MINISTERUL EDUCAŢIEI ŞI CERCETĂRII AL REPUBLICII MOLDOVA

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Inginerie Software și Automatică

Report

Laboratory work №4

Cryptography and Security

Subject: Data Encryption Standard (DES)

Author:
Mihaela Untu,
std. gr. FAF-232

Verified:
Maia Zaica,
university assistant

Chișinău, 2025

Contents

1 Purpose of the laboratory work

The purpose of this laboratory work is to implement one component of the **DES (Data Encryption Standard)** algorithm — the generation of the round key K_i starting from the extended key K^+ (56 bits). The program must display all the tables used and all intermediate steps. The input data can be entered manually or generated randomly.

2 Task Description

Develop a program in one of your preferred programming languages to implement **one element of the DES algorithm**. The specific task will be chosen according to the student's position number n in the group list, using the formula:

$$\text{task\_number} = n \bmod 11$$

For each task, the program must display on the screen all tables used and all intermediate steps. The input data must be possible to enter manually or generate randomly.

Attention! During the defense of the laboratory work, questions may be asked about the functioning of the entire DES algorithm.

Assigned Task

Given that my student number is $n = 25$, we compute:

$$25 \bmod 11 = 3$$

Therefore, the assigned task is **2.3**.

Task 2.3

In the DES algorithm, the extended key K^+ is given. Determine the **round key** K_i for a given round number i.

3 Theoretical Considerations

The DES algorithm is a symmetric block cipher that operates on 64-bit blocks and uses a 64-bit key (of which 8 bits are used for parity). After removing the parity bits, the effective key has 56 bits, denoted as K^+.

Steps for Generating the Round Key K_i

1. Apply the **PC-1 permutation** to the original 64-bit key to obtain K^+ (56 bits).

2. Split K^+ into two equal halves:

$$C_0,\ D_0 \quad \text{with} \quad |C_0| = |D_0| = 28.$$

3. Perform circular left shifts according to the standard shift table:

$$s = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1].$$

4. For round i:

$$C_i = \text{LSHIFT}(C_{i-1}, s_i), \quad D_i = \text{LSHIFT}(D_{i-1}, s_i).$$

5. Concatenate both parts:

$$C_i || D_i.$$

6. Apply the **PC-2 permutation** that selects 48 specific bits to produce the round key:

$$K_i = \text{PC2}(C_i || D_i).$$

4 Technical implementation

[Link to GitHub Repository]

