**Final Year B.Tech. (CSE) – VII [ 2024-25]**

**6CS451: Cryptography and Network Security Lab (C&NS Lab)**

**Date: 01/08/2024**

**Assignment 1**

**PRN:** 21510042  **Name:** Omkar Rajesh Auti

**1. Perform encryption, decryption using the following substitution techniques:**

**a. Ceaser cipher**

**Ans:**

The Caesar Cipher is a simple encryption technique where each letter in a message is shifted by a fixed number of positions in the alphabet. For example, with a shift of 3, "A" becomes "D," "B" becomes "E," and so on. It's one of the oldest known ciphers and is easy to implement but also easy to break.

**Python Code:**

def caesar\_encrypt(text, shift):

    """

    Encrypt the plain text using Caesar cipher.

    Parameters:

    text (str): The input text to be encrypted.

    shift (int): The number of positions to shift each character.

    Returns:

    str: The encrypted text.

    """

    encrypted\_text = ""

    for char in text:

        if char.isalpha():

            shift\_amount = shift % 26

            if char.islower():

                new\_char = chr((ord(char) - ord('a') + shift\_amount) % 26 + ord('a'))

            else:

                new\_char = chr((ord(char) - ord('A') + shift\_amount) % 26 + ord('A'))

            encrypted\_text += new\_char

        else:

            encrypted\_text += char

    return encrypted\_text

def caesar\_decrypt(text, shift):

    """

    Decrypt the encrypted text using Caesar cipher.

    Parameters:

    text (str): The input text to be decrypted.

    shift (int): The number of positions to shift each character back.

    Returns:

    str: The decrypted text.

    """

    return caesar\_encrypt(text, -shift)

def main():

    """

    The main function to run the menu-driven program.

    """

    while True:

        print("\nCaesar Cipher Program")

        print("1. Encrypt")

        print("2. Decrypt")

        print("3. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':

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

            shift = int(input("Enter the shift value: "))

            encrypted\_text = caesar\_encrypt(plain\_text, shift)

            print(f"\nEncrypted Text: {encrypted\_text}")

        elif choice == '2':

            encrypted\_text = input("Enter the encrypted text: ")

            shift = int(input("Enter the shift value: "))

            decrypted\_text = caesar\_decrypt(encrypted\_text, shift)

            print(f"Decrypted Text: {decrypted\_text}")

        elif choice == '3':

            print("Exiting the program.")

            break

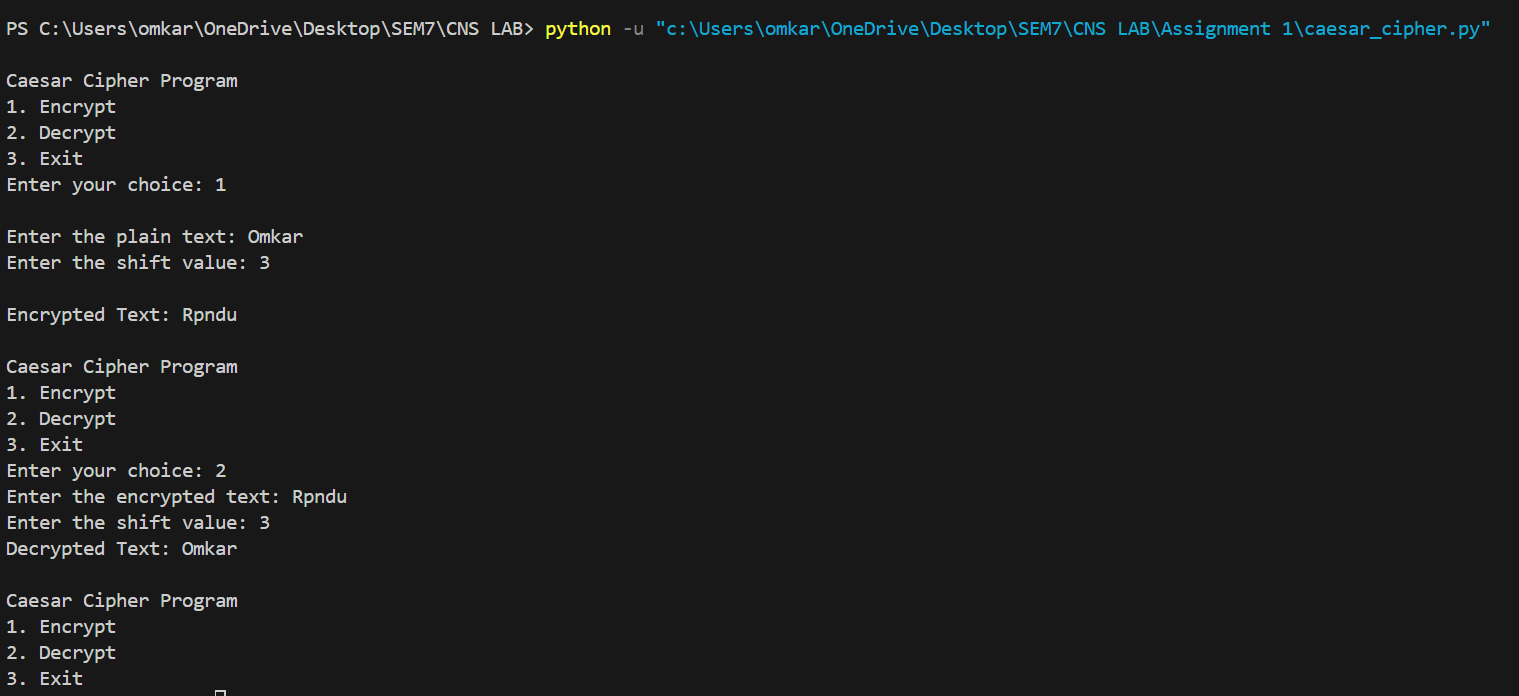
        else:

