MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF THE REPUBLIC OF MOLDOVA

Technical University of Moldova Faculty of Computers, Informatics and Microelectronics Department of Software and Automation Engineering

Postoronca Dumitru FAF-233

Report

Laboratory work n.4

on CS

Checked by:
M. Zaica, *university assistant*
DISA, FCIM, UTM

Chișinău – 2025

1 Purpose of the Laboratory Work

The purpose of this laboratory work was to study and implement a specific component of the DES (Data Encryption Standard) algorithm. DES is a symmetric-key block cipher that encrypts 64-bit blocks of plaintext using a 56-bit key through 16 rounds of processing. This laboratory work focuses on implementing Task 2.2 from the assignment list: **Given K+ in the DES algorithm, determine Ci and Di for a given round i**.

The task requires understanding the DES key schedule generation process, particularly how the permuted key K+ (56 bits after PC-1 permutation) is split into two 28-bit halves (C0 and D0) and how these halves undergo left circular shifts in each round to generate the values Ci and Di, which are subsequently used to create round keys Ki through the PC-2 permutation.

The goal was to implement a C++ program that accurately performs these operations, displays all intermediate steps, and allows for both manual input and random generation of K+ values, thereby demonstrating a deep understanding of this crucial component of the DES encryption algorithm.

2 Theoretical Background

2.1 DES Algorithm Overview

The Data Encryption Standard (DES) is a symmetric block cipher developed by IBM and adopted as a federal standard in 1977. It operates on 64-bit blocks using a 56-bit key (stored as 64 bits with 8 parity bits). The algorithm follows a Feistel network structure with 16 rounds of processing.

2.2 DES Key Schedule

The key schedule is the process of generating 16 round keys (K1 through K16), each 48 bits long, from the original 56-bit key. The process consists of the following steps:

1. **Permuted Choice 1 (PC-1):** The 64-bit key undergoes PC-1 permutation, which discards the 8 parity bits and rearranges the remaining 56 bits to produce K+.

2. **Splitting K+:** The 56-bit K+ is split into two 28-bit halves:

    - C0 = left 28 bits of K+

- D0 = right 28 bits of K+

3. **Left Circular Shifts:** For each round i (1 to 16), both Ci-1 and Di-1 undergo left circular shifts according to the shift schedule to produce Ci and Di.

4. **Permuted Choice 2 (PC-2):** The concatenated Ci and Di (56 bits) undergo PC-2 permutation to produce the 48-bit round key Ki.

2.3 Shift Schedule

The DES shift schedule, defined in FIPS PUB 46-3, specifies the number of left circular shift positions for each round:

Round	Shifts
1, 2, 9, 16	1
3-8, 10-15	2

The total number of shifts across all 16 rounds is 28, which equals the size of C and D registers. This property ensures that the key schedule can be easily reversed for decryption.

3 Strategy Used

To complete Task 2.2, I developed a C++ program that calculates Ci and Di for any given round i, starting from K+. The program is designed to be interactive and educational, showing all intermediate steps of the calculation process.

- **Input Options:** The user can either manually enter a 56-bit K+ value or generate one randomly.
- **Round Selection:** The user specifies the round number i (1-16) for which Ci and Di should be calculated.
- **Intermediate Steps Display:** The program displays the results after each round from 1 to i, showing how C and D evolve through cumulative shifts.
- **Multiple Representations:** All bitsets are displayed in both binary and hexadecimal formats for clarity.

The core strategy revolves around several key implementation decisions: using C++ `bitset` for precise bit manipulation, implementing left circular shifts correctly, and displaying the complete shift schedule table along with all intermediate computations.

3.1 Implementation Details

The program is built around several core functions and data structures that handle bit manipulation and the key schedule logic.

3.1.1 Data Structure Selection: bitset

The program uses C++ `bitset` template class for representing K+, C, and D. This choice offers several advantages:

- **Exact Size Matching:** `bitset<56>` for K+ and `bitset<28>` for C and D provide exact bit-width requirements without waste.
- **Memory Efficiency:** Bitsets use minimal memory (4 bytes for 28 bits) compared to alternatives like bool arrays (28 bytes).
- **Easy Bit Access:** Individual bits can be accessed using array-like syntax: `bits[i]`.
- **Type Safety:** Compile-time size checking prevents mixing incompatible bit widths.
- **Built-in Conversions:** Easy conversion to hexadecimal and binary string representations.

3.1.2 Left Circular Shift Implementation

The core operation in the key schedule is the left circular shift. A left circular shift by n positions moves each bit n positions to the left, wrapping bits that fall off the left end to the right end.

```cpp
bitset<28> leftCircularShift(bitset<28> bits, int shifts) {
    bitset<28> result;
    for (int i = 0; i < 28; i++) {
        result[i] = bits[(i + 28 - shifts) % 28];
    }
    return result;
}
```

Listing 1: Left circular shift implementation for 28-bit registers.

