MINISTERUL EDUCAȚIEI ȘI CERCETĂRII
AL REPUBLICII MOLDOVA

Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică şi Microelectronică
Departamentul Inginerie Software și Automatică

Artur Țugui, FAF-231

Report

Laboratory Work No. 1

on Cryptography and Security

Checked by:
Maia Zaica, university assistant
ISA, FCIM, UTM

Chișinău – 2025

1. Theoretical Background

Cifrul lui Cesar

Cifrul lui Cesar (sau Cezar). În acest cifru fiecare literă a textului clar este înlocuită cu o nouă literă obținută printr-o deplasare alfabetică. Cheia secretă k, care este aceeași la criptare cât și la decriptare, constă în numărul care indică deplasarea alfabetică, adică $k \in \{1, 2, 3, \ldots, n-1\}$, unde n este lungimea alfabetului. Criptarea și decriptarea mesajului cu cifrul Cezar poate fi definită de formulele

$$c = e_k(x) = x + k \pmod{n},$$
$$m = d_k(y) = y - k \pmod{n},$$

unde x și y sunt reprezentarea numerică a caracterului respectiv din textul clar m și din criptograma c. Funcția numită Modulo (a mod b) returnează restul împărțirii numărului întreg a la numărul întreg b.

Această metodă de criptare este numită așa după Iulius Cezar, care o folosea pentru a comunica cu generalii săi, folosind cheia $k = 3$ (tabelul **??**).

De exemplu, pentru $k = 3$ avem

$$e_k(S) = 18 + 3 \pmod{26} = 21 = V,$$
$$d_k(V) = 21 - 3 \pmod{26} = 18 = S.$$

În acest caz pentru $m =$ "cifrul cezar", obținem $c =$ "fliuxo fhcdu".

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	0	1	2
D	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	*A*	*B*	*C*

Tabelul 1. *Cifrul Cezar cu cheia k*=3

Cifrul lui Cezar este foarte ușor de spart, deci este un cifru foarte slab. Astfel, un criptanalist poate obține textul clar prin încercarea tuturor celor 25 de chei. Nu se știe cât de util era cifrul Cezar în timpul când era folosit de către cel de la care îi provine

numele, dar este probabil ca el să fi fost destul de sigur, atât timp cât numai câțiva dintre inamicii lui Cezar erau în stare să scrie și să citească, dar mai ales să cunoască concepte de criptanaliză.

Cifrul lui Cesar + permutare

Având în vedere criptorezistența scăzută a cifrului Cezar, datorată în primul rând spațiului de chei, care constă doar din 25 de chei diferite pentru alfabetul latin, acesta poate fi spart prin încercarea consecutivă a tuturor cheilor. Dacă mesajul a fost criptat cu cifrul Cezar, atunci una dintre chei ne va da un text citibil în limba în care a fost scris mesajul.

Spre exemplu, dacă

$$m = \text{BRUTE FORCE ATTACK}$$

este un mesaj scris în limba engleză și a fost criptat cu cheia

$$k = 17,$$

obținem criptograma

$$c = \text{SILKVWFITVRKKRTB}$$

Dacă criptanalistul interceptează mesajul criptat și parcurge toate cheile 1, 2, . . . , 25 — va obține următoarele:

După cum se poate observa — doar textul obținut prin utiliza cheii $k = 17$ este unul cu sens în limba engleză, deci mesajul corespunzător criptogramei este

$$m = \text{BRUTEFORCEATTACK}.$$

Pentru a spori criptorezistența cifrului Cezar se poate de aplicat o permutare a alfabetului prin aplicarea unui cuvânt-cheie (a nu se confunda cu cheia de bază a cifrului). Această cheie poate fi orice consecutivitate de litere a alfabetului — fie un cuvânt din vocabular, fie unul fără sens.

Fie cheia a doua este $k_2 =$ cryptography. Aplicăm această cheie asupra alfabetului

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

și obținem:

C R Y P T O G A H B D E F I J K M L N Q S U V W X Z

Această ordine nouă am obținut-o prin plasarea literelor lui k_2 la început, apoi urmează celelalte litere ale alfabetului în ordinea lor naturală. Vom ține cont de faptul că literele nu se vor repeta, adică dacă litera se întâlnește de câteva ori, ea se plasează doar o singură dată.