            print("Invalid choice. Please try again.")

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

    main()

**Output:**

****

**Advantages:**

* **Simplicity**: Easy to understand and implement.
* **Efficiency**: Fast encryption and decryption.

**Disadvantages:**

* **Weak Security**: Vulnerable to frequency analysis and brute-force attacks (only 25 possible shifts).
* **Predictability**: Does not change much between different texts.

**b. Playfair cipher**

**Ans:**

The Playfair Cipher is a digraph substitution cipher that encrypts pairs of letters. It uses a 5x5 matrix of letters created from a keyword. To encrypt, locate each letter pair in the matrix and swap or substitute based on their positions. It’s more secure than simple substitution ciphers because it encodes pairs of letters rather than individual letters.

**Python code:**

def generate\_playfair\_matrix(key):

    """

    Generate a 5x5 matrix for the Playfair cipher based on the provided key.

    Parameters:

    key (str): The key to generate the matrix.

    Returns:

    list: A 5x5 matrix for the Playfair cipher.

    """

    key = key.upper().replace("J", "I")

    matrix = []

    used = set()

    for char in key:

        if char not in used and char.isalpha():

            used.add(char)

            matrix.append(char)

    for char in "ABCDEFGHIKLMNOPQRSTUVWXYZ":

        if char not in used:

            used.add(char)

            matrix.append(char)

    return [matrix[i:i + 5] for i in range(0, 25, 5)]

def find\_position(matrix, char):

    """

    Find the row and column of a character in the Playfair matrix.

    Parameters:

    matrix (list): The 5x5 matrix for the Playfair cipher.

    char (str): The character to find in the matrix.

    Returns:

    tuple: The row and column of the character in the matrix.

    """

    for row in range(5):

        for col in range(5):

            if matrix[row][col] == char:

                return row, col

    return None

def playfair\_encrypt(text, key):

    """

    Encrypt the plain text using the Playfair cipher.

    Parameters:

    text (str): The input text to be encrypted.

    key (str): The key for the Playfair cipher.

    Returns:

    str: The encrypted text.

    """

    text = text.upper().replace("J", "I").replace(" ", "")

    if len(text) % 2 != 0:

        text += "X"

    matrix = generate\_playfair\_matrix(key)

    encrypted\_text = ""

    for i in range(0, len(text), 2):

        char1, char2 = text[i], text[i + 1]

        if char1 == char2:

            char2 = 'X'

        row1, col1 = find\_position(matrix, char1)

        row2, col2 = find\_position(matrix, char2)

        if row1 == row2:

            encrypted\_text += matrix[row1][(col1 + 1) % 5]

            encrypted\_text += matrix[row2][(col2 + 1) % 5]

        elif col1 == col2:

            encrypted\_text += matrix[(row1 + 1) % 5][col1]

            encrypted\_text += matrix[(row2 + 1) % 5][col2]

        else:

            encrypted\_text += matrix[row1][col2]

            encrypted\_text += matrix[row2][col1]

    return encrypted\_text

def playfair\_decrypt(text, key):

    """

    Decrypt the encrypted text using the Playfair cipher.

