Ministry of Education 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: Berco Andrei, FAF - 221

Checked: asist. univ., Nirca Dumitru

Chișinău 2024

Topic: Polyalphabetic Cipher

Tasks:

1. Implement the Playfair algorithm in one of the programming languages for messages in Romanian (31 letters). The character values of the text are between 'A' and 'Z', 'a' and 'z', and no other values are allowed. If the user enters other values, they will be suggested the correct range of characters. The key length must be at least 7. The user will be able to choose the operation - encryption or decryption, will be able to enter the key, message, or ciphertext, and will obtain the ciphertext or the decrypted message. The final phase of adding new spaces, depending on the language used and the logic of the message, will be done manually.

Theoretical notes:

The Playfair algorithm for encryption involves three main steps:

1. **Text Preparation**: The plaintext is formatted by converting all letters to uppercase, removing spaces and punctuation, replacing 'J' with 'I', and dividing the text into pairs of letters. If there are duplicate letters in a pair, a filler letter (e.g., 'Q', 'X', or 'Z') is inserted.
2. **Key Matrix Construction**: Repeated letters in the encryption key are removed, and a 5x5 matrix is constructed using the processed key followed by the remaining letters of the alphabet (excluding 'J'). This matrix can vary in size based on the alphabet used.
3. **Encrypted Message Construction**: Pairs of letters from the prepared text are encrypted based on their positions in the matrix:
    - If the letters are in different rows and columns, each is replaced by the letter in the same row but in the column of the other letter.
    - If the letters are in the same row, each is replaced by the letter immediately to its right, wrapping around if necessary.
    - If the letters are in the same column, each is replaced by the letter immediately below, again wrapping around if needed.

Decryption Process

To decrypt a message, the same matrix is used, and the encryption steps are reversed:

1. For letters in different rows and columns, the same substitution is applied as in encryption.
2. For letters in the same row, each is replaced by the letter immediately to its left.
3. For letters in the same column, each is replaced by the letter immediately above.

Final formatting of the decrypted text involves removing spaces between letter pairs and reintroducing spaces based on the logical structure of the original message.

Implementation:

```
def remove_punctuation(self, text):
        return text.translate(str.maketrans('', '',
string.punctuation)).replace(" ", "").upper().replace("J", "I")


    def insert_additional_char(self, text):
        for i in range(0, len(text) - 1):
            if text[i] == text[i+1]:
                text = text[:i+1] + "X" + text[i+1:]
        if len(text) % 2 != 0:
            text += "X"
        return text


    def split_text_in_pairs(self, text):
        splited_text = ""
        i = 0
        for char in text:
            splited_text += char
            i+=1
            if i % 2 == 0:
                splited_text += " "
        return splited_text
```

- **remove_punctuation(self, text)**:
    - This method removes all punctuation from the input text and converts it to uppercase. It also replaces the letter 'J' with 'I' and removes spaces, effectively standardizing the input.
- **insert_additional_char(self, text)**:

- This method checks for consecutive duplicate characters in the text. If it finds any, it inserts the letter 'X' between them to prevent them from being encrypted together. If the final text length is odd, it appends an 'X' to make it even.

- **split_text_in_pairs(self, text)**:

  - This method splits the prepared text into pairs of characters. It adds a space after every two characters to facilitate processing during encryption and decryption.

```
def prepare_key(self):
        used_letters = []
        return "".join([char for char in self.key if char not in used_letters
and not used_letters.append(char)]).replace(" ", "").upper()

    def construct_matrix(self):
        key = self.prepare_key()
        alphabet = "AĂÂBCDEFGHIÎKLMNOPQRSȘTȚUVWXYZ"
        matrix = np.zeros((6,5), dtype=str)

        for i in range(6):
            for j in range(5):
                if i*5+j < len(key):
                    matrix[i][j] = key[i*5+j]
                else:
                    for char in alphabet:
                        if char not in key:
                            matrix[i][j] = char
                            alphabet = alphabet.replace(char, "")
                            break
        return matrix
```

- **prepare_key(self)**:

  - This method processes the key by removing duplicate letters and spaces, converting it to uppercase. It ensures that the key is unique for matrix construction.