În continuare se aplică cifrul Cezar, ținând cont de noua ordine a alfabetului:

Deoarece există $26! = 403291461126605635584000000$, numărul de chei pentru această versiune a algoritmului va fi

$$26! \times 25 = 10082286528165140889600000000,$$

ceea ce complică spargerea prin metoda exhaustivă, dar nu ne salvează de atacul prin analiza frecvențelor.

2. Conditions of the Problems

Sarcina 1.1

De implementat algoritmul Cezar pentru alfabetul limbii engleze în unul din limbajele de programare. Utilizați doar codificarea literelor cum este arătat în tabelul 1 (nu se permite de folosit codificările specificate în limbajul de programare, de ex. ASCII sau Unicode). Valorile cheii vor fi cuprinse între 1 și 25 inclusiv și nu se permit alte valori. Valorile caracterelor textului sunt cuprinse între 'A' și 'Z', 'a' și 'z' și nu sunt premise alte valori. În cazul în care utilizatorul introduce alte valori — i se va sugera diapazonul corect. Înainte de criptare textul va fi transformat în majuscule și vor fi eliminate spațiile. Utilizatorul va putea alege operația — criptare sau decriptare, va putea introduce cheia, mesajul sau criptograma și va obține respectiv criptograma sau mesajul decriptat.

Sarcina 1.2

De implementat algoritmul Cezar cu 2 chei, cu păstrarea condițiilor exprimate în Sarcina 1.1. În plus, cheia 2 trebuie să conțină doar litere ale alfabetului latin, și să aibă o lungime nu mai mică de 7.

3. Program Code

I made the program in Java.

Sarcina 1.1

```
// InputProcessor.java
package task_1;

public class InputProcessor {
    public static String prepareInput(String m) {
        // eliminate spaces
        String noSpace_m = m.replaceAll(" ", "");

        // validate
        // isInputValid(noSpace_m);

        // toUppercase
        return noSpace_m.toUpperCase();
    }

    public static boolean isInputValid(String input) {
        String noSpace_m = input.replaceAll(" ", "");

        for (char c : noSpace_m.toCharArray()) {
            if (!((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z'))) {
                System.out.println("Invalid character: " + c);
                return false;
            }
        }
        return true;
    }
}



// Main.java
package task_1;
```

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        showMenu(scanner);
    }

    public static void showMenu(Scanner scanner) {
        char choice;

        while (true) {
            System.out.println("Enter the given number to:");
            System.out.println("\t1. Encrypt a message with Caesar Cipher
            System.out.println("\t2. Decrypt a message encrypted with Cae

            String input = scanner.nextLine();
            if (input.isEmpty()) {
                continue;
            }
            choice = input.charAt(0);

            switch (choice) {
                case '1':
                    showEncryptionMenu(scanner);
                    break;
                case '2':
                    showDecryptionMenu(scanner);
                    break;
                default:
                    System.out.println("Invalid choice, try again");
```

```
                continue;
            }

            break; // exit loop after valid choice
        }
    }

    public static void showEncryptionMenu(Scanner scanner) {
        System.out.println("Enter the message to be encrypted:\n");
        String m;

        while (true) {
            m = scanner.nextLine();
            if (m.isEmpty()) {
                continue;
            }

            if (!InputProcessor.isInputValid(m)) {
                System.out.print("Try again");
                continue;
            }

            break;
        }

        System.out.println("Enter the key (1 - 25):\n");
        int k;

        while (true) {
            k = scanner.nextInt();

            if (k < 1 || k > 25) {
                System.out.println("Invalid input, try again");
                continue;
```

```
            }

            break;
        }

        String prepared_m = InputProcessor.prepareInput(m);

        String c = RegularCaesarCipher.encrypt(prepared_m, k);

