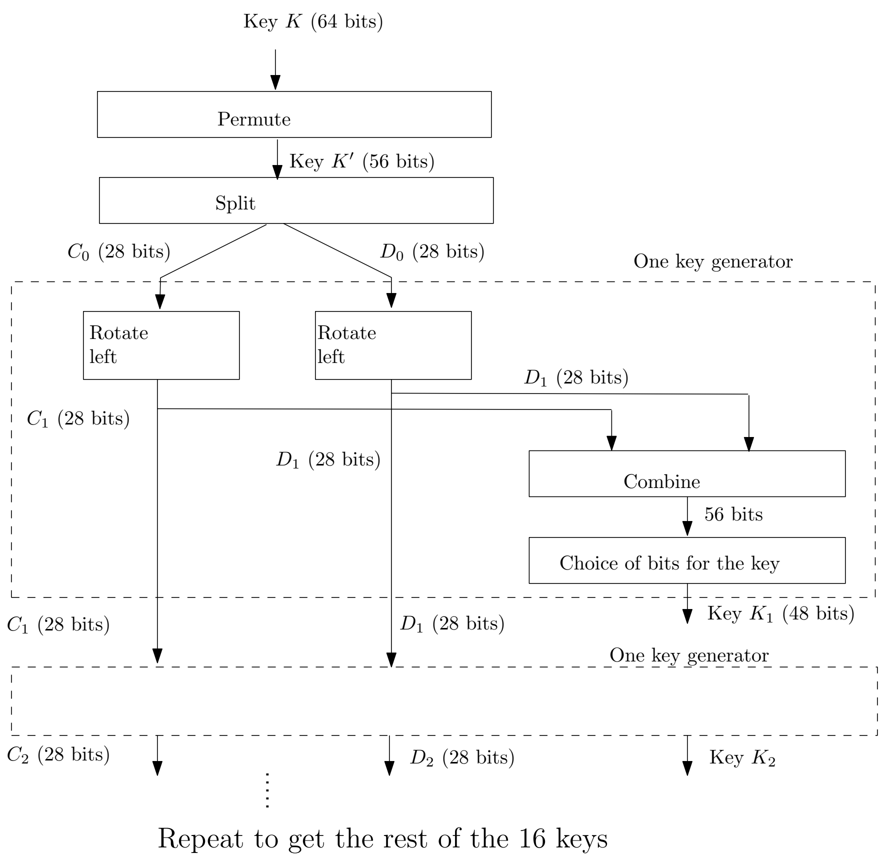


Figure 2: Key generation.

The tables used for key permutation, and the choice of key bits are given below:

P_K

57	49	41	33	25	17	9	C_0
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	D_0
7	62	54	46	38	30	22	
14	6	61	53	45	37	29	
21	13	5	28	20	12	4	

Key choice

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

1.2 Encryption Function

As shown in Figure 1, the encryption function f_{round} accepts a 48-bit key and a 64-bit data block and generates a 64-bit data output. The function is called 16 times (rounds), each time with a different key and, as input, the output of the previous function run. At round n, of the 16 rounds, the function splits its input data into two halves: Left (L_n) and right (R_n) and the following occurs:

- R_n becomes the left part L_{n+1} of the output.
- R_n is input, along with key K_n to a function f' that produces output R'_n.
- R'_n is XOR'ed with L_n to produce R_{n+1}, the right part of the output.

The function f' is explained in the next section. The encryption function is depicted in Figure 3.

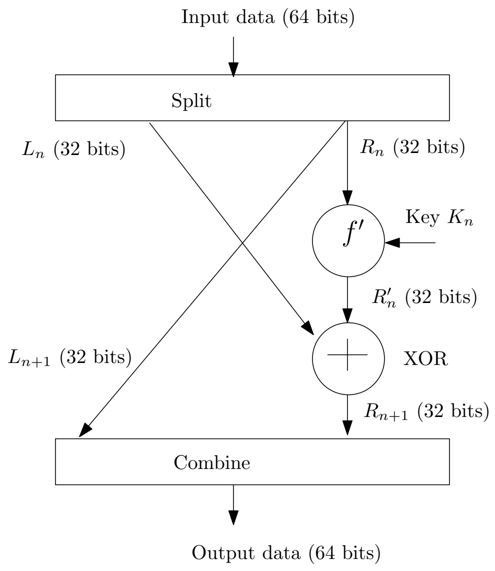


Figure 3: Encryption function.

