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. 3
Discipline: Cryptography and Security

Elaborated: Grigoraș Dumitru FAF-221,

Checked: Dumitru Nirca asist. univ.,

Chișinău 2024

Topic: Caesar Cipher

Tasks 1:

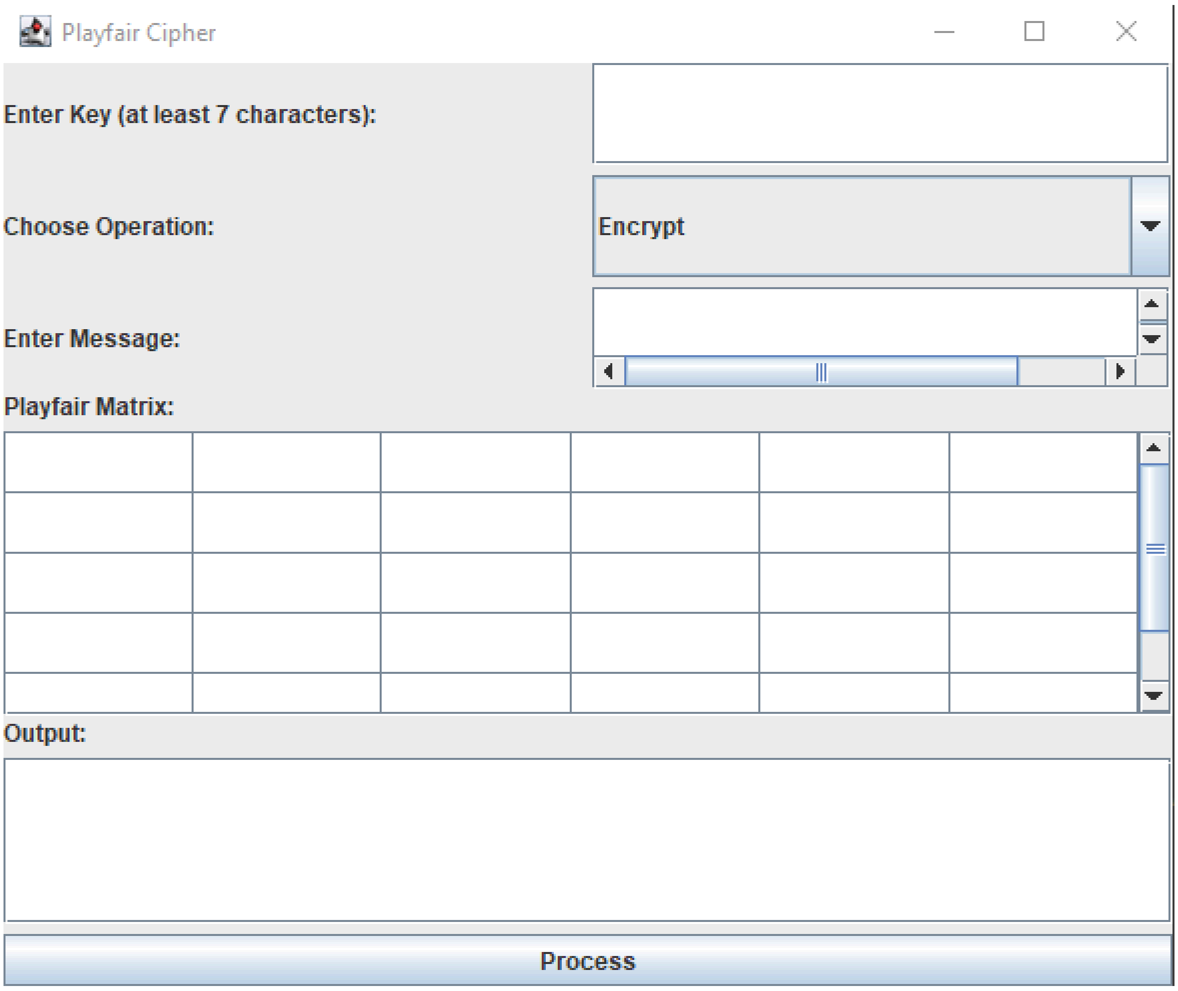
De implementat algoritmul Playfair în unul din limbajele de programare pentru mesaje în limba română (31 de litere). Valorile caracterelor textului sunt cuprinse între ‘A’ și ’Z’, ’a’ și ’z’ și nu sunt premise alte valori. În cazul în care utilizatorul introduce alte valori - i se va sugera diapazonul corect al caracterelor. Lungimea cheii nu trebuie să fie mai mică de 7. Utilizatorul va putea alege operația - criptare sau decriptare, va putea introduce cheia, mesajul sau criptograma și va obține criptograma sau mesajul decriptat. Faza finală de adăugare a spațiilor noi, în funcție de limba folosită și de logica mesajului – se va face manual.

