MINISTRY OF EDUCATION, CULTURE, AND RESEARCH OF THE REPUBLIC OF MOLDOVA

Technical University of Moldova

Faculty of Computers, Informatics, and Microelectronics

Department of Software Engineering and Automation

Report

Laboratory Work No. ***2***

at Cryptography and Security

Topic: *Polyalphabetic Ciphers. The Playfair Cipher*

Realized by:

Ciprian Moisenco, FAF - 232

Checked by:

Zaica Maia, university assistant

Chişinău – 2025

Contents

1 Purpose of the Laboratory Work

The purpose of this laboratory work is to study and implement the **Playfair Cipher** for messages written in the **Romanian language** (31 letters). By completing this task, we aim to understand the operational principles of polyalphabetic substitution ciphers, such as the use of encryption matrices and rules for encrypting character pairs (block ciphers). Furthermore, the exercise develops programming skills by implementing data validation mechanisms and adhering to the minimum key length condition.

2 Laboratory Task (3.1)

The assigned task (for students with an odd registration number) is **Task 3.1**:

1. Implementation of the **Playfair** algorithm in a programming language (Python) for messages in the **Romanian language** (31 letters).
2. The character values of the text must be between 'A' and 'Z', 'a' and 'z' (including the specific Romanian characters: Ă, Â, Î, Ș, Ț). No other values are permitted. If the user enters invalid values, they will be prompted with the correct character range.
3. The key length must not be less than **7** characters.
4. The user must be able to choose the operation (**encryption** or **decryption**), enter the key, input the message or ciphertext, and receive the corresponding ciphertext or decrypted message.
5. The final phase of manually adding new spaces, based on the language used and the logic of the message, will be done **manually**.

3 Theoretical Considerations

3.1 Polyalphabetic Ciphers

The weakness of monoalphabetic ciphers lies in the fact that their frequency distribution reflects the distribution of the alphabet used. A cryptographically more secure cipher exhibits a distribution that is as regular as possible, offering no exploitable information to the cryptanalyst.

One way to flatten this distribution is by combining high and low frequencies. **Polyalphabetic ciphers** use character substitution that varies throughout the text based on various parameters (position, context, etc.), unlike monoalphabetic ciphers where a character's substitution is fixed. This leads to a significantly larger keyspace.

3.2 The Playfair Encryption Algorithm

The Playfair cipher, invented by Charles Wheatstone and popularized by Baron Lyon Playfair, is a polyalphabetic substitution cipher that uses the modern principles of **block ciphers** (by encrypting pairs of letters). Although considered outdated in modern cryptography, it was used by the British military in various wars and provides an intuitive understanding of early modern cryptography.

Algorithm Steps

Playfair encryption involves three main steps:

1. Preparation of the plaintext.
2. Construction of the encryption matrix (a 5×5 matrix for the English alphabet).
3. Construction of the encrypted message.

Text Preparation

The plaintext is processed as follows:

- All letters are converted to **uppercase**, with no spaces or punctuation.
- The text is divided into **pairs** of letters.
- A filler letter ('X' or 'Q' is common) is inserted between double letters (e.g., $SS \rightarrow SXS$), or at the end if the text has an odd length, to ensure all blocks are two letters long and contain no identical characters.

Matrix Construction (The Key)

A key (a word or phrase) is used to construct a 5×5 matrix (or 6×6 for the 31-letter Romanian alphabet).

- First, repeated letters are removed from the encryption key.
- The matrix is filled row-by-row, starting from the top-left corner, with the unique letters of the key.
- The remaining spaces are filled with the rest of the alphabet letters in their natural order, excluding any previously removed letters.

Encryption/Decryption Rules (Pairs)

The letter pairs are encrypted (or decrypted) based on their position in the matrix:

1. **Different Row and Column (Rectangle):** Each letter is replaced by the letter on the same row, but in the column of the other letter in the pair.
2. **Same Row:** Each letter is replaced by the letter immediately to its **right** during encryption and to its **left** during decryption. The matrix wraps around (last column $\rightarrow$ first column, and vice-versa).
3. **Same Column:** Each letter is replaced by the letter immediately **below** it during encryption and **above** it during decryption. The matrix wraps around (bottom row $\rightarrow$ top row, and vice-versa).

4 Technical Implementation

The Playfair algorithm was implemented in **Python**, adapted for the 31-letter Romanian alphabet: AĂÂBCDEFGHIÎKLMNOPQRSȘTȚUVWXYZ. Since 31 does not perfectly fit a 5×5 matrix, a 6×6 matrix (36 cells) was chosen to simplify the modulo rules. The unused cells are filled with the placeholder character '*'.