    Parameters:

    text (str): The input text to be decrypted.

    key (str): The key for the Playfair cipher.

    Returns:

    str: The decrypted text.

    """

    text = text.upper().replace("J", "I").replace(" ", "")

    matrix = generate\_playfair\_matrix(key)

    decrypted\_text = ""

    for i in range(0, len(text), 2):

        char1, char2 = text[i], text[i + 1]

        row1, col1 = find\_position(matrix, char1)

        row2, col2 = find\_position(matrix, char2)

        if row1 == row2:

            decrypted\_text += matrix[row1][(col1 - 1) % 5]

            decrypted\_text += matrix[row2][(col2 - 1) % 5]

        elif col1 == col2:

            decrypted\_text += matrix[(row1 - 1) % 5][col1]

            decrypted\_text += matrix[(row2 - 1) % 5][col2]

        else:

            decrypted\_text += matrix[row1][col2]

            decrypted\_text += matrix[row2][col1]

    return decrypted\_text

def main():

    """

    The main function to run the menu-driven program.

    """

    while True:

        print("\nPlayfair Cipher Program")

        print("1. Encrypt")

        print("2. Decrypt")

        print("3. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':

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

            key = input("Enter the key: ")

            encrypted\_text = playfair\_encrypt(plain\_text, key)

            print(f"\nEncrypted Text: {encrypted\_text}")

        elif choice == '2':

            encrypted\_text = input("\nEnter the encrypted text: ")

            key = input("Enter the key: ")

            decrypted\_text = playfair\_decrypt(encrypted\_text, key)

            print(f"\nDecrypted Text: {decrypted\_text}")

        elif choice == '3':

            print("Exiting the program.")

            break

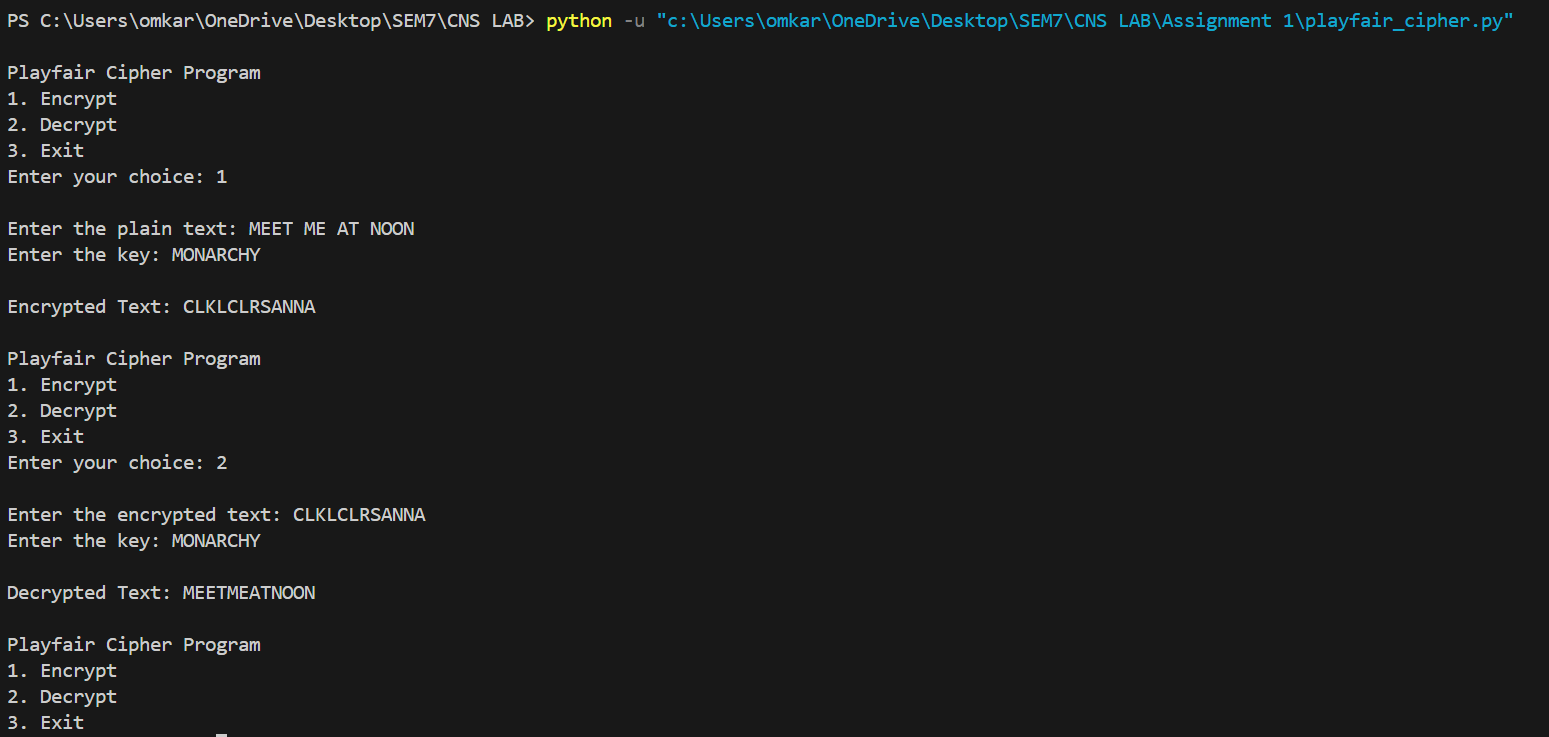
        else:

            print("Invalid choice. Please try again.")

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

    main()

**Output:**

****

**Advantages:**

* **Improved Security**: More secure than Caesar Cipher as it encrypts digraphs (pairs of letters).
* **Simplicity**: Slightly more complex but still relatively easy to implement.

**Disadvantages:**

* **Key Management**: Requires a good keyword and matrix setup.
* **Vulnerability**: Can still be broken with modern techniques like frequency analysis of digraphs.

**c. Hill Cipher**

**Ans:**

The Hill Cipher is a polygraphic substitution cipher that uses linear algebra. It encrypts blocks of text (usually 2x2 or 3x3 matrices) by multiplying them with a key matrix. The key matrix must be invertible for decryption. This method allows for more complex encryption compared to simple substitution ciphers.

**Python code:**

import numpy as np

def mod\_inverse(matrix, modulus):

    """

    Calculate the modular inverse of a matrix under a given modulus.

    Parameters:

    matrix (numpy.ndarray): The matrix to invert.

    modulus (int): The modulus value.

    Returns:

    numpy.ndarray: The modular inverse of the matrix.

    """

    det = int(np.round(np.linalg.det(matrix)))

    det\_inv = pow(det, -1, modulus)

    matrix\_modulus\_inv = (

        det\_inv \* np.round(det \* np.linalg.inv(matrix)).astype(int) % modulus

    )

    return matrix\_modulus\_inv

def hill\_encrypt(text, key):

    """

    Encrypt the plain text using the Hill cipher.

    Parameters:

    text (str): The input text to be encrypted.

    key (list of int): The key for the Hill cipher as a flat list.

    Returns:

    str: The encrypted text.

    """

    size = int(len(key) \*\* 0.5)

    key\_matrix = np.array(key).reshape(size, size)

    modulus = 26

    text\_vector = np.array([ord(char) - ord('A') for char in text])

    text\_vector = text\_vector.reshape(-1, size).T

    encrypted\_vector = (np.dot(key\_matrix, text\_vector) % modulus).T

    encrypted\_text = ''.join(chr(num + ord('A')) for num in encrypted\_vector.flatten())

    return encrypted\_text

def hill\_decrypt(text, key):

    """

    Decrypt the encrypted text using the Hill cipher.

    Parameters:

    text (str): The input text to be decrypted.

    key (list of int): The key for the Hill cipher as a flat list.

    Returns:

    str: The decrypted text.

    """

    size = int(len(key) \*\* 0.5)

    key\_matrix = np.array(key).reshape(size, size)

    modulus = 26

    key\_matrix\_inv = mod\_inverse(key\_matrix, modulus)

    text\_vector = np.array([ord(char) - ord('A') for char in text])

    text\_vector = text\_vector.reshape(-1, size).T

    decrypted\_vector = (np.dot(key\_matrix\_inv, text\_vector) % modulus).T

    decrypted\_text = ''.join(chr(int(num) + ord('A')) for num in decrypted\_vector.flatten())

    return decrypted\_text

def main():

    """

    The main function to run the menu-driven program.

