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. 1
Discipline: Cryptography and Security

Elaborated: Grigoraș Dumitru FAF-221,

Checked: Dumitru Nirca asist. univ.,

Chișinău 2023

Topic: Caesar Cipher

Tasks 1:

Implement the Caesar algorithm for the English alphabet in one of the programming languages. Use only the letter encodings as shown in Table 1 (encodings specified in the programming language, e.g. ASCII or Unicode, are not allowed to be used). Key values shall be between 1 and 25 inclusive and no other values are allowed. Text character values shall be between 'A' and 'Z', 'a' and 'z' and no other values are prefixed. If the user enters other values - the user will be prompted for the correct slide. Before encryption the text will be converted to upper case and spaces will be removed. The user will be able to choose the operation - encryption or decryption, enter the key, message or cryptogram and get the cryptogram or decrypted message respectively.

Implementation

```
import java.util.Scanner;

public class CaesarCipher {
   private static final String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

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

       System.out.print("Choose operation (E for encryption, D for decryption):
");
       String operation = scanner.nextLine().toUpperCase();

       if (!operation.equals("E") && !operation.equals("D")) {
           System.out.println("Invalid operation. Please enter 'E' for
encryption or 'D' for decryption.");
           return;
       }

       System.out.print("Enter key (1-25): ");
       int key = scanner.nextInt();
       if (key < 1 || key > 25) {
           System.out.println("Invalid key. Please enter a value between 1 and
25.");
           return;
       }
       scanner.nextLine();

       System.out.print("Enter the message: ");
       String message = scanner.nextLine().toUpperCase().replaceAll(" ", "");
```

```
        String result;
        if (operation.equals("E")) {
            result = encrypt(message, key);
            System.out.println("Encrypted message: " + result);
        } else {
            result = decrypt(message, key);
            System.out.println("Decrypted message: " + result);
        }

        scanner.close();
    }

    private static String encrypt(String message, int key) {
        return shiftMessage(message, key);
    }

    private static String decrypt(String message, int key) {
        return shiftMessage(message, 26 - key);
    }

    private static String shiftMessage(String message, int shift) {
        StringBuilder shiftedMessage = new StringBuilder();

        for (char ch : message.toCharArray()) {
            int letterIndex = ALPHABET.indexOf(ch);

            // Only process valid alphabetic characters
            if (letterIndex != -1) {
                int newIndex = (letterIndex + shift) % 26;
                shiftedMessage.append(ALPHABET.charAt(newIndex));
            } else {
                System.out.println("Invalid character in the message. Only A-Z
characters are allowed.");
                return "";
            }
        }

        return shiftedMessage.toString();
    }
}
```

Output:

Encryption:

```
Choose operation (E for encryption, D for decryption): E
Enter key (1-25): 23
Enter the message: cevamesaj
Encrypted message: ZBSXJBPXG
```

Decryption:

```
Choose operation (E for encryption, D for decryption): d
Enter key (1-25): 23
Enter the message: ZBSXJBPXG
Decrypted message: CEVAMESAJ
```

Tasks 2:

Implement the Caesar algorithm with 2 keys, preserving the conditions expressed in Task 1.1. In addition, key 2 must contain only letters of the Latin alphabet, and have a length of not less than 7.

