**CNS Lab**

**Assignment 4**

**Title: Study Playfair Cipher And Vigenere Cipher**

**Information:**

**Encryption:**

The plaintext is divided into pairs of letters (digraphs).

For each digraph, you apply the following rules:

If the two letters of the digraph are in the same row in the Playfair Square, replace them with the letters to their right (wrapping around if they are at the end of the row).

If the two letters are in the same column, replace them with the letters below (wrapping around if they are at the bottom of the column).

If the two letters form a rectangle (neither in the same row nor the same column), replace them with the letters at the opposite corners of the rectangle.

Repeat this process for all digraphs to form the ciphertext.

**Decryption:**

The decryption process is essentially the reverse of encryption. The same Playfair Square is used.

For each digraph in the ciphertext, apply the reverse of the encryption rules to obtain the plaintext digraph.

**Code:**

def prepare\_key(key):

    prepared\_key = ""

    used = [False] \* 26  # to keep track of used letters

    # Remove spaces from the key and convert to uppercase

    for c in key:

        if c.isalpha():

            c = c.upper()

            if not used[ord(c) - ord('A')]:

                prepared\_key += c

                used[ord(c) - ord('A')] = True

    # Fill the remaining spaces with the alphabet (except 'J' which is combined with 'I')

    for c in range(ord('A'), ord('Z') + 1):

        if chr(c) != 'J' and not used[c - ord('A')]:

            prepared\_key += chr(c)

            used[c - ord('A')] = True

    return prepared\_key

def create\_matrix(key):

    prepared\_key = prepare\_key(key)

    matrix = [['' for \_ in range(5)] for \_ in range(5)]

    k = 0

    for i in range(5):

        for j in range(5):

            matrix[i][j] = prepared\_key[k]

            k += 1

    return matrix

def find\_position(matrix, char):

    if char == 'J':

        char = 'I'  # treat 'J' as 'I'

    for i in range(5):

        for j in range(5):

            if matrix[i][j] == char:

                return i, j

def encrypt\_playfair(plaintext, key):

    matrix = create\_matrix(key)

    ciphertext = ""

    length = len(plaintext)

    for i in range(0, length, 2):

        a = plaintext[i]

        b = plaintext[i + 1] if i + 1 < length else 'X'  # If the text length is odd, add 'X' at the end

        row\_a, col\_a = find\_position(matrix, a)

        row\_b, col\_b = find\_position(matrix, b)

        # row

        if row\_a == row\_b:

            ciphertext += matrix[row\_a][(col\_a + 1) % 5]

            ciphertext += matrix[row\_b][(col\_b + 1) % 5]

        # column

        elif col\_a == col\_b:

            ciphertext += matrix[(row\_a + 1) % 5][col\_a]

            ciphertext += matrix[(row\_b + 1) % 5][col\_b]

        # rectangle

        else:

            ciphertext += matrix[row\_a][col\_b]

            ciphertext += matrix[row\_b][col\_a]

    return ciphertext

def decrypt\_playfair(ciphertext, key):

    matrix = create\_matrix(key)

    plaintext = ""

    length = len(ciphertext)

    for i in range(0, length, 2):

        a = ciphertext[i]

        b = ciphertext[i + 1] if i + 1 < length else 'X'  # If the text length is odd, add 'X' at the end

        row\_a, col\_a = find\_position(matrix, a)

        row\_b, col\_b = find\_position(matrix, b)

       #same row

        if row\_a == row\_b:

            plaintext += matrix[row\_a][(col\_a - 1) % 5]  # Decrement by 1

            plaintext += matrix[row\_b][(col\_b - 1) % 5]  # Decrement by 1

        # column

        elif col\_a == col\_b:

            plaintext += matrix[(row\_a - 1) % 5][col\_a]  # Decrement by 1

            plaintext += matrix[(row\_b - 1) % 5][col\_b]  # Decrement by 1

        # rectangle

        else:

            plaintext += matrix[row\_a][col\_b]

            plaintext += matrix[row\_b][col\_a]

    return plaintext

key = input("Enter the key: ")

plaintext = input("Enter the plaintext: ")