4.1 Playfair Cipher Code Implementation

```
def create_playfair_matrix(key):
    # Romanian alphabet without J
    alphabet = "AĂÂBCDEFGHIÎKLMNOPQRSȘTȚUVWXYZ"

    key = key.upper().replace('J', 'I')
    seen = set()
    processed_key = []

    for char in key:
        if char in alphabet and char not in seen:
            seen.add(char)
            processed_key.append(char)

    matrix_chars = processed_key.copy()
    for char in alphabet:
        if char not in seen:
            matrix_chars.append(char)
```

```
    # Fill remaining spaces to make 36 characters (6x6 matrix)
    while len(matrix_chars) < 36:
        matrix_chars.append('*')

    # Create 6x6 matrix
    matrix = []
    for i in range(0, 36, 6):
        row = matrix_chars[i:i+6]
        matrix.append(row)

    return matrix


def find_position(matrix, char):
    for i, row in enumerate(matrix):
        for j, c in enumerate(row):
            if c == char:
                return i, j
    return None, None


def print_matrix(matrix):
    print("\nPlayfair Matrix:")
    for row in matrix:
        print(' '.join(row))
    print()


def prepare_text(text):
    alphabet = "AĂÂBCDEFGHIÎKLMNOPQRSȘTȚUVWXYZ"
    text = text.upper().replace('J', 'I')
    cleaned = ''.join([c for c in text if c in alphabet])
    return cleaned


def split_into_pairs(text):
    pairs = []
    i = 0

    while i < len(text):
        if i == len(text) - 1:
            pairs.append(text[i] + 'X')
```

```
            i += 1
        elif text[i] == text[i + 1]:
            pairs.append(text[i] + 'X')
            i += 1
        else:
            pairs.append(text[i] + text[i + 1])
            i += 2

    return pairs


def encrypt_pair(matrix, pair):
    char1, char2 = pair[0], pair[1]

    row1, col1 = find_position(matrix, char1)
    row2, col2 = find_position(matrix, char2)

    if row1 is None or row2 is None:
        return pair

    # Same row
    if row1 == row2:
        new_col1 = (col1 + 1) % 6
        new_col2 = (col2 + 1) % 6
        return matrix[row1][new_col1] + matrix[row2][new_col2]

    # Same column
    elif col1 == col2:
        new_row1 = (row1 + 1) % 6
        new_row2 = (row2 + 1) % 6
        return matrix[new_row1][col1] + matrix[new_row2][col2]

    # Rectangle
    else:
        return matrix[row1][col2] + matrix[row2][col1]


def decrypt_pair(matrix, pair):
    char1, char2 = pair[0], pair[1]

    row1, col1 = find_position(matrix, char1)
    row2, col2 = find_position(matrix, char2)
```

```
    if row1 is None or row2 is None:
        return pair

    # Same row
    if row1 == row2:
        new_col1 = (col1 - 1) % 6
        new_col2 = (col2 - 1) % 6
        return matrix[row1][new_col1] + matrix[row2][new_col2]

    # Same column
    elif col1 == col2:
        new_row1 = (row1 - 1) % 6
        new_row2 = (row2 - 1) % 6
        return matrix[new_row1][col1] + matrix[new_row2][col2]

    # Rectangle
    else:
        return matrix[row1][col2] + matrix[row2][col1]


def encrypt(plaintext, key):
    matrix = create_playfair_matrix(key)
    prepared = prepare_text(plaintext)
    pairs = split_into_pairs(prepared)

    print("Prepared text:", prepared)
    print("Pairs:", ' '.join(pairs))
    print_matrix(matrix)

    encrypted_pairs = [encrypt_pair(matrix, pair) for pair in pairs]
    ciphertext = ''.join(encrypted_pairs)

    return ciphertext


def decrypt(ciphertext, key):
    matrix = create_playfair_matrix(key)
    prepared = prepare_text(ciphertext)

    pairs = [prepared[i:i+2] for i in range(0, len(prepared), 2)]
```

```
    print("Prepared ciphertext:", prepared)
    print("Pairs:", ' '.join(pairs))
    print_matrix(matrix)

    decrypted_pairs = [decrypt_pair(matrix, pair) for pair in pairs]
    plaintext = ''.join(decrypted_pairs)

    return plaintext


def validate_input(text):
    allowed = "AĂÂBCDEFGHIÎKLMNOPQRSȘTȚUVWXYZaăâbcdefghiîklmnopqrsștțuvwxyz "

    for char in text:
        if char not in allowed:
            return False
    return True


def main():
    print("Playfair Cipher - Lab 3")
    print()

    print("Choose operation:")
    print("1. Encrypt")
    print("2. Decrypt")

    choice = input("\nEnter option (1/2): ").strip()

    if choice not in ['1', '2']:
        print("Invalid option!")
        return

    while True:
        key = input("\nEnter key (minimum 7 characters): ").strip()

        if len(key) < 7:
            print("Key must be at least 7 characters!")
            continue

        if not validate_input(key):
            print("Key contains invalid characters!")
```

```
            print("Allowed: A-Z, a-z, Ă, Â, Î, Ș, Ț")
            continue

        break

    if choice == '1':
        while True:
            plaintext = input("\nEnter message to encrypt: ").strip()

            if not validate_input(plaintext):
                print("Message contains invalid characters!")
                print("Allowed: A-Z, a-z, Ă, Â, Î, Ș, Ț")
                continue

            if not plaintext:
                print("Message cannot be empty!")
                continue

            break

        print("\n--- ENCRYPTION ---")
        ciphertext = encrypt(plaintext, key)

        print("Ciphertext:", ciphertext)
        spaced = ' '.join([ciphertext[i:i+2]
        for i in range(0, len(ciphertext), 2)])
        print("Ciphertext (spaced):", spaced)

    else:
        while True:
            ciphertext = input("\nEnter ciphertext to decrypt: ").strip()

            if not validate_input(ciphertext):
                print("Ciphertext contains invalid characters!")
                print("Allowed: A-Z, a-z, Ă, Â, Î, Ș, Ț")
                continue

            if not ciphertext:
                print("Ciphertext cannot be empty!")
                continue

            break
```

```
        print("\n--- DECRYPTION ---")
        plaintext = decrypt(ciphertext, key)

        print("Decrypted text:", plaintext)
        print("\nNote: Remove 'X' characters and add spaces manually.")


if __name__ == "__main__":
    main()


# The main function provides the user interface for selection and validation.
```