Implementation:

```
import java.util.LinkedHashSet;
import java.util.Scanner;
import java.util.Set;

public class CaesarCipherWithPermutation {
   private static final String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

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

       System.out.print("Choose operation (E for encryption, D for decryption):
");
       String operation = scanner.nextLine().toUpperCase();

       if (!operation.equals("E") && !operation.equals("D")) {
           System.out.println("Invalid operation. Please enter 'E' for
encryption or 'D' for decryption.");
           return;
```

```
        }

        System.out.print("Enter key 1 (1-25): ");
        int key1 = scanner.nextInt();
        if (key1 < 1 || key1 > 25) {
            System.out.println("Invalid key. Please enter a value between 1 and
25.");
            return;
        }
        scanner.nextLine();

        System.out.print("Enter key 2 (a word with unique letters, minimum length
7): ");
        String key2 = scanner.nextLine().toUpperCase();
        if (key2.length() < 7 || !key2.matches("[A-Z]+")) {
            System.out.println("Invalid key 2. Ensure it has only letters and is
at least 7 characters long.");
            return;
        }

        System.out.print("Enter the message: ");
        String message = scanner.nextLine().toUpperCase().replaceAll(" ", "");

        // Generate permuted alphabet based on key 2
        String permutedAlphabet = generatePermutedAlphabet(key2);

        String result;
        if (operation.equals("E")) {
            result = encrypt(message, key1, permutedAlphabet);
            System.out.println("Encrypted message: " + result);
        } else {
            result = decrypt(message, key1, permutedAlphabet);
            System.out.println("Decrypted message: " + result);
        }

        scanner.close();
    }

    private static String encrypt(String message, int key1, String
permutedAlphabet) {
        return shiftMessage(message, key1, permutedAlphabet);
    }

    private static String decrypt(String message, int key1, String
permutedAlphabet) {
        return shiftMessage(message, 26 - key1, permutedAlphabet);
    }

    private static String shiftMessage(String message, int shift, String
permutedAlphabet) {
        StringBuilder shiftedMessage = new StringBuilder();
```

```
        for (char ch : message.toCharArray()) {
            int letterIndex = permutedAlphabet.indexOf(ch);

            // Only process valid alphabetic characters
            if (letterIndex != -1) {
                int newIndex = (letterIndex + shift) % 26;
                shiftedMessage.append(permutedAlphabet.charAt(newIndex));
            } else {
                System.out.println("Invalid character in the message. Only A-Z
characters are allowed.");
                return "";
            }
        }

        return shiftedMessage.toString();
    }

    private static String generatePermutedAlphabet(String key2) {
        Set<Character> uniqueLetters = new LinkedHashSet<>();

        // Add key2 letters first, ensuring no duplicates
        for (char ch : key2.toCharArray()) {
            uniqueLetters.add(ch);
        }

        // Add remaining letters of the alphabet in order
        for (char ch : ALPHABET.toCharArray()) {
            uniqueLetters.add(ch);
        }

        // Convert the set to a string to form the permuted alphabet
        StringBuilder permutedAlphabet = new StringBuilder();
        for (char ch : uniqueLetters) {
            permutedAlphabet.append(ch);
        }

        return permutedAlphabet.toString();
    }
}
```

Output:

Encryption:

```
Choose operation (E for encryption, D for decryption): e
Enter key 1 (1-25): 3
Enter key 2 (a word with unique letters, minimum length 7): cryptog
Enter the message: message
Encrypted message: SIWWEDI
```

Decryption:

```
Choose operation (E for encryption, D for decryption): d
Enter key 1 (1-25): 3
Enter key 2 (a word with unique letters, minimum length 7): cryptog
Enter the message: SIWWEDI
Decrypted message: MESSAGE
```

Conclusion:

In this lab, we successfully implemented both the Caesar Cipher and an enhanced version of it, the Caesar Cipher with Permutation. The basic Caesar Cipher demonstrated the fundamental principles of a substitution cipher, where each character in the plaintext is shifted by a set number of positions to produce the ciphertext. This simple method provided insight into classic encryption techniques and their limitations, particularly in terms of predictability and ease of decryption when faced with brute force attacks. The addition of a permutation step enhanced the Caesar Cipher by introducing another layer of complexity. By permuting characters before applying the Caesar shift, we achieved greater obfuscation, making the ciphertext more resistant to simple frequency analysis and direct brute force attacks. This implementation highlights how combining cryptographic techniques can strengthen security and serves as a foundation for understanding more advanced encryption methods.