Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Department of Physics

Criptography and security

Laboratory work #4

Block ciphers. DES algorithm

Author:
Dmitrii Belih
std. gr. FAF-232

Verified:
Zaica M.

Chișinău 2025

Theory Background

The Data Encryption Standard (DES) is a symmetric key block cipher developed by IBM in the early 1970s and adopted by the U.S. National Bureau of Standards (NBS) in 1977 as a federal standard (FIPS PUB 46). DES encrypts data in 64-bit blocks using a 56-bit key. Its design is based on the Feistel cipher structure, which divides the data block into two halves and processes them through multiple rounds of substitution and permutation.

Structure of DES

DES uses a 16-round Feistel network, meaning that the encryption process consists of 16 similar stages or rounds. Each round uses a subkey derived from the main key. The overall structure can be summarized as follows:

1. Initial Permutation (IP)
2. Sixteen Feistel rounds
3. Final Permutation (IP^{-1})

The Feistel structure allows the same algorithm to be used for both encryption and decryption, differing only in the order of the subkeys.

Initial Permutation (IP)

Before the 16 rounds begin, DES applies an *initial permutation* (IP) to the 64-bit plaintext. This permutation rearranges the bits of the plaintext according to a predefined table. Although IP does not contribute to the security of DES, it was included for hardware efficiency in the original design.

The Feistel Function

Each round of DES uses a function $f(R_{i-1}, K_i)$, where R_{i-1} is the right half of the data and K_i is the round subkey. The function consists of the following steps:

1. **Expansion (E-box):** The 32-bit R_{i-1} is expanded to 48 bits.
2. **Key Mixing:** The expanded block is XORed with the 48-bit round key K_i.
3. **Substitution (S-boxes):** The result is divided into eight 6-bit blocks, each substituted by a 4-bit output using one of eight substitution boxes (S-boxes).

4. **Permutation (P-box):** The 32-bit output from the S-boxes is permuted using a fixed P-box.

Finally, the result of $f(R_{i-1}, K_i)$ is XORed with the left half L_{i-1} of the previous round to produce the new right half:

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

Key Generation Process

The key schedule in DES generates sixteen 48-bit round keys from the original 56-bit key. The process is as follows:

1. Apply the *Permuted Choice 1 (PC-1)* to select 56 bits from the 64-bit key (8 bits are used for parity).
2. Split the key into two 28-bit halves (C_0 and D_0).
3. For each round i, the halves are shifted left by 1 or 2 bits according to a predefined schedule.
4. Apply the *Permuted Choice 2 (PC-2)* to produce the 48-bit round key K_i.

Encryption and Decryption

The encryption process consists of applying the 16 Feistel rounds to the permuted plaintext. The final result is the ciphertext obtained after applying the inverse initial permutation (IP^{-1}).

Decryption in DES uses the same process but with the subkeys applied in reverse order ($K_{16}, K_{15}, \ldots, K_1$) due to the symmetric nature of the Feistel structure.

The Task

Develop a program in one of the student's preferred programming languages to implement an element of the DES (Data Encryption Standard) algorithm. The specific task assigned to each student is determined by their position number n in the group list, using a given formula ex = n mod 11, where n corresponds to the list of students number.

My Variant:
Given K+ in the DES algorithm, determine Ci and Di for a given i.

Technical implementation

In DES, the 56-bit key K^+ is split into two halves:

$$C_0 = K^+[1..28], \quad D_0 = K^+[29..56]$$

At each round i, both halves are cyclically shifted to the left by either one or two positions, depending on the round number. The number of left shifts for each iteration is defined in the DES shift schedule:

```
        iteration = {
    1: 1, 2: 1, 3: 2, 4: 2, 5: 2, 6: 2, 7: 2, 8: 2,
    9: 1, 10: 2, 11: 2, 12: 2, 13: 2, 14: 2, 15: 2, 16: 1
}
```

Program Description

The program is implemented in Python and performs the following steps:

1. **Define the iteration table:** A dictionary named `iteration` is used to represent the shift schedule for each round i.

   ```
   iteration = {
       1: 1, 2: 1, 3: 2, 4: 2, 5: 2, 6: 2, 7: 2, 8: 2,
       9: 1, 10: 2, 11: 2, 12: 2, 13: 2, 14: 2, 15: 2, 16: 1
   }
   ```

2. **Generate or input a 56-bit key:** The user can either enter their own K^+ value or use a randomly generated 56-bit binary string.

   ```
   k_random = ""
   while len(k_random) < 56:
       k_random += str(random.randint(0, 1))
   ```

3. **Validate key length:** The program checks that K^+ is exactly 56 bits long.

4. **Split into two halves:** The 56-bit key is divided into two 28-bit halves:

   ```
   c0 = kplus[:28]
   d0 = kplus[28:]
   ```

5. **Read the round number i:** The user inputs a round number i, and the program verifies that $i \in [1, 16]$.

