Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering Department

Cryptography and Security

Laboratory work #4

Variant 11

Block Ciphers. DES

Author: Timur Cravtov
std. gr. FAF-231

Verified by: Maia Zaica
asist. univ

Chișinău
2025

Contents

1 Introduction 3

2 DES Cipher 3

3 Implementation of DES Cipher 4

3.1 Main datatypes 4

3.2 Tables 4

4 Demonstation 7

5 Conclusion 9

1 Introduction

Blocks ciphers are a type of ciphers which encode a fixed size block. They are more used in modern cryptography than stream ciphers due to their enhanced security features. Block ciphers operate on fixed-size blocks of plaintext, transforming them into ciphertext using a symmetric key. This lab focuses on Data Encryption Standard (DES) cipher only.

2 DES Cipher

DES ciphers is a symmetric-key algorithm for the encryption of digital data. It was developed in the early 1970s at IBM and later adopted as a federal standard in the United States.

DES operates on 64-bit blocks of data using a 56-bit key. However, the key is often represented as a 64-bit value, with every eighth bit used for parity checking and not for encryption. In other words, in the result the KEY is permuted to 56 bits.

The steps of DES encryption are as follows [2] [3]:

- Divide the message into 64 blocks. Let's call each block M.
- Apply an initial permutation (IP) to each block M.
- Split the permuted block into two halves: left (L) and right (R), each 32 bits.
- Perform 16 rounds of processing, where in each round:
  1. we set $L_i = R_{i-1}$
  2. we set $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$, where K_i is the subkey for round i, generated from the main key using a key schedule algorithm, and f is a complex function involving expansion, substitution using S-boxes, and permutation.
- In the last round, with L_{16} and R_{16} we make a swap, so the final output before the last permutation is $R_{16}L_{16}$.
- Apply the final permutation (IP^{-1}) to the combined block to produce C.
- The cipher text is then outputed in necessary format

Despite its historical significance, DES is no longer considered secure for many applications due to its relatively short key length, which makes it vulnerable to brute-force attacks. Modern encryption standards, such as the Advanced Encryption Standard (AES), have largely replaced DES in most applications. For example, new ciphers like Triple DES (3DES) apply the DES algorithm three times to each data block, effectively increasing the key length and enhancing security.

3 Implementation of DES Cipher

My task: (2.11): From L_{16} and R_{16} compute C (ciphertext) and output in hex format.

Unfortunately, due to missread of the lab instructions, I implemented the full algorithm instead of some part of it. Thus, I will describe the full algorithms, because I don't want the work to go to waste.

The DES cipher was implemented in Kotlin. The source code can be found in GitHub reporitory [1]. In this laboratory work, I implemented both encryption and decryption functions of the DES. I have core function like encrypt and decrypt data block, which are used in DesIO module, which provides additional functionality like encrypting/decrypting hex strings, plaintext in ascii and etc.

3.1 Main datatypes

In my implementation, I mostly used BooleanArray to represent bits. For example, 64-bit block is represented as BooleanArray of size 64, where each element is either true (1) or false (0). This representation allows for easy manipulation of individual bits during the various transformations required by the DES algorithm. The input or output can be presented in different formats like hex string, ascii string or byteArray, but internally all data is converted to BooleanArray for processing.

3.2 Tables

DES algorithm relies on several predefined tables for permutations and substitutions. These tables are crucial for the various transformations that occur during the encryption and decryption processes. In Kotlin, they are represented as intArrays with the exception of S-boxes, which are represented as 2D arrays.

