MINISTERUL EDUCAŢIEI ŞI CERCETĂRII AL REPUBLICII MOLDOVA

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Inginerie Software și Automatică

Report

Laboratory work №3

Cryptography and Security

Subject: Polyalphabetic Ciphers

Author:
Mihaela Untu,
std. gr. FAF-232

Verified:
Maia Zaica,
university assistant

Chișinău, 2025

Contents

1 Purpose of the laboratory work **3**

2 The Task **3**

3 Theoretical Considerations **3**

4 Technical implementation **4**

4.1 The `prepare_key()` Function 4

4.2 The `build_matrix()` Function 5

4.3 The `display_matrix()` Function 5

4.4 The `prepare_text_for_encryption()` Function 6

4.5 The `split_into_pairs()` Function 6

4.6 The `find_position()` Function 7

4.7 The `encrypt_pair()` Function 7

4.8 The `decrypt_pair()` Function 8

4.9 The `encrypt_message()` Function 8

4.10 The `decrypt_message()` Function 9

4.11 The `validate_input()` Function 9

4.12 The `main()` Function 10

5 Execution Examples **10**

5.1 Encryption Example 10

5.2 Decryption Example 11

6 Conclusion **12**

References **13**

1 Purpose of the laboratory work

This laboratory work aims to implement the Playfair encryption algorithm adapted for the Romanian alphabet, which contains 31 letters. The Playfair cipher represents a polyalphabetic encryption system invented by Charles Wheatstone in 1854, being one of the first ciphers that operates with digraphs (pairs of letters) instead of individual characters. This particular feature makes it significantly more resistant to frequency analysis compared to classical monoalphabetic ciphers.

2 The Task

Implement the Playfair algorithm in one of the programming languages for messages in Romanian (31 letters). The character values of the text must be between 'A' and 'Z', or 'a' and 'z', and no other values are allowed. If the user enters invalid characters, they will be prompted with the correct character range. The key length must not be shorter than 7. The user will be able to choose the operation — encryption or decryption — and will be able to enter the key, the message, or the ciphertext, and obtain the ciphertext or the decrypted message. The final step of adding new spaces, depending on the language used and the logic of the message, will be done manually.

3 Theoretical Considerations

Although the algorithm bears the name of Baron Lyon Playfair, it was actually invented by his friend, **Charles Wheatstone**, and was first described in a document on **March 26, 1854**. Initially, it was rejected by the *British Foreign Office* because it was considered too difficult to understand. When Wheatstone offered to demonstrate that within 15 minutes he could teach *three out of four nearby schoolboys* to use the algorithm, the Foreign Office secretary replied:

> "Yes, that may be possible, but you won't be able to teach them to be good diplomats."

After the algorithm was created, Baron Playfair persuaded the *British government* to adopt it for official use, which is why it carries his name rather than that of its true creator, Wheatstone.x The algorithm was used by the *British Army* during the *Boer War* in South Africa, and modified versions were later used by the *British during World War I* and by the *Australian Army during World War II*.

From the perspective of *modern cryptography*, the Playfair cipher is considered **obsolete and primitive**. Any modern personal computer can discover (crack) the key and decrypt the message within a few seconds—or even fractions of a second—using

the right software. Skilled *cryptanalysts* or even some *crossword experts* can break a Playfair-encrypted message within minutes using only a pencil and paper.

Although outdated in every practical sense, the **Playfair cipher** was one of the **first encryption algorithms to use the principles of modern block ciphers**. Studying this algorithm can provide a better *intuitive understanding of modern cryptography*, without requiring complex mathematical or number theory knowledge.

General Description of the Playfair Algorithm

The Playfair encryption involves the following steps:

1. Preparing the text to be encrypted;
2. Constructing the encryption matrix;
3. Constructing the encrypted message.

4 Technical implementation

[Link to GitHub Repository]

The implementation of the Playfair algorithm was realized following the principles of structured programming, organizing functionality into distinct and reusable modules. The program is structured in specialized functions, each having a well-defined responsibility, which facilitates understanding, testing, and code maintenance. This modular approach also allows easy extension of functionality in the future, if necessary. The main flow of the program is managed by the `main()` function, which implements an interactive menu for the user. This function coordinates calls to other functional modules, ensuring a coherent and intuitive user experience. The user can choose between encryption and decryption operations, can enter the desired key and text, and the program will validate input data and display results in a clear and structured format.