The implementation utilizes a 6×6 matrix to accommodate the 31 letters of the Romanian alphabet. The `create_playfair_matrix` function constructs the cipher square using the key, eliminating duplicates, and padding the unused cells with '*'. The `encrypt_pair` and `decrypt_pair` functions apply the core Playfair rules for rows, columns, and rectangles, using modulo 6 arithmetic for wrap-around effects. Input validation is handled in the main program (not shown here, but assumed based on the task), ensuring the key length is ≥ 7 and only valid Romanian/Latin characters are used.

Results for Task 3.1

```
Choose operation:
1. Encrypt
2. Decrypt

Enter option (1/2): 1

Enter key (minimum 7 characters): Barcelona

Enter message to encrypt: Ciprian

--- ENCRYPTION ---
Prepared text: CIPRIAN
Pairs: CI PR IA NX

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

Ciphertext: RÎSBHRÂV
Ciphertext (spaced): RÎ SB HR ÂV

Process finished with exit code 0
```

Figure 1: Encryption using the key `Barcelona`

```
Choose operation:
1. Encrypt
2. Decrypt

Enter option (1/2): 2

Enter key (minimum 7 characters): Barcelona

Enter ciphertext to decrypt: RÎSBHRÂV

--- DECRYPTION ---
Prepared ciphertext: RÎSBHRÂV
Pairs: RÎ SB HR ÂV

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

Decrypted text: CIPRIANX

Note: Remove 'X' characters and add spaces manually.

Process finished with exit code 0
```

Figure 2: Decryption of the resulting ciphertext

5 Conclusion

In this laboratory work, we successfully studied and implemented the **Playfair Algorithm** adapted for the **Romanian alphabet** (31 letters). The implementation required adjusting the encryption matrix to a 6×6 dimension and adapting the text preprocessing rules (e.g., validating specific Romanian characters).

All requirements of Task 3.1 were fulfilled:

- The Playfair algorithm was implemented, managing the creation of the 6×6 matrix based on a key.
- **Minimum key length** validation (minimum 7 characters) and **character validation** (including Ă, Â, Î, Ș, Ț) were implemented.
- The logic for **text preparation** (uppercase, space removal, pair splitting, and handling double/odd letters) was correctly implemented.
- The rules for **encryption** (shift right/down) and **decryption** (shift left/up) were correctly applied for the three cases (rectangle, same row, same column).

The exercise reinforced the understanding of polyalphabetic encryption principles and block ciphers, demonstrating that, although Playfair is a classic algorithm, it offers significantly greater security than monoalphabetic ciphers (like Caesar) through the use of a key-based matrix and encryption on digraphs.

Overall, the exercises strengthened understanding of encryption techniques and their practical implementation in programming, providing a foundation for studying more advanced cryptographic systems.