For example, this is how PC_1 table if presented in the code:

```kotlin
val PC_1 = intArrayOf(
    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
)
```

Listing 1: Table PC1

Most of the tables are simple permutation tables but S Boxes. The method of appliying S Box is presented below:

```
fun apply_S_box(block: BooleanArray, boxNumber: Int):
    BooleanArray {

    val row = (if (block[0]) 2 else 0) + (if (block[5]) 1 else
        0)
    val column = (if (block[1]) 8 else 0) +
            (if (block[2]) 4 else 0) +
            (if (block[3]) 2 else 0) +
            (if (block[4]) 1 else 0)

    val value = S_BOXES[boxNumber - 1][row][column]

    return BooleanArray(4) { bit ->
        (value shr (3 - bit) and 1) == 1 // 2bit => 0110 -> 0011
            -> 0001 = 1 => (~, ~, 1, ~)
    }
}
```

Listing 2: Table PC1

Now, let's describe the main functions of DES implementation.

```
fun BooleanArray.encryptDesBlock(key: BooleanArray):
    BooleanArray {

    val mPermuted = applyPermutation(IP, this) // keeps size, 64
        bits

    val L0 = mPermuted.sliceArray(0..31) // 32 bits
    val R0 = mPermuted.sliceArray(32..63) // 32 bits

    var LCurrent = L0
    var RCurrent = R0

    val KList = getKList(key, loggerActive);

    for (i in 1..16) {

```

```
15            val fRezult = f(RCurrent, KList[i-1])
16
17            val RNext = LCurrent xor fRezult
18
19            val LNext = RCurrent
20
21            LCurrent = LNext
22            RCurrent = RNext
23        }
24
25        val C = getCfromL16R16(LCurrent, RCurrent, loggerActive);
26
27        return C
28    }
```

Listing 3: Table PC1

In *3rd* line we get the message with initial permutation applied. Then in lines *5th* and *6th* we split the message into left and right parts. Then, in *11th* like we ket the list of Keys 1 . . . 16 with given key. The function is shown later.

Then, we have a loop, where we iterate through 16 rounds.

As we have theoretically, $L_i = R_{i-0}$; $R_i = L_i \oplus f(R_i, K_i)$

For each i we have:

- compute fRezult (15th line)
- Compute RNext = LCurrent xor fRezult (17th line)
- Set LNext = RCurrent (19th line)
- Update LCurrent and RCurrent for the next iteration (21st and 22nd lines)

And finally, with L_{16} and R_{16} we apply the final permutation and get the ciphertext.

Now, I'll show how the `f` is defined:

```
internal fun f(Rn: BooleanArray, Kn_plus_1: BooleanArray): BooleanArray {

    val R_E_permuted = applyPermutation(E, Rn) // extends R to 48 bits

    val R_Permuted_XOR_ed = R_E_permuted xor Kn_plus_1
    val B_blocks = R_Permuted_XOR_ed.toList().chunked(6).map {
        it.toBooleanArray() } // 48 -> 8 blocks of 6 bits

    val sBoxOutput = B_blocks.mapIndexed { i, b ->

        val sBox = S_BOXES[i]
        val sResult = apply_S_box(b, i + 1)
        sResult.toList()

    }.flatten().toBooleanArray()

    val PPermRezult = applyPermutation(P, sBoxOutput)
    return PPermRezult
}
```

Listing 4: Function f

Mathematically, the function f can be described as follows:

- Expand the 32-bit input R_n to 48 bits using the
- XOR the expanded R_n with the 48-bit subkey K_{n+1}.
- Divide the 48-bit result into eight 6-bit blocks.
- For each 6-bit block, use the corresponding S-box to substitute it with a 4-bit block.
- Concatenate the eight 4-bit blocks to form a 32-bit block.
- Apply a P permutation to the 32-bit block to produce the output of the function f.