The function works by taking each bit at position `i` in the result and assigning it the value from position `(i + 28 - shifts) % 28` in the input. For a left shift, each bit moves to a higher index position (towards the MSB), and bits at the top wrap around to the bottom. The modulo operation ensures that bits wrap around correctly.

Example: For a 1-position left shift:

- `result[0] = bits[27]` (bit 27 wraps to position 0)
- `result[1] = bits[0]` (bit 0 moves to position 1)
- `result[2] = bits[1]` (bit 1 moves to position 2)
- ...
- `result[27] = bits[26]` (bit 26 moves to position 27)

Visually, if C0 = `1101 0000 ... 0000`, then after a left shift by 1, C1 = `1010 0000 ... 0001`, where the leftmost bit wraps to the rightmost position.

3.1.3 Bitset Indexing Convention

An important consideration is that C++ `bitset` indexes bits in reverse order compared to how we typically write binary numbers:

- `bitset[0]` is the rightmost (least significant) bit
- `bitset[N-1]` is the leftmost (most significant) bit

Therefore, when splitting K+ into C0 and D0:

- C0 (left half) = bits [55:28] of K+
- D0 (right half) = bits [27:0] of K+

3.1.4 Key Schedule Algorithm

The main algorithm performs cumulative left shifts for each round from 1 to i:

```
// Initialize C and D with C0 and D0
bitset<28> C = C0;
bitset<28> D = D0;

int totalShifts = 0;
for (int round = 0; round < i; round++) {
    int shifts = SHIFT_SCHEDULE[round];
    totalShifts += shifts;

```

```
    // Perform left circular shifts
    C = leftCircularShift(C, shifts);
    D = leftCircularShift(D, shifts);

    // Display intermediate result
    cout << "Round " << (round + 1) << " (shift " << shifts
         << " position" << (shifts > 1 ? "s" : "") << "):" << endl;
    cout << "  C" << (round + 1) << ": " << formatBitset(C)
         << " (" << toHex(C) << ")" << endl;
    cout << "  D" << (round + 1) << ": " << formatBitset(D)
         << " (" << toHex(D) << ")" << endl;
}
```

Listing 2: Main algorithm for calculating Ci and Di.

The key insight is that shifts are **cumulative**: to obtain C5, we apply all shifts from rounds 1 through 5 sequentially. This is why we iterate from round 1 to round i, applying the shift for each round.

4 Execution Example

The following demonstrates the program's functionality with a concrete example where we calculate C5 and D5 from a randomly generated K+.

4.1 Program Output

```
==================================================
    DES Algorithm - Task 2.2 Implementation
    Determine Ci and Di for a given round i
==================================================

=== DES Left Shift Schedule Table ===
+--------+-----------+
| Round  | Shifts    |
+--------+-----------+
|    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     |
+--------+-----------+

How would you like to provide K+?
1. Enter manually (56 bits)
2. Generate randomly
Choice: 2
```

```
K+ generated randomly.

=== Initial K+ (56 bits) ===
Binary: 1111 0001 0110 1011 0110 1110 1111 0101 0000 1101 0110 1010 0010 0110
Hex:    0xF16B6EF50D6A26

=== Initial Split ===
C0 (left 28 bits):  1111 0001 0110 1011 0110 1110 1111 (0xF16B6EF)
D0 (right 28 bits): 0101 0000 1101 0110 1010 0010 0110 (0x50D6A26)

Enter round number i (1-16): 5

=== Calculating Ci and Di (showing all intermediate steps) ===

Round 1 (shift 1 position):
  C1: 1110 0010 1101 0110 1101 1101 1111 (0xE2D6DDF)
  D1: 1010 0001 1010 1101 0100 0100 1100 (0xA1AD44C)

Round 2 (shift 1 position):
  C2: 1100 0101 1010 1101 1011 1011 1111 (0xC5ADBBF)
  D2: 0100 0011 0101 1010 1000 1001 1001 (0x435A899)

Round 3 (shift 2 positions):
  C3: 0001 0110 1011 0110 1110 1111 1111 (0x16B6EFF)
  D3: 0000 1101 0110 1010 0010 0110 0101 (0x0D6A265)

Round 4 (shift 2 positions):
  C4: 0101 1010 1101 1011 1011 1111 1100 (0x5ADBBFC)
  D4: 0011 0101 1010 1000 1001 1001 0100 (0x35A8994)

Round 5 (shift 2 positions):
  C5: 0110 1011 0110 1110 1111 1111 0001 (0x6B6EFF1)
  D5: 1101 0110 1010 0010 0110 0101 0000 (0xD6A2650)

==================================================
                 FINAL RESULT
==================================================
```

```
For round i = 5:

C5 (28 bits):
  Binary: 0110 1011 0110 1110 1111 1111 0001
  Hex:    0x6B6EFF1

D5 (28 bits):
  Binary: 1101 0110 1010 0010 0110 0101 0000
  Hex:    0xD6A2650

Total left shifts applied: 8
==================================================
```

4.2 Analysis of Results

The output demonstrates several important aspects of the DES key schedule:

1. **Shift Schedule Display:** The program begins by showing the complete shift schedule table, which is defined in the FIPS 46-3 standard.