1.2.1 The Function f'

The function f' is executed once at each round n. It takes as input data the 32-bit R_n and the 48-bit key K_n, where $1 \leq n \leq 16$. It generates a 32-bit output R'_n that gets XOR'ed with L_n. The function operates as follows:

- R_n is expanded into 48 bits according to a table in the DES standard [2], we call it the expansion table.
- The expanded R_n is XOR'ed with K_n.
- The result of the XOR operation is split into 8 blocks, each one is 6-bits long.
- Each 6-bit block B_i goes through a substituiton function S_i, called an S-box. There are 8 substitution tables defined in DES, one for each S_i.
- The output of each S_i is a 4-bit block O_i.
- All the outputs O_i are combined to get a 32-bit output.

- The 32-bit output is permuted by a permutation function P. The permutation table "P" is part of DES and is given below.

These operations are depicted in Figure 4. The expansion table, substituion tables for S-boxes and permutation table P are all given in the following.

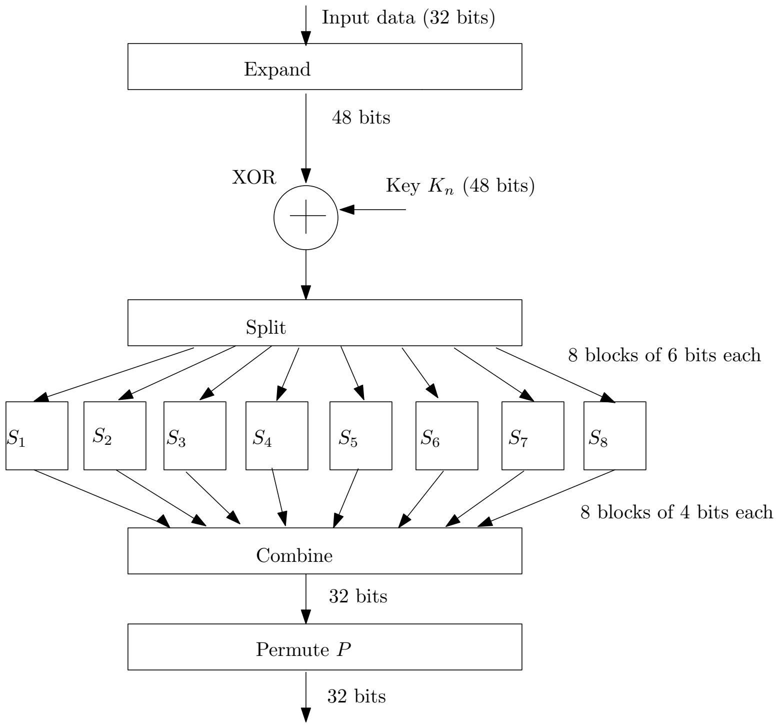


Figure 4: The function f'.

Expansion table

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

The substituion tables for S-boxes are given below. The way to use these tables is as follows:

- We start by a block of six bits: $b_1.b_2.b_3.b_4.b_5.b_6$
- The value of $b_1.b_6$, in decimal, determine a row r in the table.
- The value of $b_2.b_3.b_4.b_5$, in decimal, determine a column c in the table.
- The output of the S-box is the binary value of the table cell at row r and column c.
- As an example, consider an input 110100 to table S_1. In this case, $r = 10$ in binary which is 2 in decimal, and $c = 1010$ in binary, which is 10 in decimal. At row 2 and column 10, the value is 9, so the output bits are 1001.