Implementation

```java
public class PlayfairCipher {
  private static final String ALPHABET = "AĂÂBCDEFGHIÎJKLMNOPQRSŞTŢUVWXYZ";
   private static final int SIZE = 6; // 6x6 matrix for Romanian alphabet
with 31 letters

   private char[][] matrix;
   private String key;

   public PlayfairCipher(String key) {
       if (key.length() < 7) {
           throw new IllegalArgumentException("Key length must be at least
7 characters.");
       }
       this.key = prepareKey(key.toUpperCase());
       System.out.println("Prepared Key: " + this.key);
       this.matrix = generateMatrix(this.key);
       printMatrix();
   }

   private String prepareKey(String key) {
       StringBuilder preparedKey = new StringBuilder();
       HashSet<Character> seen = new HashSet<>();
       for (char c : key.toCharArray()) {
           if (ALPHABET.contains(String.valueOf(c)) && seen.add(c)) {
               preparedKey.append(c);
           }
       }
       for (char c : ALPHABET.toCharArray()) {
           if (seen.add(c)) {
               preparedKey.append(c);
           }
       }
       return preparedKey.toString();
   }

   private char[][] generateMatrix(String key) {
       char[][] matrix = new char[SIZE][SIZE];
       int k = 0;
       for (int i = 0; i < SIZE; i++) {
           for (int j = 0; j < SIZE; j++) {
               if (k < key.length()) {
                   matrix[i][j] = key.charAt(k++);
               } else {
                   matrix[i][j] = '-';
               }
           }
       }
       return matrix;
   }

   private void printMatrix() {
       System.out.println("Playfair Matrix:");
       for (char[] row : matrix) {
           for (char c : row) {
               System.out.print(c + " ");
           }
           System.out.println();
       }
   }

   public String encrypt(String message) {
       System.out.println("Encrypting message: " + message);
       return process(message, true);
   }
   public String decrypt(String cryptogram) {
       System.out.println("Decrypting cryptogram: " + cryptogram);
       return process(cryptogram, false);
   }

   private String process(String text, boolean isEncryption) {
       text = prepareText(text.toUpperCase());
       System.out.println("Prepared Text: " + text);
       StringBuilder result = new StringBuilder();

       for (int i = 0; i < text.length(); i += 2) {
           char a = text.charAt(i);
           char b = text.charAt(i + 1);
           int[] posA = findPosition(a);
           int[] posB = findPosition(b);
           System.out.println("Pair: " + a + " " + b);
           System.out.println("Positions: (" + posA[0] + "," + posA[1] + ") and (" + posB[0] + "," + posB[1] + ")");
           if (posA[0] == posB[0]) { // Same row, wrap around horizontally
               result.append(matrix[posA[0]][(posA[1] + (isEncryption ? 1 : SIZE - 1)) % SIZE]);
               result.append(matrix[posB[0]][(posB[1] + (isEncryption ? 1 : SIZE - 1)) % SIZE]);
           } else if (posA[1] == posB[1]) { // Same column, wrap around vertically
               result.append(matrix[(posA[0] + (isEncryption ? 1 : SIZE - 1)) % SIZE][posA[1]]);
               result.append(matrix[(posB[0] + (isEncryption ? 1 : SIZE - 1)) % SIZE][posB[1]]);
           } else { // Rectangle swap, ensure no '-' is used
               char charA = matrix[posA[0]][posB[1]];
               char charB = matrix[posB[0]][posA[1]];
               // Handle empty cell issue by adjusting the position if '-' is encountered
               if (charA == '-') {
                   charA = matrix[0][posB[1]];
               }
               if (charB == '-') {
                   charB = matrix[0][posA[1]];
               }

               result.append(charA);
               result.append(charB);
           }

           System.out.println("Result after processing pair: " +
result.toString());
       }

       System.out.println("Final " + (isEncryption ? "encrypted" :
"decrypted") + " text: " + result.toString());
       return result.toString();
   }



   private String prepareText(String text) {
       StringBuilder prepared = new StringBuilder();
       for (char c : text.toCharArray()) {
           if (ALPHABET.contains(String.valueOf(c))) {
               prepared.append(c);
           } else {
               System.out.println("Invalid character '" + c + "'. Please
use characters within A-Z, a-z.");
           }
       }
       if (prepared.length() % 2 != 0) {
           prepared.append('X'); // Padding for odd length
           System.out.println("Odd length detected, padding with 'X'");
       }
       return prepared.toString();
   }

   private int[] findPosition(char c) {
       for (int i = 0; i < SIZE; i++) {
           for (int j = 0; j < SIZE; j++) {
               if (matrix[i][j] == c) {
                   return new int[]{i, j};
               }
           }
       }
       return null; // Should not happen if input is validated
   }
```