        System.out.println("Encrypted message: " + c);
    }

    public static void showDecryptionMenu(Scanner scanner) {
        System.out.println("Enter the message to be decrypted:\n");
        String c;

        while (true) {
            c = scanner.nextLine();
            if (c.isEmpty()) {
                continue;
            }

            if (!InputProcessor.isInputValid(c)) {
                System.out.print("Try again");
                continue;
            }

            break;
        }

        System.out.println("Enter the key (1 - 25):\n");
        int k;

        while (true) {
```

```
            k = scanner.nextInt();

            if (k < 1 || k > 25) {
                System.out.println("Invalid input, try again");
                continue;
            }

            break;
        }

        String prepared_c = InputProcessor.prepareInput(c);

        String m = RegularCaesarCipher.decrypt(prepared_c, k);

        System.out.println("Decrypted message: " + m);
    }

}
```

```
// Mapper.java
package task_1;

public class Mapper {
    public static String decode(int[] encoded_m) {
        StringBuilder decoded_m = new StringBuilder();

        for (int j : encoded_m) {
            char c = toChar(j);
            decoded_m.append(c);
        }

        return decoded_m.toString();
```

```
    }

    public static int[] encode(String upper_m) {
        int[] encoded_m = new int[upper_m.length()];

        for (int i = 0; i < upper_m.length(); i++) {
            encoded_m[i] = toInt(upper_m.charAt(i));
        }

        return encoded_m;
    }

    public static int toInt(char chr) {
        if (Character.isUpperCase(chr)) {
            return chr - 65;
        } else if (Character.isLowerCase(chr)) {
            return chr - 97;
        } else {
            return -1;
        }
    }

    public static char toChar(int i) {
        if (i >= 0 && i <= 25) {
            return (char) (i + 65);
        } else {
            return '~';
        }
    }
}



// RegularCaesarCipher.java
```

```
package task_1;

public class RegularCaesarCipher {
    public static String encrypt(String upper_m, int k) {
        // encode
        int[] encoded_m = Mapper.encode(upper_m);

        // encrypt
        for (int i = 0; i < encoded_m.length; i++) {
            encoded_m[i] = (encoded_m[i] + k) % 26;
        }

        // decode
        return Mapper.decode(encoded_m);
    }

    public static String decrypt(String upper_c, int k) {
        // encode
        int[] encoded_c = Mapper.encode(upper_c);

        // decrypt
        for (int i = 0; i < encoded_c.length; i++) {
            encoded_c[i] = (encoded_c[i] - k) % 26;
            if (encoded_c[i] < 0) {
                encoded_c[i] += 26;
            }
        }

        // decode
        return Mapper.decode(encoded_c);
    }

}
```

Sarcina 1.2

```
// CaesarCipher2Keys.java
package task_2;

import java.util.ArrayList;
import java.util.List;

public class CaesarCipher2Keys {
    public static String encrypt(String upper_m, int k, String upper_key_

        List<Character> new_alphabet = new ArrayList<Character>();
        for (char c : upper_key_word.toCharArray()) {
            if (!new_alphabet.contains(c)) {
                new_alphabet.add(c);
            }
        }

        for (int i = 0; i < 26; i++) {
            char c = (char) ('A' + i);
            if (!new_alphabet.contains(c)) {
                new_alphabet.add(c);
            }
        }

        System.out.println("Permuted alphabet: " + new_alphabet);

        // encode
        int[] encoded_m = Mapper.encode(upper_m, new_alphabet);

        // encrypt
        for (int i = 0; i < encoded_m.length; i++) {
            encoded_m[i] = (encoded_m[i] + k) % 26;
        }
```

```
    // decode
    return Mapper.decode(encoded_m, new_alphabet);
}

public static String decrypt(String upper_c, int k, String upper_key_

    List<Character> new_alphabet = new ArrayList<Character>();
    for (char c : upper_key_word.toCharArray()) {
        if (!new_alphabet.contains(c)) {
            new_alphabet.add(c);
        }
    }

    for (int i = 0; i < 26; i++) {
        char c = (char) ('A' + i);
        if (!new_alphabet.contains(c)) {
            new_alphabet.add(c);
        }
    }

    System.out.println("Permuted alphabet: " + new_alphabet);