2. **Initial Values:** K+ is displayed in both binary (with 4-bit grouping for readability) and hexadecimal formats. The split into C0 and D0 shows the left 28 bits and right 28 bits respectively.

3. **Cumulative Shifts:** The intermediate steps show how C and D evolve through each round:

   - Round 1: Shift by 1 position (total: 1 shift)
   - Round 2: Shift by 1 position (total: 2 shifts)
   - Round 3: Shift by 2 positions (total: 4 shifts)
   - Round 4: Shift by 2 positions (total: 6 shifts)
   - Round 5: Shift by 2 positions (total: 8 shifts)

4. **Verification:** The total of 8 left shifts for the first 5 rounds can be verified: $1 + 1 + 2 + 2 + 2 = 8$ shifts.

5 Conclusion

This laboratory work provided a comprehensive understanding of the DES key schedule mechanism, specifically the generation of Ci and Di values through left circular shifts. By implementing Task 2.2, I gained practical experience with several important concepts:

5.1 Key Learning Points

- **DES Key Schedule Structure:** Understanding how the 56-bit K+ is split into two 28-bit registers and how these evolve through cumulative shifts according to a predefined schedule.
- **Bit Manipulation in C++:** The importance of choosing appropriate data structures for cryptographic implementations. The `bitset` template provides exact bit-width control, type safety, and efficient memory usage, making it ideal for implementing cryptographic algorithms.
- **Circular Shifts:** Implementing and understanding left circular shifts, which preserve all key bits while changing their positions. This operation is fundamental to the DES key schedule and ensures that all key bits contribute to multiple round keys.
- **Cumulative Transformations:** Understanding that Ci is not calculated directly from C0 with a single shift, but rather through a sequence of shifts applied in each round. This cumulative nature is essential for the security properties of DES.
- **Educational Design:** The importance of displaying intermediate steps in cryptographic implementations for understanding and verification purposes. The program shows all values from C1/D1 through Ci/Di, making it easy to trace the algorithm's execution.

5.2 Significance in DES Algorithm

The Ci and Di values calculated by this program are crucial intermediate values in the DES encryption process. After calculating Ci and Di for a round, the full DES algorithm would:

1. Concatenate Ci and Di to form a 56-bit value

2. Apply the PC-2 permutation to produce the 48-bit round key Ki
3. Use Ki in the F-function of round i to encrypt the data

Understanding this component is essential for comprehending the complete DES algorithm and provides a foundation for studying more modern block ciphers like AES, which also use complex key schedules.

5.3 Practical Applications

While DES is no longer considered secure for modern applications (due to its small key size), studying its implementation provides valuable insights into:

- Symmetric encryption algorithm design
- The role of key schedules in providing diffusion and confusion
- Bit-level operations in cryptographic systems
- The evolution from classical to modern cryptography

By successfully implementing and testing this program, I have demonstrated a solid understanding of this fundamental component of the DES algorithm and gained practical experience in cryptographic programming.