S_1

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

S_2

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

S_3

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

S_4

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

S_5

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

S_6

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

S_7

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

S_8

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

The permutation table used to permute the combined output bits from the S-boxes is given below:

Permutation table 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

2 DES Modes of Operation

The Data Encryption Standard (DES) specifies that the input data is a 64-bit block that is encrypted by a 64-bit key. Actually, the key is 56-bits long and 8 bits are used for parity checking. Usually however, we need to encrypt data that are larger than 64 bits. For instance, using an 8-bit per character encoding, 64 bits constitute just eight characters. Most messages are larger than this. So, to encrypt data larger than 64 bits, we need to split it into 64-bit blocks, process the blocks with the key, and, then, combine the encrypted output to get the cipher text. This is shown in Figure 5

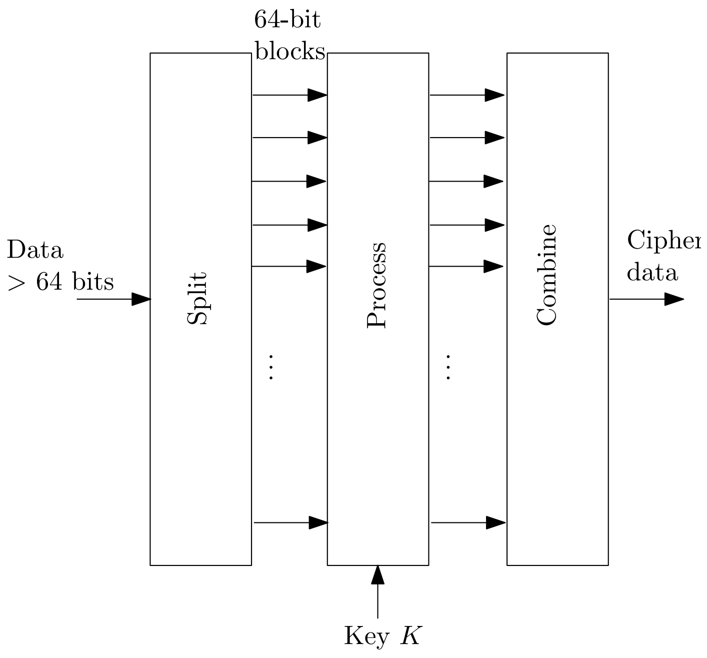


Figure 5: DES encryption of data larger than 64 bits.

There exist several methods for processing the split data, i.e., the 64-bit blocks. Each method defines a *mode of operation.*

2.1 Electronic Code Book Mode

In the Electronic Code Book (ECB) mode, data to be encrypted is split into blocks of 64 bits each, each block is encrypted with the encryption key to produce an encrypted block. Finally, all encrypted blocks are combined again to form the cipher text. The system schematic is shown in figure 6.

The most obvious disadvantage in this mode is that if two input, i.e., plain text, blocks are similar, the two corresponding output blocks will also be similar. This means that repeating patterns in the plain data will be transformed to repeating patterns in the cipher data.

In general, the length of a large chunk of data to be encrypted may not be a multiple of 64 bits. Therefore, when splitting the data into 64-bit blocks, we will obtain one data block,i.e., the last one, that is less than

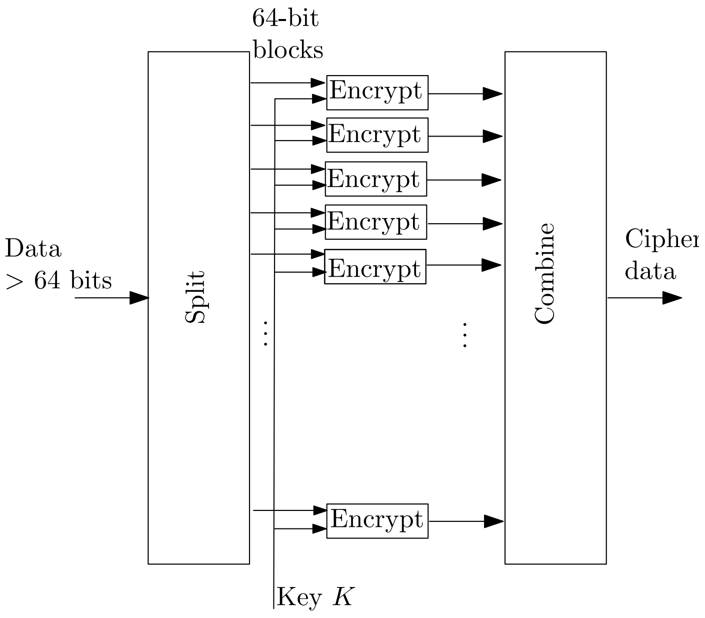


Figure 6: ECB mode of operation.

