Ministry of Education, Culture and Research of the Republic of Moldova
Technical University of Moldova
Department of Software and Automation Engineering

REPORT

Laboratory work No. 4
Discipline: Cryptography and Security

Elaborated: FAF-221,
Revenco Victor

Checked: asist. univ.,
Dumitru Nirca

Chișinău 2024
Topic: Polialphabetic Cipher

Tasks:

În algoritmul DES în runda i este cunoscut Ki și Ri-1. De calculat B1B2B3B4B5B6B7B8

Theoretical notes:

Data Encryption Standard (DES) is a symmetric-key block cipher that encrypts data in fixed-size blocks of 64 bits using a 56-bit key. It was developed in the 1970s and became a widely used standard for secure data encryption.

Key Features of DES

1. **Block Cipher**:
   - DES operates on 64-bit blocks of plaintext, producing 64-bit ciphertext blocks.
2. **Symmetric-Key**:
   - The same key is used for encryption and decryption.
3. **Feistel Structure**:
   - DES employs a Feistel network, dividing data into two halves and iteratively applying a round function.
4. **Key Length**:
   - DES uses a 64-bit key, but only 56 bits are effective. The remaining 8 bits are parity bits used for error-checking.
5. **16 Rounds of Encryption**:
   - The plaintext undergoes 16 rounds of transformations, including permutation, substitution, and XOR operations.
6. **Initial and Final Permutations**:
   - DES starts with an Initial Permutation (IP) and ends with its inverse (IP^{-1}).

DES Algorithm Steps

1. **Initial Permutation (IP)**:
   - The plaintext is permuted using a fixed table, rearranging the bits.
2. **Splitting**:
   - The permuted plaintext is split into two 32-bit halves: and .
3. **Rounds (1 to 16)**:
   - For each round :
     - A round key is generated using a key scheduling algorithm.
     - The Feistel function is applied to and , and the result is XORed with .
     - The halves are swapped: , .
4. **Final Permutation (IP^{-1})**:
   - After 16 rounds, the halves are combined and permuted using the inverse of the Initial Permutation.

DES Components

1. **Expansion Table**:
   - Expands a 32-bit input into 48 bits for the Feistel function.
2. **S-Boxes**:

    - Eight substitution boxes that map 6-bit inputs to 4-bit outputs. These introduce non-linearity.
3. **P-Box**:
    - A fixed permutation applied to the S-Box output to increase diffusion.
4. **Key Schedule**:
    - Generates 16 subkeys (48 bits each) from the original 56-bit key using shifts and permutations.
5. **Feistel Function (f)**:
    - Combines the round key and right half of the data to produce a new 32-bit output.

Implementation

The code implements the **Feistel round** of DES, focusing on computing the outputs, which are the results of the S-Box substitutions. Below is an explanation of the key parts of the code:

1. Inputs:

- : The 32-bit right half of the previous round.
- : The 48-bit round key for the current round.

2. Expansion:

- The 32-bit is expanded to 48 bits using the **expansion table**.

3. XOR Operation:

- The expanded is XORed with the 48-bit round key . This combines the key into the data.

4. S-Box Substitution:

- The 48-bit result from the XOR operation is divided into 8 blocks of 6 bits each.
- Each block is processed by its corresponding S-Box to produce a 4-bit output.
- The combined result from all S-Boxes is 32 bits long.

5. Output:

- The 32-bit output represents , the result of the Feistel function for the current round.

```
import java.util.Random;
import java.util.Scanner;
```

```
public class DESRound {
    private static final int[] 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
    };

    private static final int[][][] S_BOXES = {
            {
                    {14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},
                    {0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8},
                    {4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},
                    {15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13}
            },
            {
                    {15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10},
                    {3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5},
                    {0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},
                    {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}
            },
            {
                    {10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8},
                    {13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1},
                    {13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7},
                    {1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12}
            },
            {
                    {7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15},
                    {13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9},
                    {10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4},
                    {3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14}
            },
            {
                    {2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9},
                    {14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6},
                    {4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14},
                    {11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3}
            },
            {
                    {12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11},
                    {10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8},
                    {9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6},
                    {4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13}
            },
            {
                    {4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1},
                    {13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6},
                    {1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2},
                    {6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12}
            },
            {
                    {13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7},
                    {1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2},
                    {7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8},
                    {2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11}
            }
    };
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Random random = new Random();

        System.out.println("Do you want to input values manually or
generate them randomly?");
        System.out.println("1. Manual input");
        System.out.println("2. Random generation");
        int choice = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        String riMinus1;
        String ki;

        if (choice == 1) {
            System.out.print("Enter R(i-1) (32 bits): ");
            riMinus1 = scanner.nextLine();
            System.out.print("Enter K(i) (48 bits): ");
            ki = scanner.nextLine();
        } else {
            riMinus1 = generateRandomBinaryString(32, random);
            ki = generateRandomBinaryString(48, random);
            System.out.println("Generated R(i-1): " + riMinus1);
            System.out.println("Generated K(i): " + ki);
        }

        System.out.println("===== DES ROUND =====");

        // Expansion
        String expandedRiMinus1 = permute(riMinus1, EXPANSION_TABLE);
        System.out.println("Expanded R(i-1) (length " +
expandedRiMinus1.length() + "): " + expandedRiMinus1);

        // XOR with Key
        String xorResult = xorStrings(expandedRiMinus1, ki);
        System.out.println("XOR with K(i) (length " + xorResult.length() +
"): " + xorResult);

        // S-Box substitution
        String sBoxResult = sBoxSubstitution(xorResult);
        System.out.println("S-Box Output (B1...B8): " + sBoxResult);

        scanner.close();
    }

    private static String generateRandomBinaryString(int length, Random
random) {
        StringBuilder sb = new StringBuilder(length);
        for (int i = 0; i < length; i++) {
            sb.append(random.nextInt(2)); // Append 0 or 1
        }
        return sb.toString();
    }

    private static String permute(String input, int[] table) {
        StringBuilder output = new StringBuilder();
        for (int index : table) {
            output.append(input.charAt(index - 1));
        }
        return output.toString();
```

```
    }

    private static String xorStrings(String a, String b) {
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < a.length(); i++) {
            result.append(a.charAt(i) ^ b.charAt(i));
        }
        return result.toString();
    }

    private static String sBoxSubstitution(String input) {
        StringBuilder output = new StringBuilder();

        if (input.length() != 48) {
            throw new IllegalArgumentException("Input to S-Box substitution
must be 48 bits long. Received: " + input.length());
        }

        for (int i = 0; i < 8; i++) {
            String block = input.substring(i * 6, (i + 1) * 6);

            int row = Integer.parseInt("" + block.charAt(0) +
block.charAt(5), 2);
            int col = Integer.parseInt(block.substring(1, 5), 2);

            int sBoxValue = S_BOXES[i][row][col];
            output.append(String.format("%4s",
Integer.toBinaryString(sBoxValue)).replace(' ', '0'));
        }
        return output.toString();
    }
}
```