4.1 The `prepare_key()` Function

This function prepares the encryption key entered by the user. It converts all letters to uppercase, removes spaces, and eliminates duplicate characters, keeping only the first occurrence of each. It also replaces special letters according to the Playfair rules. The final result is a clean key ready to be used in building the matrix.

```
def prepare_key(key):
    key = key.upper()
    key = key.replace('I', 'I')
    key = key.replace(' ', '')
```

```
    prepared_key = ''
    for char in key:
        if char not in prepared_key:
            prepared_key += char
    return prepared_key
```

4.2 The `build_matrix()` Function

This function builds the Playfair matrix (5×6) based on the processed key and the Romanian alphabet. It appends the remaining unused letters of the alphabet to the key, then arranges all letters row by row in the matrix. The resulting matrix will be used in both encryption and decryption.

```
def build_matrix(key):
    alphabet = "ABCDEFGHIJKLMNOPQRSSTTUVWXYZAA"

    prepared_key = prepare_key(key)

    matrix_string = prepared_key
    for letter in alphabet:
        if letter not in matrix_string:
            matrix_string += letter

    matrix = []
    for i in range(5):
        row = []
        for j in range(6):
            index = i * 6 + j
            if index < len(matrix_string):
                row.append(matrix_string[index])
            else:
                row.append('')
        matrix.append(row)
    return matrix
```

4.3 The `display_matrix()` Function

This function displays the Playfair matrix in a clear, table-like format on the screen. It helps the user visualize how the letters are arranged and verify that the matrix was built correctly based on the key.

```
def display_matrix(matrix):
    print("\nEncryption Matrix:")
    print("+" + "-" * 47 + "+")
    for row in matrix:
        print("|", end="")
```

```
        for cell in row:
            if cell:
                print(f" {cell:^5} |", end="")
            else:
                print("       |", end="")
        print()
        print("+" + "-" * 47 + "+")
    print()
```

4.4 The `prepare_text_for_encryption()` Function

This function cleans and normalizes the input text before encryption. It converts all letters to uppercase, removes spaces, replaces special characters, and keeps only valid Romanian letters. The result is a string of valid characters ready to be split into pairs.

```
def prepare_text_encrypt(text):
    text = text.upper().replace(' ', '')
    text = text.replace('I', 'I')

    valid_chars = "ABCDEFGHIJKLMNOPQRSSRTUVWXYZAA"
    clean_text = ""
    for char in text:
        if char in valid_chars:
            clean_text += char
    return clean_text
```

4.5 The `split_into_pairs()` Function

This function divides the cleaned text into digraphs (pairs of letters). If two identical letters appear in a pair, an "X" is inserted between them. If the text has an odd number of letters, an "X" is added at the end. These pairs are then ready for encryption.

```
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
```

4.6 The `find_position()` Function

This helper function finds the position of a given letter in the Playfair matrix. It returns the row and column where the letter is located, which are then used to apply the encryption or decryption rules.

```
def find_position(matrix, letter):
    letter = letter.upper()
    if letter == 'I':
        letter = 'I'

    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if matrix[i][j] == letter:
                return i, j
    return None, None
```

4.7 The `encrypt_pair()` Function

This function encrypts a single pair of letters according to the Playfair rules. If the letters are in the same row, each is replaced by the one to its right. If they are in the same column, each is replaced by the one below. Otherwise, they are replaced by the letters at the corners of the rectangle they form.

```
def encrypt_pair(matrix, pair):
    letter1, letter2 = pair[0], pair[1]
    row1, col1 = find_position(matrix, letter1)
    row2, col2 = find_position(matrix, letter2)

    if row1 is None or row2 is None:
        return pair

    #Rule 1: Rectangle - different rows and columns
    if row1 != row2 and col1 != col2:
        return matrix[row1][col2] + matrix[row2][col1]
    #Rule 2: Same Row
    elif row1 == row2:
        new_col1 = (col1 + 1) % 6
        new_col2 = (col2 + 1) % 6

        while not matrix[row1][new_col1]:
            new_col1 = (new_col1 + 1) % 6
        while not matrix[row2][new_col2]:
            new_col2 = (new_col2 + 1) % 6

        return matrix[row1][new_col1] + matrix[row2][new_col2]
    #Rule 3: Same Column
```

```
    else:
        new_row1 = (row1 + 1) % 5
        new_row2 = (row2 + 1) % 5
        return matrix[new_row1][col1] + matrix[new_row2][col2]
```