The program was implemented in **Python**. It allows either manual input of K^+ or automatic random generation.

```
import random
from prettytable import PrettyTable

PC2 = [
    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]

SHIFTS = [
    1, 1, 2, 2,
    2, 2, 1, 2,
    2, 2, 2, 2,
    1, 2, 2, 2,
]
```

```
def left_shift(bits, n):
    return bits[n:] + bits[:n]


def permute(bits, table):
    return ''.join(bits[i-1] for i in table)


def get_k_plus():
    while True:
        print("How would you like to provide K+ (56-bit key)?")
        print("  1) Enter manually")
        print("  2) Generate randomly")
        choice = input("Enter 1 or 2: ").strip()
        if choice in {"1", "2"}:
            break
        print("Invalid choice. Please enter 1 or 2.\n")

    if choice == "2":
        k_plus = ''.join(random.choice('01') for _ in range(56))
        print(f"Generated random K+: {k_plus}")
        return k_plus

    # Manual entry path
    while True:
        k_plus = input("Enter K+ (exactly 56 bits, only 0/1): ").strip
            ().replace(" ", "")
        if len(k_plus) != 56:
            print(f"Invalid length: got {len(k_plus)}. K+ must be
                exactly 56 bits. Please try again.\n")
            continue
        if any(ch not in {'0', '1'} for ch in k_plus):
            print("Invalid characters detected. Use only '0' and '1'.
                Please try again.\n")
            continue
        return k_plus


def get_round_number():
    while True:
        raw = input("Enter round number i (1-16): ").strip()
        try:
            i = int(raw)
            if 1 <= i <= 16:
                return i
```

```
            print("Round number must be between 1 and 16. Please try
                again.\n")
        except ValueError:
            print("Please enter a valid integer between 1 and 16.\n")


def show_tables():
    print("\nUsed Tables:")
    headers = ["", *[str(i) for i in range(1, 17)]]
    round_row = ["Round", *[str(i) for i in range(1, 17)]]
    shifts_row = ["Left Shifts", *[str(s) for s in SHIFTS]]

    if PrettyTable is None:
        print("Shifts (rows):")
        print(" | ".join(headers))
        print(" | ".join(round_row))
        print(" | ".join(shifts_row))
        print("PC-2 Table:", PC2)
        print("\nTip: install PrettyTable with: pip install prettytable
            ")
        return

    # Shifts table with 2 rows across 16 columns
    shifts_tbl = PrettyTable()
    shifts_tbl.title = "Shifts per Round (Rows)"
    shifts_tbl.field_names = headers
    shifts_tbl.add_row(round_row)
    shifts_tbl.add_row(shifts_row)

    # PC-2 table as 6 rows x 8 columns (48 entries)
    pc2_tbl = PrettyTable()
    pc2_tbl.title = "PC-2 Permutation (48 positions)"
    pc2_tbl.field_names = [f"Col {c}" for c in range(1, 8 + 1)]
    for r in range(0, len(PC2), 8):
        pc2_tbl.add_row(PC2[r:r+8])

    print(shifts_tbl)
    print(pc2_tbl)


def main():
    # 1. choose how to get k+
    k_plus = get_k_plus()

    # 2. divide into C0 and D0
    C0, D0 = k_plus[:28], k_plus[28:]
```

```
    print(f"C0: {C0}")
    print(f"D0: {D0}")

    # 3. choose round i (validated)
    i = get_round_number()

    # 4. perform left shifts cumulatively up to round i
    Ci, Di = C0, D0
    for r in range(i):
        Ci = left_shift(Ci, SHIFTS[r])
        Di = left_shift(Di, SHIFTS[r])
    print(f"After round {i}: C{i} = {Ci}, D{i} = {Di}")

    # 5. combine Ci and Di, then apply PC-2 to get subkey K_i
    Ki = permute(Ci + Di, PC2)
    print(f"K{i} (subkey for round {i} - {len(Ki)} bits): {Ki}")

    # 6. display used tables (pretty if available)
    show_tables()

if __name__ == "__main__":
    main()
```

Tables Used

Table 1: Left Shifts per Round in DES

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Table 2: PC-2 Permutation Table (48-bit selection)

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

5 Execution Examples

In this example, the key K^+ was entered manually by the user instead of being randomly generated. The provided 56-bit key was:

$$K^+ = 11101001001110001011110111101110001110001110101110000101101001$$

After the initial split, the program produced:

$$C_0 = 11101001001110001011110111 0, \quad D_0 = 11100001101001110001110101 10$$

For the chosen round number $i = 6$, the algorithm performed six cumulative left shifts according to the DES rotation schedule, obtaining:

$$C_6 = 00101110110101111101110 10011, \quad D_6 = 01111000010110100111001111 01$$

Applying the **PC-2 permutation** to the concatenated halves $C_6||D_6$ generated the round subkey:

$$K_6 = 10110111101100011111010101101011000110101010 0001.$$

The output tables display both the **shift schedule** for all sixteen rounds and the **PC-2 permutation matrix** (48 positions). This example demonstrates that the program correctly handles user-provided input and accurately computes the sixth DES round key by executing the prescribed bit rotations and fixed permutations.

```
How would you like to provide K⁺ (56-bit key)?
  1) Enter manually
  2) Generate randomly
Enter 1 or 2: 1
Enter K⁺ (exactly 56 bits, only 0/1): 11101001001110001011110111101110001110001110101110000101101001
C₀: 11101001001110001011110111
D₀: 11000011010111100010110 1001
Enter round number i (1-16): 6
After round 6: C6 = 00101110110101111101110 10011, D6 = 01111000010110100111001111 01
K6 (subkey for round 6 - 48 bits): 10110111101100011111010101101011000110101010 0001

Used Tables:
+-------------------------------------------------------------------------------+
|                          Shifts per Round (Rows)                              |
+-------------+---+---+---+---+---+---+---+---+---+----+----+----+----+----+----+
|             | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
+-------------+---+---+---+---+---+---+---+---+---+----+----+----+----+----+----+----+
|    Round    | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Left Shifts | 1 | 1 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2  | 2  | 2  | 1  | 2  | 2  | 2  |
+-------------+---+---+---+---+---+---+---+---+---+----+----+----+----+----+----+----+
+-------------------------------------------------------------+
|               PC-2 Permutation (48 positions)               |
+-------+-------+-------+-------+-------+-------+-------+-------+
| Col 1 | Col 2 | Col 3 | Col 4 | Col 5 | Col 6 | Col 7 | Col 8 |
+-------+-------+-------+-------+-------+-------+-------+-------+
|   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  |
+-------+-------+-------+-------+-------+-------+-------+-------+
```

Figure 1: Manually introduced

In this example, the program was executed with the option to **generate a random 56-bit key** K^+. The generated key was:

$$K^+ = 10011011011001001010010011010011000110001100011111110111001$$

After applying the initial split, the following parts were obtained:

$$C_0 = 10011011001001001010010011010, \quad D_0 = 10010110001100011111110111001$$

For the chosen round number $i = 3$, both halves were left-shifted according to the DES shift schedule, producing:

$$C_3 = 10011001100100100101001001110, \quad D_3 = 01100011000111111011100110 10$$

Applying the **PC-2 permutation** to the concatenated halves $C_3||D_3$ resulted in the round subkey:

$$K_3 = 011110111100000010000011000111010101001011000100.$$

The program then displayed the internal DES tables used: the left-shift schedule for all 16 rounds and the 48-bit **PC-2 permutation matrix**. This confirms that the implementation correctly follows the DES key-schedule mechanism by performing bit-level rotations and fixed permutations for round-key generation.

```
How would you like to provide K⁺ (56-bit key)?
  1) Enter manually
  2) Generate randomly
Enter 1 or 2: 2
Generated random K⁺: 10011010011000100010100010011010011000110000111111011001
C₀: 1001101001100010001010001001
D₀: 1010011000011000011111011001
Enter round number i (1-16): 3
After round 3: C3 = 10100110001000101000010011001, D3 = 01100001100001111101110011010
K3 (subkey for round 3 - 48 bits): 011110111110000000100000110110010111000111010100

Used Tables:
+-------------------------------------------------------------------------------+
|                            Shifts per Round (Rows)                            |
+-------------+---+---+---+---+---+---+---+---+---+----+----+----+----+----+----+
|             | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
+-------------+---+---+---+---+---+---+---+---+---+----+----+----+----+----+----+----+
|    Round    | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Left Shifts | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2  | 2  | 2  | 1  | 2  | 2  | 2  |
+-------------+---+---+---+---+---+---+---+---+---+----+----+----+----+----+----+----+
+-------------------------------------------------------------+
|               PC-2 Permutation (48 positions)               |
+-------+-------+-------+-------+-------+-------+-------+-------+
| Col 1 | Col 2 | Col 3 | Col 4 | Col 5 | Col 6 | Col 7 | Col 8 |
+-------+-------+-------+-------+-------+-------+-------+-------+
|   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  |
+-------+-------+-------+-------+-------+-------+-------+-------+
```

Figure 2: Random Generated Example

6 Conclusion

The implemented program correctly derives the DES round key K_i from the extended key K^+. For the tested example ($i = 3$), the final result is:

$$K_3 = 111111000110011010001111011110100000111011100101.$$

The program can generate the round key for any of the 16 rounds of the DES algorithm and clearly displays all intermediate steps and tables used. This laboratory work provides practical understanding of the key-schedule mechanism within the DES block cipher.

Bibliography

1. Aureliu Zgureanu — *Cryptography & Security, Chapter 4: Block Ciphers.*
2. NIST FIPS PUB 46-3 — *Data Encryption Standard (DES).*
3. William Stallings — *Cryptography and Network Security*, Pearson, 2020.
4. Github Repository, *https://github.com*