    // encode
    int[] encoded_c = Mapper.encode(upper_c, new_alphabet);

    // decrypt
    for (int i = 0; i < encoded_c.length; i++) {
        encoded_c[i] = (encoded_c[i] - k) % 26;
        if (encoded_c[i] < 0) {
            encoded_c[i] += 26;
        }
    }
```

```
        // decode
        return Mapper.decode(encoded_c, new_alphabet);
    }

}
```

```
// InputProcessor.java
package task_2;

public class InputProcessor {
    public static String prepareInput(String m) {
        // eliminate spaces
        String noSpace_m = m.replaceAll(" ", "");

        // validate
        // isInputValid(noSpace_m);

        // toUppercase
        return noSpace_m.toUpperCase();
    }

    public static boolean isInputValid(String input) {
        String noSpace_m = input.replaceAll(" ", "");

        for (char c : noSpace_m.toCharArray()) {
            if (!((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z'))) {
                System.out.println("Invalid character: " + c);
                return false;
            }
        }
        return true;
    }
```

```
    public static boolean isKeyWordValid(String input) {
        if (input.contains(" ")) {
            System.out.println("The key word cannot contain spaces");
            return false;
        }

        if (input.length() < 7) {
            System.out.println("The key should be at least 7 letters long
            return false;
        }

        for (char c : input.toCharArray()) {
            if (!((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z'))) {
                System.out.println("Invalid character: " + c);
                return false;
            }
        }

        return true;
    }
}
```

```
// Main.java
package task_2;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
```

```
    Scanner scanner = new Scanner(System.in);
    showMenu(scanner);
}

public static void showMenu(Scanner scanner) {
    char choice;

    while (true) {
        System.out.println("Enter the given number to:");
        System.out.println("\t1. Encrypt a message with Caesar Cipher
        System.out.println("\t2. Decrypt a message encrypted with Cae

        String input = scanner.nextLine();
        if (input.isEmpty()) {
            continue;
        }
        choice = input.charAt(0);

        switch (choice) {
            case '1':
                showEncryptionMenu(scanner);
                break;
            case '2':
                showDecryptionMenu(scanner);
                break;
            default:
                System.out.println("Invalid choice, try again");
                continue;
        }

        break; // exit loop after valid choice
    }
}
```

```
public static void showEncryptionMenu(Scanner scanner) {
    System.out.println("Enter the message to be encrypted:\n");
    String m;

    while (true) {
        m = scanner.nextLine();
        if (m.isEmpty()) {
            continue;
        }

        if (!InputProcessor.isInputValid(m)) {
            System.out.print("Try again");
            continue;
        }

        break;
    }

    System.out.println("Enter the numerical key (1 - 25):\n");
    int k;

    while (true) {
        k = scanner.nextInt();

        if (k < 1 || k > 25) {
            System.out.println("Invalid input, try again");
            continue;
        }

        break;
    }

    System.out.println("Enter the key word (>= 7 letters):\n");
```

```
        String key_word;

        while (true) {
            key_word = scanner.nextLine();
            if (key_word.isEmpty()) {
                continue;
            }

            if (!InputProcessor.isKeyWordValid(key_word)) {
                System.out.print("Try again");
                continue;
            }

            break;
        }

        String prepared_m = InputProcessor.prepareInput(m);

        String upper_key_word = key_word.toUpperCase();
        String c = CaesarCipher2Keys.encrypt(prepared_m, k, upper_key_wor

        System.out.println("Encrypted message: " + c);
    }

    public static void showDecryptionMenu(Scanner scanner) {
        System.out.println("Enter the message to be decrypted:\n");
        String c;

        while (true) {
            c = scanner.nextLine();
            if (c.isEmpty()) {
                continue;
            }
```

