Ministerul Educaţiei şi Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work 3:
Cryptography and Security
Playfair Algorithm

Elaborated:
st. gr. FAF-213 Afteni Maria

Verified:
asist. univ. Cătălin Mîțu

Chişinău - 2023

ALGORITHM ANALYSIS

Objective: Polyalphabetic substitution

Tasks:

To implement the Playfair algorithm in one of the programming languages for messages in Romanian (31 letters). The text character values are between 'A' and 'Z', 'a' and 'z' and no other values are assumed. If the user enters other values - the correct range of characters will be suggested. The length of the key must not be less than 7. The user will be able to choose the operation - encryption or decryption, he will be able to enter the key, message or cryptogram and get the decrypted cryptogram or message. The final phase of adding new spaces, depending on the language used and the logic of the message – will be done manually.

IMPLEMENTATION

First step in this laboratory work is to reform the source text from the user. To do this I created the reform_source function. It converts the text to uppercase, removes spaces, and ensures that the length of the text is even by appending 'X' if necessary. It also handles the case where consecutive characters are the same, inserting 'X' between them.

```
def reform_source(source):
    source = source.upper().replace(' ', '')
    if len(source) % 2 == 1:
        source += 'X'

    i = 0
    reformed_source = ''

    while i < len(source) - 1:
        if source[i] == source[i + 1]:
            reformed_source += source[i] + 'X'
            i += 1

        else:
            reformed_source += source[i] + source[i + 1]
            i += 2

    return reformed_source
```

Then, using the key that is given by the user, we need to create a matrix.

```
def create_matrix(key):
    matrix = []
    for char in key:
        if char not in matrix:
            matrix.append(char)
    for char in alphabet:
        if char not in matrix:
            matrix.append(char)
    return [matrix[i:i + 6] for i in range(0, 36, 6)]
```

The create_matrix function generates a 6x6 matrix for the Playfair cipher. It takes the key as input and constructs the matrix by appending characters from the key and then filling in the remaining characters from the alphabet.

```
def find_position(matrix, char):
    for x in range(6):
        for y in range(6):
            if matrix[x][y] == char:
                return x, y
    return
```

To determine the position of a character in the matrix I created the find_position function. The function returns the row and column indices of the needed . If the character is not found in the matrix, it returns None.

```
def encrypt(matrix, source):
encrypted_text = ""
i = 0
while i < len(source):
    x1, y1 = find_position(matrix, source[i])
    x2, y2 = find_position(matrix, source[i + 1])

    if x1 == x2:  # Same row
        encrypted_text += matrix[x1][(y1 + 1) % 6] + matrix[x2][(y2 + 1) % 6]

    elif y1 == y2:  # Same column
        encrypted_text += matrix[(x1 + 1) % 6][y1] + matrix[(x2 + 1) % 6][y2]

    else:
        encrypted_text += matrix[x1][y2] + matrix[x2][y1]

    i += 2

return encrypted_text
```

The encrypt function is the fundamental part of a Playfair cipher implementation. The function initializes an empty string called encrypted_text to store the encrypted result. It then iterates through the source text in pairs, using variables i and i+1, to determine the positions of the characters in the matrix. The function applies the Playfair cipher rules to transform the characters in the following ways:

- If the characters are in the same row, they are shifted one position to the right within the same row, taking into account wrap-around behavior using modulo.
- If the characters are in the same column, they are shifted one position down within the same column, again considering wrap-around behavior.
- If the characters are neither in the same row nor the same column, they are replaced with characters from opposite corners of the matrix.

```
def decrypt(matrix, source):
 decrypted_text = ""

 i = 0
 while i < len(source):
```

```
        x1, y1 = find_position(matrix, source[i])
        x2, y2 = find_position(matrix, source[i + 1])

        if x1 == x2:
            decrypted_text += matrix[x1][(y1 - 1) % 6] + matrix[x2][(y2 - 1) % 6]

        elif y1 == y2:
            decrypted_text += matrix[(x1 - 1) % 6][y1] + matrix[(x2 - 1) % 6][y2]
        else:
            decrypted_text += matrix[x1][y2] + matrix[x2][y1]

        if decrypted_text[-1] == "X":
            decrypted_text = decrypted_text[:-1]

        i += 2

    return decrypted_text
```

The decryption function operated in the opposite way. It takes ciphertext and transforms it back into plaintext using the reverse of the Playfair cipher's rules. The function shifts characters within the matrix (left, up, or diagonally in the opposite direction) to retrieve the original plaintext.

CONCLUSIONS

In conclusion, this laboratory work involved the implementation of the Playfair cipher, a classical encryption technique, that offered a unique and interesting approach to securing messages. Through this project, I've gained valuable insights into both the cryptographic and programming aspects of secure communication. This project has deepened my understanding of cryptography, specifically the principles behind symmetric key ciphers. The Playfair cipher, though less commonly used today, exemplifies fundamental cryptographic concepts like substitution and transposition. While the Playfair cipher has its limitations, it serves as a stepping stone in the broader landscape of information security, and the principles learned here can be applied to more advanced cryptographic algorithms in the future.