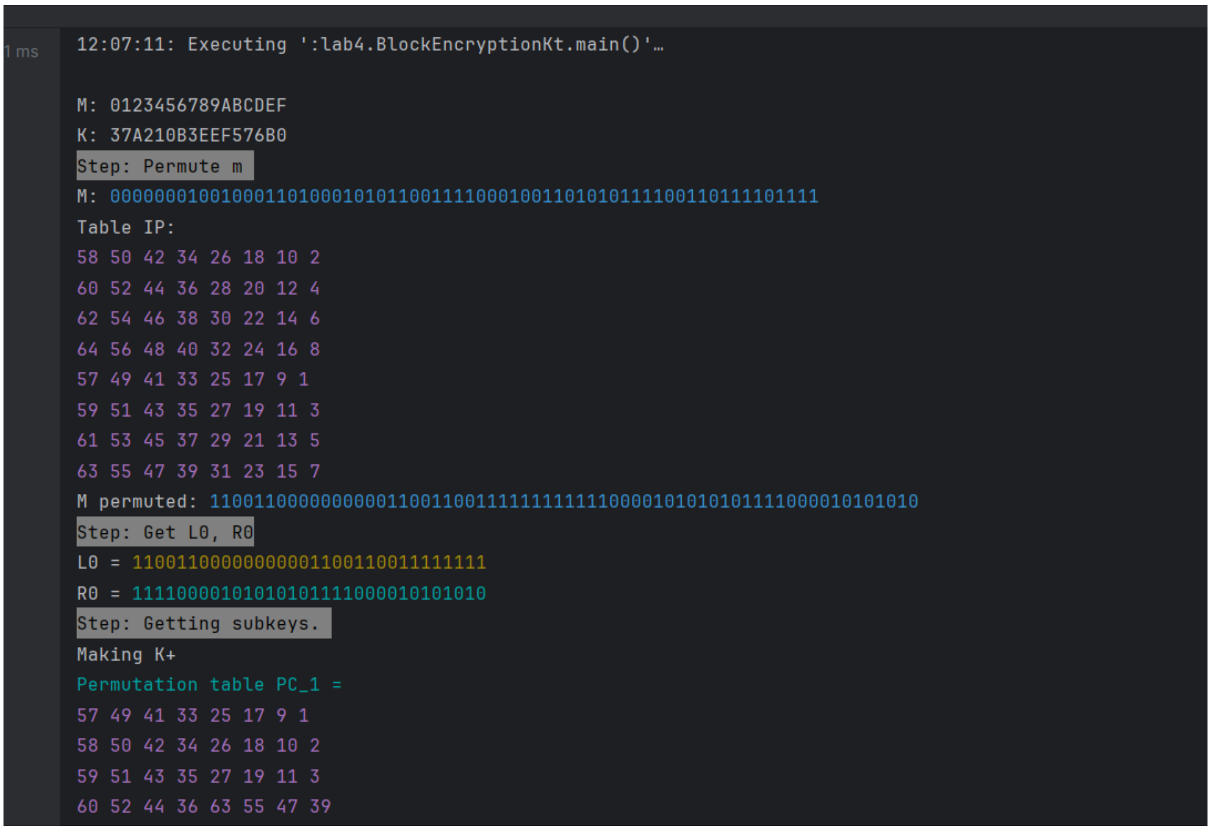
4 Demonstation

To show the implementation of the algorithm, I'll encrypt and decrypt a simple hex encoded block.

```
fun main() {
    val M = "0123456789ABCDEF"
    val K = randomBooleanArray(64).toHexString();
    println("M: $M")
    println("K: $K")
    val enc = M.hexToBooleanArray().encryptDesBlock(K.
        hexToBooleanArray(), true)
    println("C: ${enc.toHexString()}")
}
```