4.8 The `decrypt_pair()` Function

This function performs the reverse operation of `encrypt_pair()`. If the letters are on the same row, each is replaced by the one to its left. If they are in the same column, each is replaced by the one above. For rectangles, the same column-swap rule applies, restoring the original pair.

```
def decrypt_pair(matrix, pair):
    letter1, letter2 = pair[0], pair[1]
    row1, col1 = find_position(matrix, letter1)
    row2, col2 = find_position(matrix, letter2)

    if row1 is None or row2 is None:
        return pair

    #Rule 1: Rectangle - different rows and columns
    if row1 != row2 and col1 != col2:
        return matrix[row1][col2] + matrix[row2][col1]
    #Rule 2: Same Row
    elif row1 == row2:
        new_col1 = (col1 - 1) % 6
        new_col2 = (col2 - 1) % 6

        while not matrix[row1][new_col1]:
            new_col1 = (new_col1 - 1) % 6
        while not matrix[row2][new_col2]:
            new_col2 = (new_col2 - 1) % 6

        return matrix[row1][new_col1] + matrix[row2][new_col2]
    #Rule 3: Same Column
    else:
        new_row1 = (row1 - 1) % 5
        new_row2 = (row2 - 1) % 5
        return matrix[new_row1][col1] + matrix[new_row2][col2]
```

4.9 The `encrypt_message()` Function

This function manages the full encryption process. It prepares the text, splits it into pairs, builds the matrix, and encrypts each pair. Finally, it combines all encrypted pairs into a single ciphertext string.

```
def encrypt_message(message, key):
    #a: prepare text
    prepared_text = prepare_text_encrypt(message)
    pairs = split_into_pairs(prepared_text)
    #b: build matrix
    matrix = build_matrix(key)
    #c: encrypt pairs
    ciphertext = ""
    for pair in pairs:
        ciphertext += encrypt_pair(matrix, pair)
    return ciphertext, matrix, pairs
```

4.10 The `decrypt_message()` Function

This function decrypts the ciphertext using the same matrix and key as encryption. It splits the text into pairs and applies the reverse Playfair rules to restore the original message, which may still contain the inserted "X" characters.

```
def decrypt_message(ciphertext, key):
    #a: prepare text
    prepared_text = prepare_text_encrypt(ciphertext)
    pairs = split_into_pairs(prepared_text)
    #b: build matrix
    matrix = build_matrix(key)
    #c: decrypt pairs
    plaintext = ""
    for pair in pairs:
        plaintext += decrypt_pair(matrix, pair)
    return plaintext, matrix, pairs
```

4.11 The `validate_input()` Function

This function checks whether the user input contains only valid Romanian letters and spaces. If it finds invalid characters, it warns the user and requests correction before continuing the encryption or decryption process.

```
def validate_input(text):
    valid_chars = "
        ABCDEFGHIJKLMNOPQRSTUVWXYZ      abcdefghijklmnopqrstuvwxyz
         "

    invalid_chars = set()
    for char in text:
        if char not in valid_chars:
            invalid_chars.add(char)
```

```
    if invalid_chars:
        print(f"\nWARNING: Invalid characters detected: {', '.join(
            invalid_chars)}")
        print("Valid character range: A-Z, a-z (including   ,   ,   ,
              ,   ) and spaces")
        return False
    return True
```

4.12 The `main()` Function

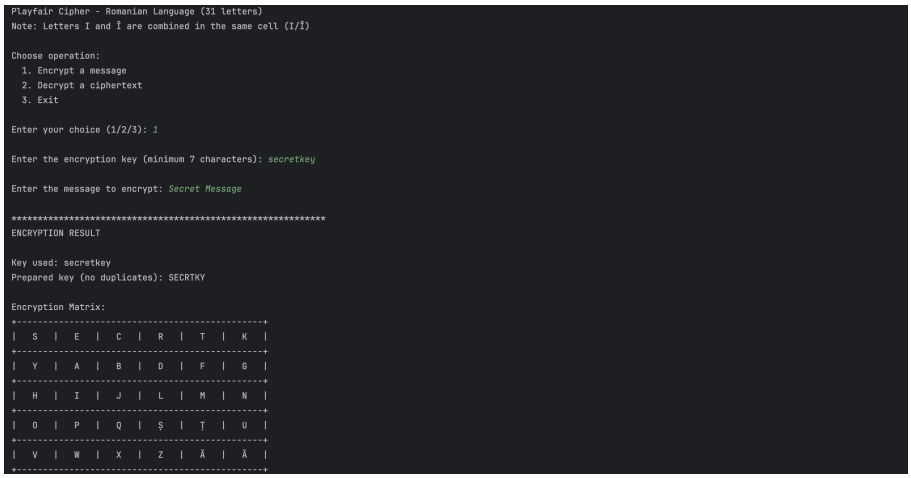
The main control function of the program provides an interactive menu where the user can choose to encrypt, decrypt, or exit. It manages input validation, calls the necessary functions, and displays results such as the matrix, digraphs, and the final message.

5 Execution Examples

5.1 Encryption Example

To demonstrate the program's operation, let us consider the following encryption example. The user chooses the encryption option from the menu and enters the key "SECRETKEY", which meets the minimum requirement of seven characters. The program processes the key by eliminating duplicates and builds the corresponding Playfair matrix. Let us assume the user wants to encrypt the message "SECRET MESSAGE".