6. **Calculate total shift:** The total number of shifts up to iteration i is computed by summing the shift values from the iteration table.

```
count = 0
for j in iteration.keys():
    if j <= i_value:
        count += iteration[j]
```

7. **Perform circular shifts:** The left and right halves C_0 and D_0 are circularly shifted left by the total number of positions (`count`) to obtain C_i and D_i.

```
c_new = c0[count:] + c0[:count]
d_new = d0[count:] + d0[:count]
```

8. **Display results:** The program prints:
   - The iteration table
   - The initial halves C_0 and D_0
   - The total shift value
   - The computed C_i and D_i
   - All intermediate values for verification

Full code of the program:

```
import random

iteration = {
    1: 1,
    2: 1,
    3: 2,
    4: 2,
    5: 2,
    6: 2,
    7: 2,
    8: 2,
    9: 1,
    10: 2,
    11: 2,
    12: 2,
    13: 2,
    14: 2,
    15: 2,
    16: 1,
```

```
}

print(iteration.keys())
print(iteration[16])

k_random = ""

while len(k_random) < 56:
    k_random += str(random.randint(0, 1))

print(len(k_random))
print(k_random)

kplus = input("Enter kplus value: ")
if len(kplus) == 56:

    print(iteration)
    print(kplus)

    c0 = ""
    d0 = ""
    for i in range(len(kplus) // 2):
        c0 += str(kplus[i])

    for i in range(len(kplus) // 2):
        d0 += str(kplus[i + len(kplus) // 2])

    print(c0)
    print(d0)

    i_value = int(input("Write i value: "))
    if i_value < 17 and i_value > 1:
        print("i value is correct")
        count = 0
        for i in iteration.keys():
            if i <= i_value:
                count += iteration.get(i)
                print(iteration.get(i))
            else:
                break

        print("--------------------------------")
        print(iteration.get(i_value))

        print("--------------------------------")
        print(count)
```

```
66
67          c_new = c0[count:] + c0[:count]
68          d_new = d0[count:] + d0[:count]
69
70          print(c_new + " C of iteration: " + str(i_value))
71          print(d_new + " D of iteration: " + str(i_value))
72
73          for i in range(i_value + 1):
74              c_all = c0[i:] + c0[:i]
75              d_all = d0[i:] + d0[:i]
76              print("--------------------------------")
77              print(c_all + " C iteration: of " + str(i))
78              print(d_all + " D iteration: of " + str(i))
79      else:
80          print("i value must be between 1 and 16")
81
82  else:
83      print("kplus must be 56 characters long")
```

Results

```
D:\CS-lab\lab4>python main.py
dict_keys([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16])
1
56
01111010000001110111111010100110001010011100110101011110
Enter kplus value: 01111010000001110111111010100110001010011100110101011110
{1: 1, 2: 1, 3: 2, 4: 2, 5: 2, 6: 2, 7: 2, 8: 2, 9: 1, 10: 2, 11: 2, 12: 2, 13: 2, 14: 2, 15: 2, 16: 1}
01111010000001110111111010100110001010011100110101011110
0111101000000111011111101010
0110001010011100110101011110
Write i value: 5
i value is correct
1
1
2
2
2
------------------------------
2
------------------------------
8
00000111011111101010011111010 C of iteration: 5
10011100110101011110011000010 D of iteration: 5
--------------------------------
01111010000001110111111010100 C iteration: of 0
01100010100111001101010111110 D iteration: of 0
--------------------------------
11110100000011101111110101000 C iteration: of 1
11000101001110011010101111000 D iteration: of 1
--------------------------------
11101000000111011111101010001 C iteration: of 2
10001010011100110101011110001 D iteration: of 2
--------------------------------
11010000001110111111010100011 C iteration: of 3
00010100111001101010111100011 D iteration: of 3
--------------------------------
10100000011101111110101000111 C iteration: of 4
00101001110011010101111000110 D iteration: of 4
--------------------------------
01000000111011111110101001111 C iteration: of 5
01010011100110101011110011000 D iteration: of 5
```

Figure 1: Program execution showing the system

Conclusion

In this lab, we implemented a key step of the Data Encryption Standard (DES) algorithm: generating the intermediate keys C_i and D_i from the 56-bit key K^+. The program allows both manual and random input of the key and calculates the left and right halves for a given iteration i according to the DES shift schedule.

Through this exercise, we gained practical experience in understanding the DES key schedule and the role of cyclic shifts in subkey generation, splitting the 56-bit key into halves and applying iterative left rotations, displaying intermediate steps to verify the correctness of the algorithm, and writing a program that combines theoretical knowledge with computational implementation.

Overall, this lab helped us understand the inner workings of DES key management and reinforced the importance of careful handling of keys and iterative transformations in symmetric cryptography. The results demonstrate that the algorithm produces correct C_i and D_i values, providing a solid foundation for implementing full DES encryption and decryption in future tasks.