64 bits long. In order to be able to apply the DES encryption algorithm to this block we need to *pad* it with extra bits. The addition of these bits will bring the block size to 64 bits, however, during decryption, the original data should be recovered. So, there should be a way to know what extra bits were added, especially that the receiver of the encrypted data does not know the original bits in the message.

2.1.1 Padding

There exists several methods for message padding [1]. The most common of which is to add zeros at the end of the last block and store the number of added bytes as the last byte in the new 64-bit block. At the receiver side, reading the last byte of the decrypted data, one would now how many bytes to remove from the data in order to obtain the original message. Here, a confusion may arise in case no padding bytes were added, since, in this case, the last byte of the decrypted data will be an original byte of the message. The solution is to *always* add padding bytes, even if the message length is a multiple of 64 bits. Therefore, in the case of DES, the number of padding bytes will range from one to eight. Examples are shown in Figure 7.

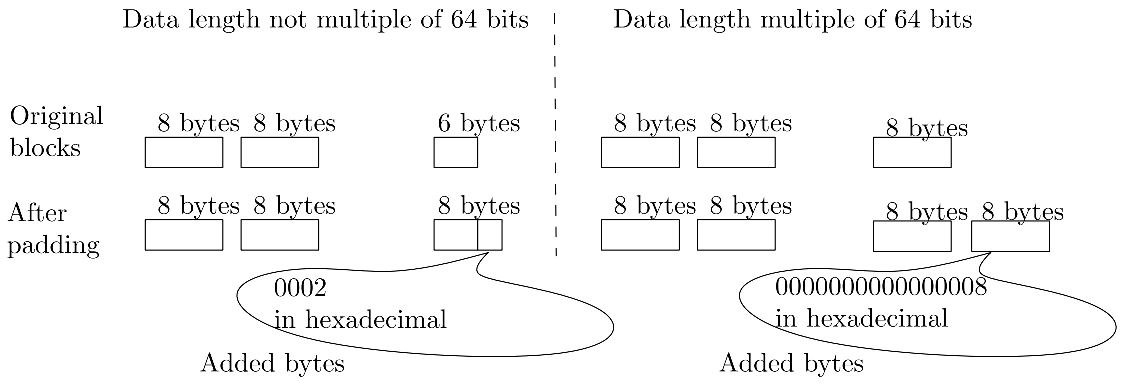


Figure 7: Examples of padding.

2.2 Cipher Block Chaining Mode

In the ECB mode, similar patterns in the plain data are transformed into similar patterns in the cipher data. It is therefore generally not recommended to use ECB. In order to fix this, the Cipher Block Chaining (CBC) mode was proposed. The first step is similar to ECB. It consists of splitting the message to be encrypted into 64-bit blocks and adding the padding bytes. However, instead of encrypting these blocks, each block is first XOR'ed with the cipher output of the previous block. The first block, however, is XOR'ed with a random Initialization Vector (IV) of 64 bits. Of course, the IV needs to be sent along with the transmitted cipher data. The system is depicted in Figure 8.