# Remove spaces and convert to uppercase

plaintext = ''.join(plaintext.split())

plaintext = plaintext.upper()

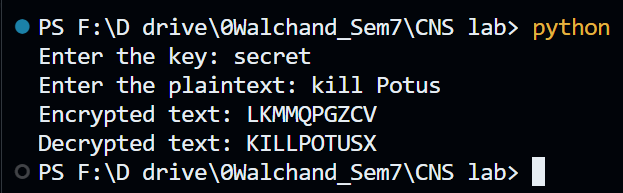
ciphertext = encrypt\_playfair(plaintext, key)

print("Encrypted text:", ciphertext)

decrypted\_text = decrypt\_playfair(ciphertext, key)

print("Decrypted text:", decrypted\_text)

**Output:**

****

**Limitations:**

1. Fixed Key Length: The Playfair cipher requires a fixed 5x5 key matrix, which may not be suitable for all applications, and changing the key can be cumbersome.

2. Known-Plaintext Vulnerability: If an attacker has access to both the ciphertext and a portion of the corresponding plaintext, they can potentially determine parts of the key and decrypt the entire message.

3. Limited Character Set: Playfair was designed for English text and may not work well with other languages or character sets with different alphabets or symbols.

4. No Authentication or Integrity Protection: Playfair provides confidentiality but lacks authentication and data integrity protection, making it susceptible to tampering.

5. Security Weakness: While stronger than simple substitution ciphers, Playfair is still vulnerable to certain cryptanalysis techniques, making it less secure compared to modern encryption methods.

**Code for Vigenere Cipher:**

**Inforation:**

**Key Setup:**

Select a keyword or key phrase.

Duplicate the keyword to match the length of the plaintext, repeating it as necessary.

Convert the letters of the keyword into their corresponding numerical values (A=0, B=1, ..., Z=25).

**Encryption:**

Divide the plaintext into individual letters and convert them into numerical values.

For each letter, compute the shift value using the corresponding keyword letter.

Apply the calculated shift value to the plaintext letter (using modulo 26).

Convert the resulting numerical value back into a letter to generate the ciphertext.

**Decryption:**

Split the ciphertext into individual letters and convert them to numerical values.

For each letter, determine the shift value using the corresponding keyword letter.

Reverse the shift operation (subtract the shift value, modulo 26).

Convert the adjusted numerical value back into a letter to reveal the original plaintext.

**Advantages:**

Resistance to Frequency Analysis: Unlike monoalphabetic ciphers, it is less susceptible to frequency analysis.

Key Length Impact: Security improves as the keyword length increases, expanding the key space.

**Disadvantages:**

Vulnerability to Kasiski Examination: For shorter keywords, the Vigenère Cipher can be vulnerable to attacks like the Kasiski examination, which may reveal key length repetitions.

Short or Predictable Keywords: Security may weaken if the keyword is too short or easily predictable, making it easier for adversaries to decipher the message.

**Code:**

def main():

    choice = int(input("Choose an option:\n1. Encryption\n2. Decryption\nEnter your choice (1 or 2): "))

    input()

    if choice == 1:

        # Encryption

        plain\_text = input("\nEnter plain text: ")

        key = input("\nEnter key: ")

        plain\_text = plain\_text.replace(" ", "").lower()

        key = key.replace(" ", "").lower()

        # Encryption

        cipher\_text = ""

        for i in range(len(plain\_text)):

            val = (ord(plain\_text[i]) - ord('a') + ord(key[i % len(key)]) - ord('a')) % 26

            cipher\_text += chr(ord('a') + val)

        print("\nCipher Text:", cipher\_text)

    elif choice == 2:

        # Decryption

        cipher\_text = input("\nEnter cipher text: ")

        key = input("\nEnter key: ")

        key = key.replace(" ", "").lower()

        # Decryption

        decrypted = ""

        for i in range(len(cipher\_text)):

            val = (ord(cipher\_text[i]) - ord('a') - (ord(key[i % len(key)]) - ord('a')) + 26) % 26

            decrypted += chr(ord('a') + val)

        print("\nAfter decryption:", decrypted)