```
        if (!InputProcessor.isInputValid(c)) {
            System.out.print("Try again");
            continue;
        }

        break;
    }

    System.out.println("Enter the key (1 - 25):\n");
    int k;

    while (true) {
        k = scanner.nextInt();

        if (k < 1 || k > 25) {
            System.out.println("Invalid input, try again");
            continue;
        }

        break;
    }

    System.out.println("Enter the key word (>= 7 letters):\n");
    String key_word;

    while (true) {
        key_word = scanner.nextLine();
        if (key_word.isEmpty()) {
            continue;
        }

        if (!InputProcessor.isKeyWordValid(key_word)) {
            System.out.print("Try again");
            continue;
```

```
            }

            break;
        }

        String prepared_c = InputProcessor.prepareInput(c);

        String upper_key_word = key_word.toUpperCase();
        String m = CaesarCipher2Keys.decrypt(prepared_c, k, upper_key_wor

        System.out.println("Decrypted message: " + m);
    }

}



// Mapper.java
package task_2;

import java.util.ArrayList;
import java.util.List;

public class Mapper {
    public static String decode(int[] encoded_m, List<Character> new_alph
        StringBuilder decoded_m = new StringBuilder();

        for (int j : encoded_m) {
            char c = toChar(j, new_alphabet);
            decoded_m.append(c);
        }

        return decoded_m.toString();
    }
```

```
    public static int[] encode(String upper_m, List<Character> new_alphab
        int[] encoded_m = new int[upper_m.length()];

        for (int i = 0; i < upper_m.length(); i++) {
            encoded_m[i] = toInt(upper_m.charAt(i), new_alphabet);
        }

        return encoded_m;
    }

    public static int toInt(char chr, List<Character> new_alphabet) {
        return new_alphabet.indexOf(chr);
    }

    public static char toChar(int i, List<Character> new_alphabet) {
        return new_alphabet.get(i);
    }
}
```

4. Program Execution Screenshots

```
Enter the given number to:
    1. Encrypt a message with Caesar Cipher
    2. Decrypt a message encrypted with Caesar Cipher
1
Enter the message to be encrypted:

Cifrul Caesar
Enter the key (1 - 25):

3
Encrypted message: FLIUXOFDHVDU

Process finished with exit code 0
```

Figure 1: Encrypting with Caesar Cipher

```
Enter the given number to:
    1. Encrypt a message with Caesar Cipher
    2. Decrypt a message encrypted with Caesar Cipher
2
Enter the message to be decrypted:

fliuxofdhvdu
Enter the key (1 - 25):

3
Decrypted message: CIFRULCAESAR

Process finished with exit code 0
```

Figure 2: Decrypting with Caesar Cipher

```
Enter the given number to:
    1. Encrypt a message with Caesar Cipher (2 keys)
    2. Decrypt a message encrypted with Caesar Cipher (2 keys)
1
Enter the message to be encrypted:

Brute Force Attack
Enter the numerical key (1 - 25):

3
Enter the key word (>= 7 letters):

Cryptography
Permuted alphabet: [C, R, Y, P, T, O, G, A, H, B, D, E, F, I, J, K, L, M, N, Q, S, U, V, W, X, Z]
Encrypted message: FTXAJKHTPJDAADPN

Process finished with exit code 0
```

Figure 3: Encrypting with Caesar Cipher with permutation

```
Enter the given number to:
    1. Encrypt a message with Caesar Cipher (2 keys)
    2. Decrypt a message encrypted with Caesar Cipher (2 keys)
2
Enter the message to be decrypted:

FTXAJKHTPJDAADPN
Enter the key (1 - 25):

3
Enter the key word (>= 7 letters):

Cryptography
Permuted alphabet: [C, R, Y, P, T, O, G, A, H, B, D, E, F, I, J, K, L, M, N, Q, S, U, V, W, X, Z]
Decrypted message: BRUTEFORCEATTACK

Process finished with exit code 0
```

Figure 4: Decrypting with Caesar Cipher with permutation

5. Conclusions and Insights Gained

- Regular Caesar Cipher can be easily deciphered with a brute force attack due to the small amount of keys
- Caesar Cipher with permutation is immune to brute force attack, but not to frequency analysis
- Java does not handle modulo operation for negative numbers correctly

- Caesar Cipher is easily programmable in Java
- Caesar Cipher follows the Kerckhoffs Principle, that the secrecy must be in the key and not in the algorithm