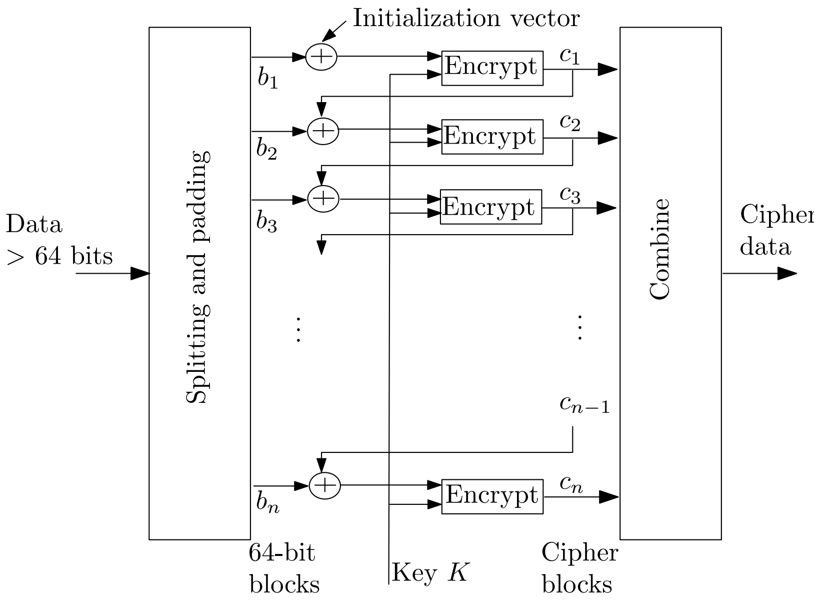


Figure 8: CBC mode of operation.

The disadvantage here is that, in order to obtain a block b_i of plain data during decryption, we need both c_i and c_{i-1} so if any cipher block is lost or corrupt, two plain data blocks are lost or corrupt during decryption.

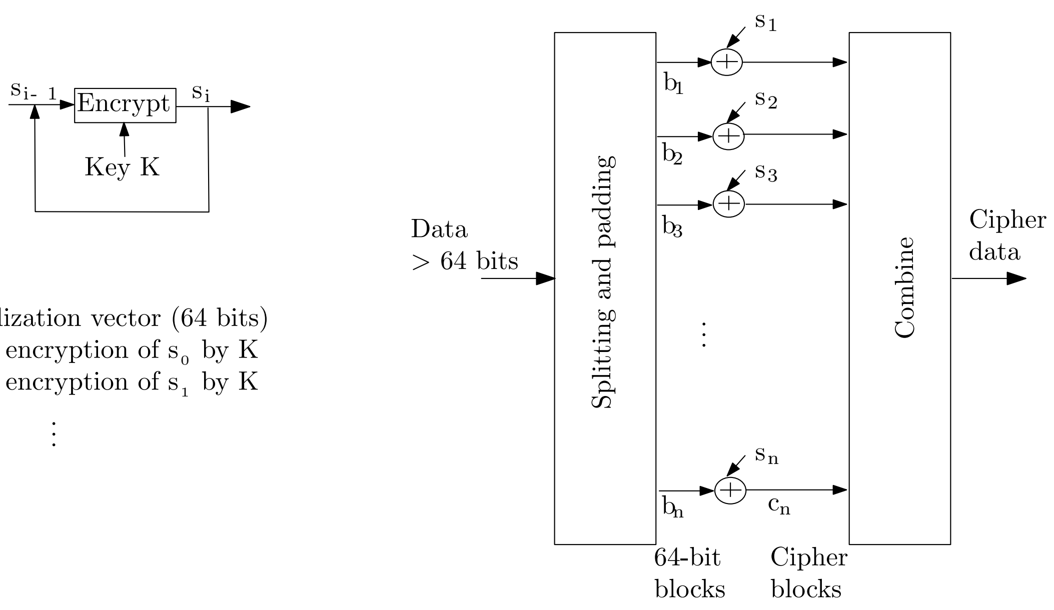
2.3 Output Feedback Mode

In the previous two modes of operation, we have to wait till the data to be encrypted is present and then begin the encryption process. In the Output FeedBack (OFB) mode, we generate a sequence of bits by using successive DES encryption. This sequence, generated in 64-bit blocks, or in general in k-bit blocks, is then XOR'ed with the plain data to get the cipher data. The advantage here is that this sequence of bits can be generated offline, i.e., when the message to be encrypted is still not available. Then, once the message is available, the encryption is just an XOR operation which is very fast. The OFB operation mode is shown in Figure 9.

References

[1] Bruce Schneier. *Applied Cryptography (Second Edition).* John Wiley & Sons, 1996.

[2] D. Stinson. *Cryptography: Theory and Practice.* Discrete Mathematics and Its Applications. Chapman & Hall/CRC, 2005.



s_0 = intialization vector (64 bits)
s_1 = DES encryption of s_0 by K
s_2 = DES encryption of s_1 by K

$\vdots$

Figure 9: OFB mode of operation.