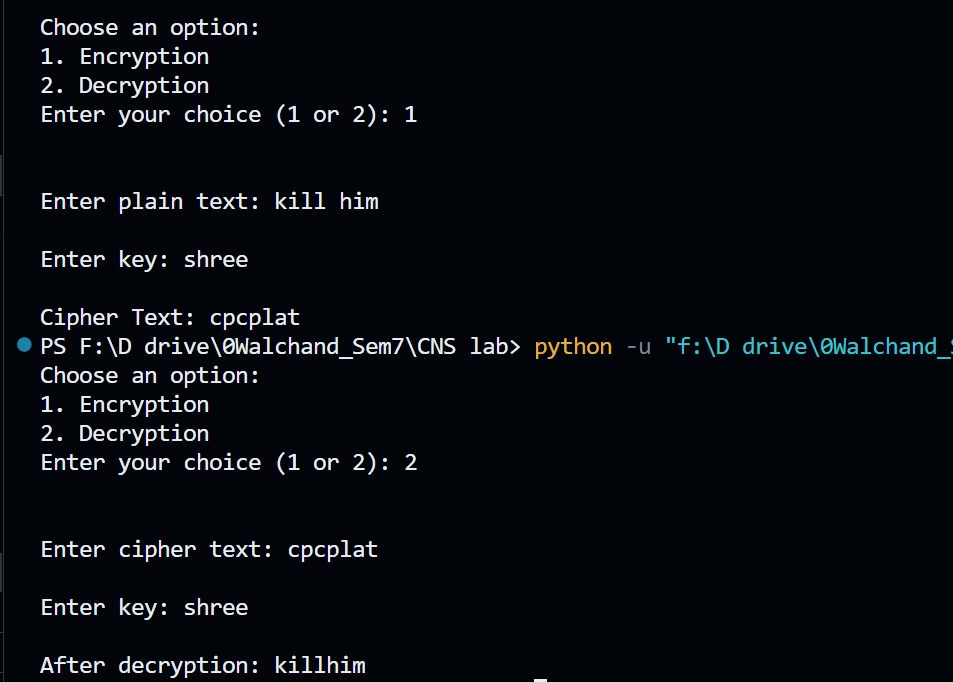
    else:

        print("Invalid choice. Please choose 1 or 2.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**

****

**Limitations:**

1. Fixed Key Length: The key in the Vigenère cipher must be repeated to match the length of the plaintext, which can be cumbersome for long messages. Longer keys are more secure, but they can be difficult to manage.

2. Weak Security: While stronger than simple Caesar ciphers, the Vigenère cipher is still relatively weak and vulnerable to various cryptographic attacks. It does not provide strong security against modern cryptanalysis techniques.

3. Known-Plaintext Attack: If an attacker has access to both the ciphertext and a portion of the corresponding plaintext (known-plaintext attack), they can potentially determine parts of the key and decrypt the entire message.

4. Repeating Key Patterns: The repetition of the key creates patterns in the ciphertext, making it susceptible to frequency analysis and revealing language characteristics.

5. No Authentication or Integrity Protection: The Vigenère cipher only provides confidentiality; it does not offer any form of authentication or data integrity protection. An attacker can tamper with the ciphertext without detection.

6. Limited Character Set: The Vigenère cipher was originally designed for English text and may not work well with other languages or character sets with different alphabets, symbols, or numbers.

7. Key Management: Managing and distributing keys can be challenging, especially for secure communication involving multiple parties or channels. Keys should be kept secret, and their distribution can be a security concern.

8. Not Suitable for Modern Cryptographic Needs: Due to its security vulnerabilities and limitations, the Vigenère cipher is not suitable for securing sensitive data in contemporary applications. Modern encryption methods offer much stronger security.

9. Complexity for Users: While encryption and decryption are relatively straightforward, key management and selection can be challenging for users. The cipher's security relies heavily on the key, and choosing a strong, random key can be complex.