Output:

Visual Implementation:



Enter Key (at least 7 characters):

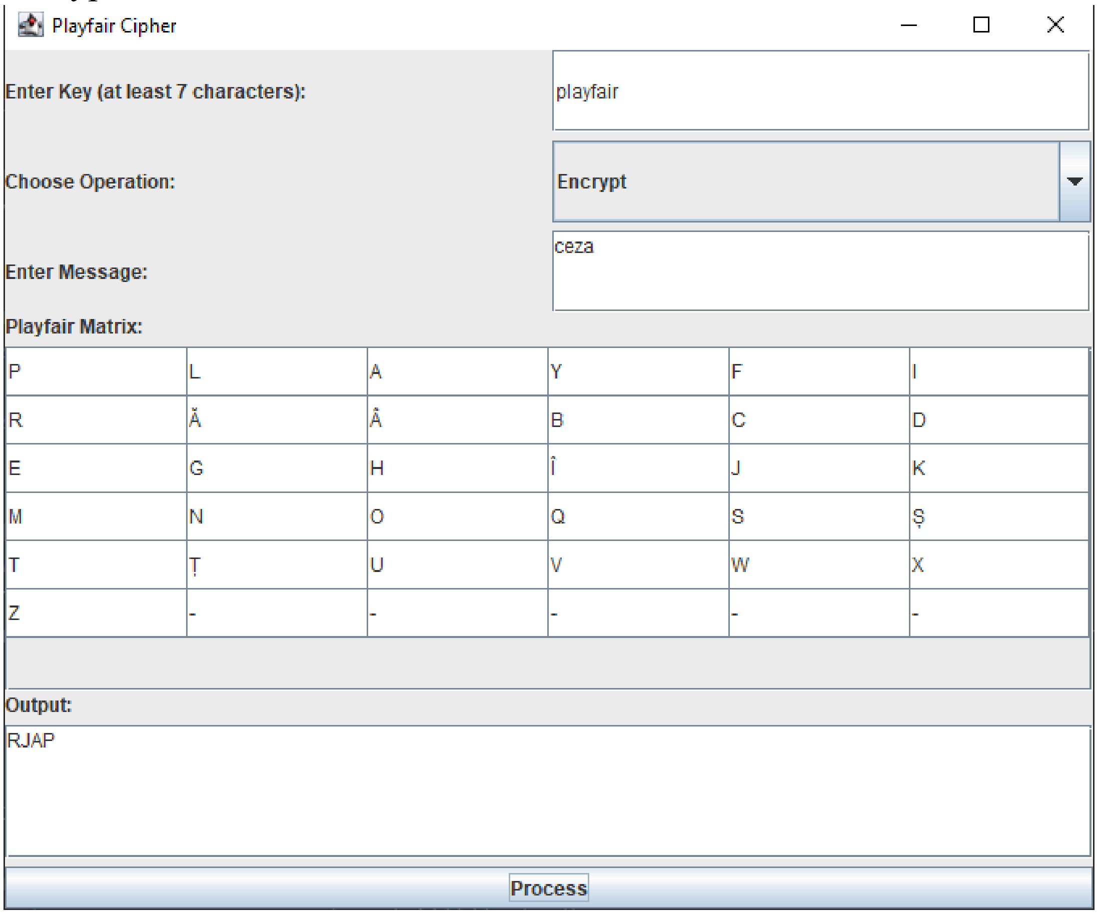
Choose Operation: Encrypt

Enter Message:

Playfair Matrix:

Output:

Encryption:



Enter Key (at least 7 characters): playfair

Choose Operation: Encrypt

Enter Message: ceza

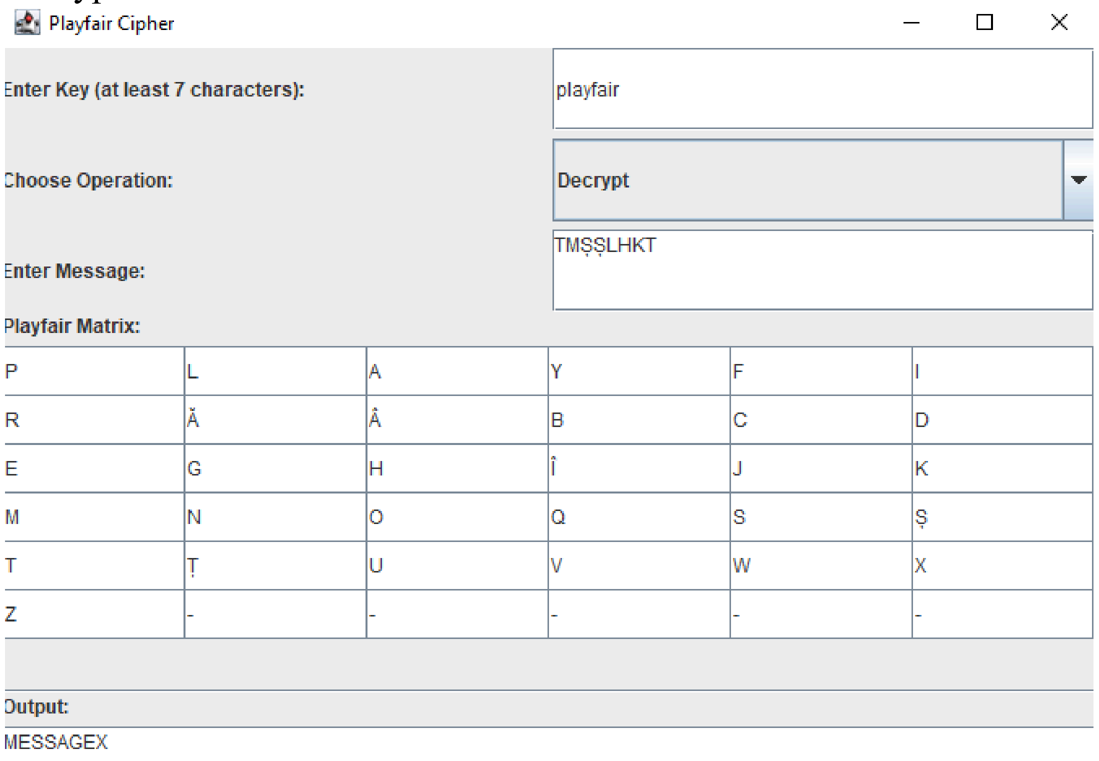
Playfair Matrix:

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

Output:

```
RJAP
```

Decryption:



Enter Key (at least 7 characters): playfair

Choose Operation: Decrypt

Enter Message: TMȘȘLHKT

Playfair Matrix:

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

Output:

```
MESSAGEX
```

Conclusion:

In this lab, I implemented the Playfair cipher algorithm for the Romanian alphabet, a substitution cipher that enhances security by working with digraphs (pairs of letters). We covered the full process, from key preparation and matrix generation to text encryption and decryption. The program ensures that only valid characters from the Romanian alphabet are processed, and it handles various edge cases such as odd-length messages by padding them with an 'X'.

Additionally, I developed a graphical user interface (GUI) to provide a user-friendly way to interact with the cipher. The GUI includes components for entering the key, selecting the operation (encrypt or decrypt), inputting the message, and displaying the encrypted or decrypted result. Moreover, the Playfair cipher matrix is visually represented in a table, offering an intuitive understanding of the underlying transformation.