- **construct_matrix(self)**:

  - Constructs a 6x5 matrix using the processed key and the defined alphabet. It fills the matrix first with the letters from the key, then with the remaining letters of the alphabet that are not in the key.

```
def get_position(self, char):
        for i in range(6):
            for j in range(5):
                if self.crypto_matrix[i][j] == char:
                    return i, j
        return None, None

    def encrypt(self):
        encrypted_text = ""
        for i in range(0, len(self.text), 3):
            first_letter = self.text[i]
            second_letter = self.text[i+1]

            first_letter_position = self.get_position(first_letter)
            second_letter_position = self.get_position(second_letter)
            # print("First letter: ", first_letter, " position: ",
first_letter_position)
            # print("Second letter: ", second_letter, " position: ",
second_letter_position)

            #Same line
            if first_letter_position[0] == second_letter_position[0]:
                encrypted_text +=
self.crypto_matrix[first_letter_position[0]][(first_letter_position[1]+1)%5]
                encrypted_text +=
self.crypto_matrix[second_letter_position[0]][(second_letter_position[1]+1)%5]

            #Same column
            elif first_letter_position[1] == second_letter_position[1]:
                encrypted_text +=
self.crypto_matrix[(first_letter_position[0]+1)%6][first_letter_position[1]]
                encrypted_text +=
self.crypto_matrix[(second_letter_position[0]+1)%6][second_letter_position[1]]

            #Rectangle
            else:
                encrypted_text +=
self.crypto_matrix[first_letter_position[0]][second_letter_position[1]]
                encrypted_text +=
self.crypto_matrix[second_letter_position[0]][first_letter_position[1]]

        return encrypted_text
```

- **get_position(self, char)**:

    - Finds the position (row and column) of a character in the matrix. This is crucial for the encryption and decryption processes.

- **encrypt(self)**:

    - This method encrypts the prepared text. It processes the text in pairs, checking if they are in the same row, same column, or form a rectangle in the matrix. It then replaces each character according to the encryption rules and constructs the encrypted text.

- **decrypt(self)**:

    - This method decrypts the text similarly to the `encrypt` method, but it adjusts the character positions based on the decryption rules (e.g., moving left or up in the matrix).

main.py

This script serves as the user interface and the entry point for running the encryption or decryption process.

- **User Interface**:
    - It displays a menu for the user to choose between encrypting, decrypting, or exiting the program.
- **Encrypt Option**:
    - It takes the plaintext and key as input, prepares the text using the `TextPreparation` class, and constructs the matrix using the `Matrix` class. If the key is valid (at least 7 characters), it encrypts the text and displays the result.
- **Decrypt Option**:
    - Similar to the encrypt option, it accepts the ciphertext and key, constructs the matrix, and decrypts the text. It also checks the key's validity.
- **Exit Option**:
    - Allows the user to exit the program.

Results:

```
1. Encrypt
2. Decrypt
3. Exit
Choose an option: 1
Messag to encrypt: Hello World
Key: password

Prepared text:  HE LX LO WO RL DX

Matrix:
 [['P' 'A' 'S' 'W' 'O']
 ['R' 'D' 'Ă' 'Â' 'B']
 ['C' 'E' 'F' 'G' 'H']
 ['I' 'Î' 'K' 'L' 'M']
 ['N' 'Q' 'Ș' 'T' 'Ț']
 ['U' 'V' 'X' 'Y' 'Z']]

Ecripted text:
 CFKYMWOPÂIĂV

1. Encrypt
2. Decrypt
3. Exit
Choose an option: 2
Messag to decrypt: CFKYMWOPÂIĂV
Key: password
Decrypted text:
 HELXLOWORLDX
```

Conclusion

In this project, we developed a robust encryption and decryption system utilizing a Playfair cipher approach, structured around three core modules: `TextPreparation`, `Matrix`, and `CryptoService`. The implementation emphasizes clear modular design, enhancing maintainability and readability while effectively handling text preprocessing, matrix construction, and cryptographic operations.

The `TextPreparation` module ensures that input text is sanitized and formatted for encryption. By removing punctuation and handling duplicate characters, it prepares the text in a way that adheres to the constraints of the Playfair cipher. The `Matrix` module constructs a custom 6x5 matrix based on a user-defined key, ensuring a unique and dynamic encryption method. The use of a distinct alphabet enhances security and complexity, making it more resistant to cryptanalysis.

The `CryptoService` module performs the core cryptographic functions, employing both encryption and decryption algorithms that utilize matrix operations. By efficiently determining character positions and applying the Playfair cipher rules, this module effectively transforms the prepared text into ciphertext and back, demonstrating the effectiveness of the chosen encryption technique.

Through the implementation of a user-friendly interface in `main.py`, the system facilitates easy interaction, allowing users to encrypt and decrypt messages seamlessly. This project not only illustrates the principles of classical cryptography but also serves as a foundation for further exploration into more advanced encryption techniques.

Overall, this encryption system showcases the importance of structured programming, the interplay between different modules, and the application of cryptographic principles in software development. Future work may include enhancements such as additional encryption algorithms, user authentication, and improved user interface features, further expanding the system's capabilities and security.