Listing 5: Table PC1

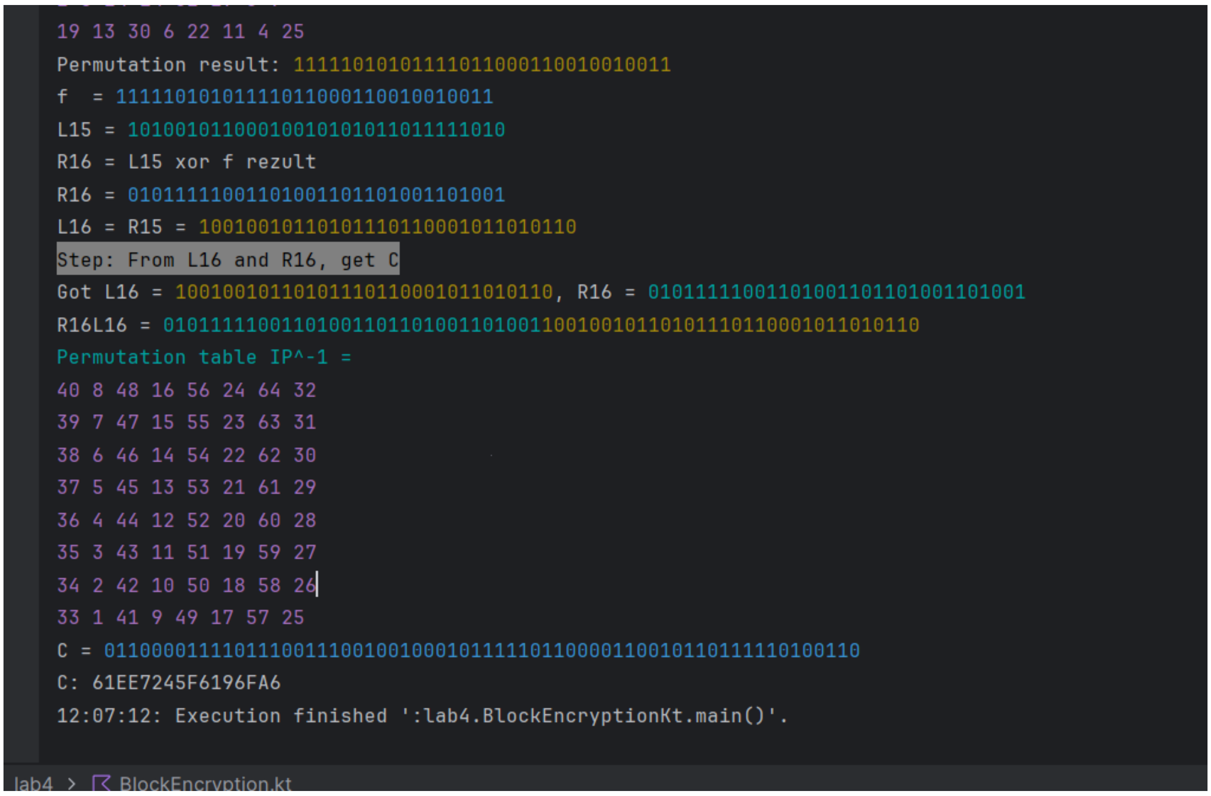
Now, let's see the output for the encryption:



```
1 ms   12:07:11: Executing ':lab4.BlockEncryptionKt.main()'…

M: 0123456789ABCDEF
K: 37A210B3EEF576B0
Step: Permute m
M: 0000000100100011010001010110011110001001101010111100110111101111
Table 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
M permuted: 1100110000000000110011001111111111110000101010101111000010101010
Step: Get L0, R0
L0 = 11001100000000001100110011111111
R0 = 11110000101010101111000010101010
Step: Getting subkeys.
Making K+
Permutation table 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
```

Figure 1: Encryption step 1



```
19 13 30 6 22 11 4 25
Permutation result: 11111010101111011000110010010011
f  = 11111010101111011000110010010011
L15 = 10100101100010010101011011111010
R16 = L15 xor f rezult
R16 = 01011111001101001101101001101001
L16 = R15 = 10010010110101110110001011010110
Step: From L16 and R16, get C
Got L16 = 10010010110101110110001011010110, R16 = 01011111001101001101101001101001
R16L16 = 0101111100110100110110100110100110010010110101110110001011010110
Permutation table 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
C = 0110000111101110011100100100010111110110000110010110111110100110
C: 61EE7245F6196FA6
12:07:12: Execution finished ':lab4.BlockEncryptionKt.main()'.
```

Figure 2: Encryption step final

As one can see, the hex encoded block `0123456789ABCDEF` with key `37A210B3EEF576B0` is encrypted to `61EE7245F6196FA6`.

The final step, getting C from L0 and R16 is outputed above.

5 Conclusion

In this laboratory work, I have implemented the DES cipher in Kotlin. The implementation includes both encryption and decryption functions, along with necessary data transformations. Even thought DES format is considered insecure for modern applications, understanding its structure and operation provides valuable insights into symmetric-key cryptography. The implementation demonstrates the key concepts of block ciphers, including permutations, substitutions, and key scheduling.

References

[1] GitHub repository `https://github.com/TimurCravtov/CryptographyAndSecurityLabs`

[2] DES illustrated `https://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm`

[3] Lecture Notes CS