After text preparation, which becomes "SECRETMESSAGE", it is split into the digraphs: SE, CR, ET, ME, SX, SA, GE (where X is added to separate duplicate S letters). Each digraph is then encrypted according to Playfair rules using the built matrix. The program displays the matrix, the list of digraphs, and the resulting ciphertext, along with indication of the rule applied for each pair. This step-by-step presentation facilitates understanding of the process and allows verification of implementation correctness.



```
Original message: Secret Message
Prepared text (uppercase, no spaces): SECRETMESSAGE
Split into pairs: SE CR ET ME SX SA GE

Pair     -> Encrypted  (Rule applied)
--------------------------------------------------
SE       -> EC         (Same row)
CR       -> RT         (Same row)
ET       -> CK         (Same row)
ME       -> IT         (Rectangle)
SX       -> CV         (Rectangle)
SA       -> EY         (Rectangle)
GE       -> AK         (Rectangle)

FINAL CIPHERTEXT: ECRTCKITCVEYAK
Formatted in pairs: EC RT CK IT CV EY AK
```

5.2 Decryption Example

For the decryption operation, the user enters the same key "SECRETKEY" and the ciphertext obtained previously. The program builds the Playfair matrix again based on the key (which will be identical to the one used for encryption) and splits the ciphertext into digraphs. Each digraph is then decrypted using the inverse rules of the Playfair algorithm. The resulting decrypted text is "SECRETMESXSAGE", which corresponds to the original message with the addition of the X character for separation. The user can easily recognize the original message and can manually remove the artificially inserted X characters. The program displays all intermediate stages of the decryption process, including the digraphs and applied rules, offering a complete understanding of the algorithm's operating mechanism.

```
Choose operation:
  1. Encrypt a message
  2. Decrypt a ciphertext
  3. Exit

Enter your choice (1/2/3):
Enter the encryption key (minimum 7 characters): secretkey

Enter the ciphertext to decrypt: ECRTCKITCVEYAK

************************************************************
DECRYPTION RESULT

Key used: secretkey
Prepared key (no duplicates): SECRTKY

Encryption Matrix:
+-----------------------------------------------+
|   S   |   E   |   C   |   R   |   T   |   K   |
+-----------------------------------------------+
|   Y   |   A   |   B   |   D   |   F   |   G   |
+-----------------------------------------------+
|   H   |   I   |   J   |   L   |   M   |   N   |
+-----------------------------------------------+
|   O   |   P   |   Q   |   Ş   |   Ţ   |   U   |
+-----------------------------------------------+
|   V   |   W   |   X   |   Z   |   Ă   |   Â   |
+-----------------------------------------------+

Ciphertext: ECRTCKITCVEYAK
Split into pairs: EC RT CK IT CV EY AK
```

```
Pair     -> Decrypted  (Rule applied)
--------------------------------------------------
EC       -> SE         (Same row)
RT       -> CR         (Same row)
CK       -> ET         (Same row)
IT       -> ME         (Rectangle)
CV       -> SX         (Rectangle)
EY       -> SA         (Rectangle)
AK       -> GE         (Rectangle)

FINAL DECRYPTED MESSAGE: SECRETMESXSAGE

------------------------------------------------------------
NOTE: You need to manually add spaces according to
      the language logic and message context.
      The letter X may indicate:
      - Separation of duplicate letters
      - Padding for odd-length text
------------------------------------------------------------
```

6 Conclusion

The implementation of the Playfair algorithm for Romanian has been successfully realized, respecting all requirements specified in the laboratory assignment. The program offers an easy-to-use interactive interface, correctly validates input data, and produces correct results for encryption and decryption operations. The adaptation of the algorithm for the Romanian alphabet, by combining the letters I and Î and keeping the letter J as a separate entity, demonstrates the flexibility of the Playfair cipher and its capacity to be applied to different writing systems.

From an educational point of view, the implementation of this algorithm offers a practical understanding of classical cryptography principles and how polyalphabetic ciphers improve security compared to monoalphabetic ones. Although the Playfair algorithm is considered outdated from the perspective of modern cryptography and can be broken relatively easily with current techniques, its study remains valuable for understanding the evolution of encryption methods and fundamental concepts that underlie contemporary cryptographic systems.

Through the realization of this laboratory work, I have gained practical experience in implementing cryptographic algorithms, working with matrix data structures, and developing interactive interfaces for users. The source code is modularly structured, well documented, and can be easily extended to include additional functionalities, such as automatic cryptanalysis or graphical interface.

References

[1] Github Repository, *https://github.com*