A Full Source Code

```cpp
#include <algorithm>
#include <bitset>
#include <iomanip>
#include <iostream>
#include <random>
#include <sstream>
#include <string>

using namespace std;

// Left shift schedule for DES (number of shifts for each round)
const int SHIFT_SCHEDULE[16] = {1, 1, 2, 2, 2, 2, 2, 2,
                                1, 2, 2, 2, 2, 2, 2, 1};

// Function to perform left circular shift on a 28-bit value
bitset<28> leftCircularShift(bitset<28> bits, int shifts) {
  bitset<28> result;
  for (int i = 0; i < 28; i++) {
    result[i] = bits[(i + 28 - shifts) % 28];
  }
  return result;
}

// Function to convert bitset to binary string with spaces
// every 4 bits for readability
template <size_t N> string formatBitset(bitset<N> bits) {
  string result = "";
  for (int i = N - 1; i >= 0; i--) {
    result += (bits[i] ? '1' : '0');
    if (i > 0 && i % 4 == 0) {
      result += " ";
    }
  }
  return result;
}

// Function to convert bitset to hexadecimal string
template <size_t N> string toHex(bitset<N> bits) {
  unsigned long long value = bits.to_ullong();
  stringstream ss;
  ss << "0x" << uppercase << hex << setw((N + 3) / 4)
     << setfill('0') << value;
  return ss.str();
```

```
}

// Function to display the shift schedule table
void displayShiftScheduleTable() {
  cout << "\n=== DES Left Shift Schedule Table ===" << endl;
  cout << "+--------+------------+" << endl;
  cout << "| Round  | Shifts     |" << endl;
  cout << "+--------+------------+" << endl;
  for (int i = 0; i < 16; i++) {
    cout << "|   " << setw(2) << (i + 1) << "   |     " << setw(2)
         << SHIFT_SCHEDULE[i] << "     |" << endl;
  }
  cout << "+--------+------------+" << endl;
}

int main() {
  cout << "================================================"
       << endl;
  cout << "    DES Algorithm - Task 2.2 Implementation" << endl;
  cout << "    Determine Ci and Di for a given round i" << endl;
  cout << "================================================"
       << endl;

  displayShiftScheduleTable();

  bitset<56> kPlus; // K+ (56 bits after PC-1)
  int choice;

  cout << "\nHow would you like to provide K+?" << endl;
  cout << "1. Enter manually (56 bits)" << endl;
  cout << "2. Generate randomly" << endl;
  cout << "Choice: ";
  cin >> choice;

  if (choice == 1) {
    string input;
    cout << "\nEnter K+ (56 bits, no spaces): ";
    cin >> input;

    // Remove any spaces
    input.erase(remove(input.begin(), input.end(), ' '),
                input.end());

    if (input.length() != 56) {
      cout << "Error: K+ must be exactly 56 bits!" << endl;
      return 1;
```

```
    }

    // Convert string to bitset (reverse order for bitset indexing)
    for (int i = 0; i < 56; i++) {
      kPlus[55 - i] = (input[i] == '1');
    }
  } else {
    // Generate random K+
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(0, 1);

    for (int i = 0; i < 56; i++) {
      kPlus[i] = dis(gen);
    }
    cout << "\nK+ generated randomly." << endl;
  }

  cout << "\n=== Initial K+ (56 bits) ===" << endl;
  cout << "Binary: " << formatBitset(kPlus) << endl;
  cout << "Hex:    " << toHex(kPlus) << endl;

  // Split K+ into C0 (left 28 bits) and D0 (right 28 bits)
  bitset<28> C0, D0;

  // Extract C0 (bits 55-28 in our representation)
  for (int i = 0; i < 28; i++) {
    C0[i] = kPlus[i + 28];
  }

  // Extract D0 (bits 27-0 in our representation)
  for (int i = 0; i < 28; i++) {
    D0[i] = kPlus[i];
  }

  cout << "\n=== Initial Split ===" << endl;
  cout << "C0 (left 28 bits):  " << formatBitset(C0) << " ("
       << toHex(C0) << ")" << endl;
  cout << "D0 (right 28 bits): " << formatBitset(D0) << " ("
       << toHex(D0) << ")" << endl;

  // Get round number from user
  int roundNumber;
  cout << "\nEnter round number i (1-16): ";
  cin >> roundNumber;

```

```
  if (roundNumber < 1 || roundNumber > 16) {
    cout << "Error: Round number must be between 1 and 16!"
         << endl;
    return 1;
  }

  // Calculate Ci and Di by performing cumulative shifts
  bitset<28> C = C0;
  bitset<28> D = D0;

  cout << "\n=== Calculating Ci and Di (showing all intermediate "
       << "steps) ===" << endl;

  int totalShifts = 0;
  for (int i = 0; i < roundNumber; i++) {
    int shifts = SHIFT_SCHEDULE[i];
    totalShifts += shifts;

    // Perform left circular shifts
    C = leftCircularShift(C, shifts);
    D = leftCircularShift(D, shifts);

    cout << "\nRound " << (i + 1) << " (shift " << shifts
         << " position" << (shifts > 1 ? "s" : "") << "):"
         << endl;
    cout << "  C" << (i + 1) << ": " << formatBitset(C)
         << " (" << toHex(C) << ")" << endl;
    cout << "  D" << (i + 1) << ": " << formatBitset(D)
         << " (" << toHex(D) << ")" << endl;
  }

  cout << "\n================================================"
       << endl;
  cout << "                 FINAL RESULT" << endl;
  cout << "================================================"
       << endl;
  cout << "For round i = " << roundNumber << ":" << endl;
  cout << "\nC" << roundNumber << " (28 bits):" << endl;
  cout << "  Binary: " << formatBitset(C) << endl;
  cout << "  Hex:    " << toHex(C) << endl;

  cout << "\nD" << roundNumber << " (28 bits):" << endl;
  cout << "  Binary: " << formatBitset(D) << endl;
  cout << "  Hex:    " << toHex(D) << endl;

  cout << "\nTotal left shifts applied: " << totalShifts << endl;
```

```cpp
    cout << "================================================"
         << endl;

    return 0;
}
```

Listing 3: Complete C++ source code for DES Task 2.2 implementation.