    """

    while True:

        print("\nHill Cipher Program")

        print("1. Encrypt")

        print("2. Decrypt")

        print("3. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':

            plain\_text = input("\nEnter the plain text (length multiple of key matrix size): ").upper().replace(" ", "")

            key = input("Enter the key matrix (comma-separated integers, e.g., '2,4,5,9' for 2x2 matrix): ")

            key\_matrix = list(map(int, key.split(',')))

            size = int(len(key\_matrix) \*\* 0.5)

            if len(plain\_text) % size != 0:

                print("Error: The length of the plain text must be a multiple of the key matrix size.")

                continue

            encrypted\_text = hill\_encrypt(plain\_text, key\_matrix)

            print(f"\nEncrypted Text: {encrypted\_text}")

        elif choice == '2':

            encrypted\_text = input("\nEnter the encrypted text: ").upper().replace(" ", "")

            key = input("Enter the key matrix (comma-separated integers, e.g., '2,4,5,9' for 2x2 matrix): ")

            key\_matrix = list(map(int, key.split(',')))

            size = int(len(key\_matrix) \*\* 0.5)

            if len(encrypted\_text) % size != 0:

                print("Error: The length of the encrypted text must be a multiple of the key matrix size.")

                continue

            decrypted\_text = hill\_decrypt(encrypted\_text, key\_matrix)

            print(f"\nDecrypted Text: {decrypted\_text}")

        elif choice == '3':

            print("Exiting the program.")

            break

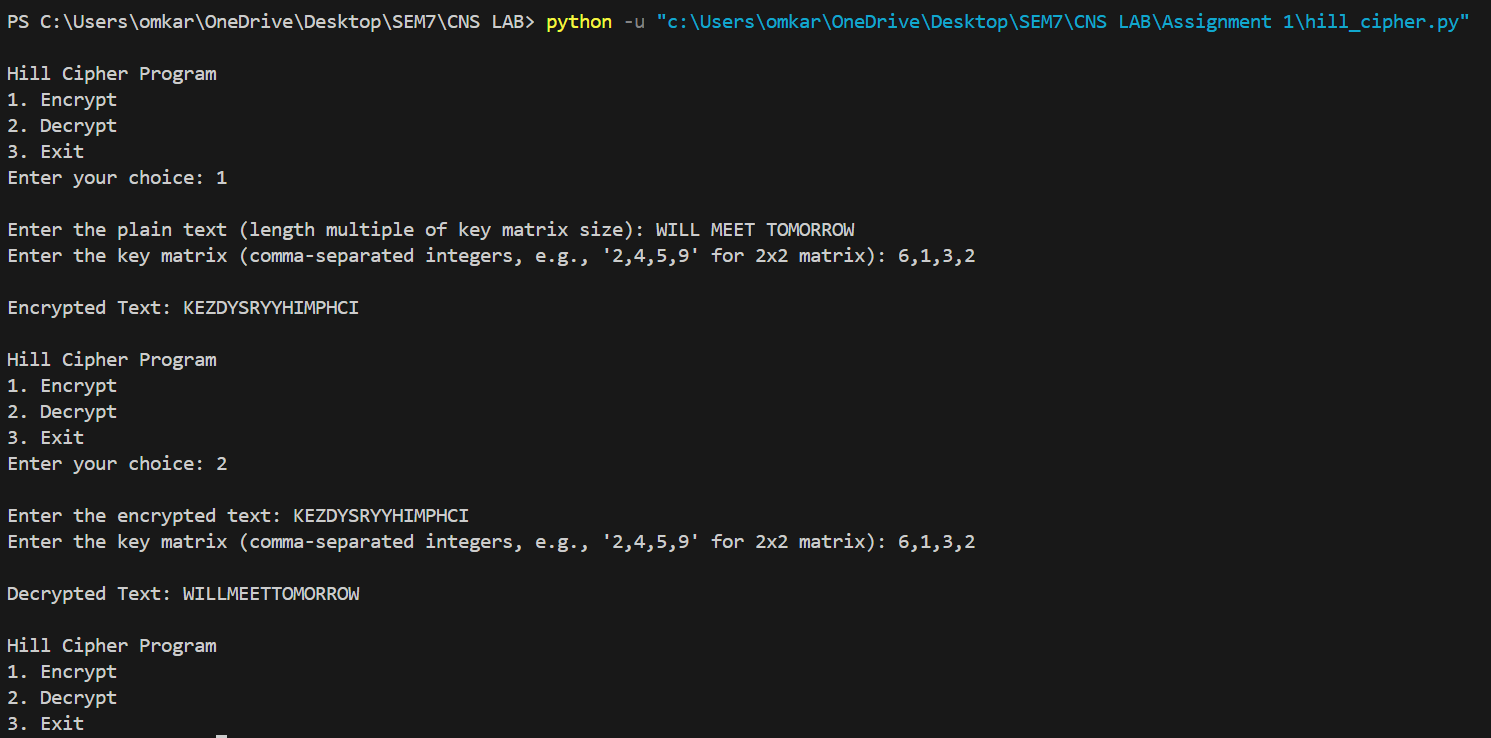
        else:

            print("Invalid choice. Please try again.")

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

    main()

**Output:**

****

**Advantages:**

* **Polyalphabetic**: Uses linear algebra for encryption, making it stronger than monoalphabetic ciphers.
* **Higher Complexity**: More resistant to frequency analysis due to the use of matrices.

**Disadvantages:**

* **Complexity**: Requires matrix inversion and modular arithmetic, which can be cumbersome.
* **Key Size**: Key matrix must be invertible, and the length of the plaintext must be a multiple of the matrix size.

**d. Vigenere cipher**

**Ans:**

The Vigenère Cipher is a method of encrypting text using a keyword. It works by shifting each letter in the plaintext by an amount determined by the corresponding letter in the keyword. The key repeats itself if it's shorter than the plaintext.

**How It Works:**

1. **Keyword**: Choose a keyword (e.g., "KEY").
2. **Encryption**:
   * Write the keyword repeatedly above the plaintext.
   * Shift each letter in the plaintext by the position of the corresponding letter in the keyword (A=0, B=1, ..., Z=25).
3. **Decryption**:
   * Use the same keyword to reverse the shifts and recover the plaintext.

**Python Code:**

def vigenere\_encrypt(plain\_text, key):

    """

    Encrypt the plain text using the Vigenere cipher.

    Parameters:

    plain\_text (str): The input text to be encrypted.

    key (str): The key for the Vigenere cipher.

    Returns:

    str: The encrypted text.

    """

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

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

    key\_length = len(key)

    encrypted\_text = ""

    for i, char in enumerate(plain\_text):

        if char.isalpha():

            shift = ord(key[i % key\_length]) - ord('A')

            encrypted\_char = chr((ord(char) - ord('A') + shift) % 26 + ord('A'))

            encrypted\_text += encrypted\_char

        else:

            encrypted\_text += char

    return encrypted\_text

def vigenere\_decrypt(cipher\_text, key):

    """

    Decrypt the cipher text using the Vigenere cipher.

    Parameters:

    cipher\_text (str): The input text to be decrypted.

    key (str): The key for the Vigenere cipher.

    Returns:

    str: The decrypted text.

    """

    cipher\_text = cipher\_text.upper().replace(" ", "")

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

    key\_length = len(key)

    decrypted\_text = ""

    for i, char in enumerate(cipher\_text):

        if char.isalpha():

            shift = ord(key[i % key\_length]) - ord('A')

            decrypted\_char = chr((ord(char) - ord('A') - shift + 26) % 26 + ord('A'))

            decrypted\_text += decrypted\_char

        else:

            decrypted\_text += char

    return decrypted\_text

def main():

    """

    The main function to run the menu-driven program.

    """

    while True:

        print("\nVigenere Cipher Program")

        print("1. Encrypt")

        print("2. Decrypt")

        print("3. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':

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

            key = input("Enter the key: ")

            encrypted\_text = vigenere\_encrypt(plain\_text, key)

            print(f"\nEncrypted Text: {encrypted\_text}")

        elif choice == '2':

            encrypted\_text = input("\nEnter the encrypted text: ")

            key = input("Enter the key: ")

            decrypted\_text = vigenere\_decrypt(encrypted\_text, key)

            print(f"\nDecrypted Text: {decrypted\_text}")

        elif choice == '3':

            print("Exiting the program.")

            break

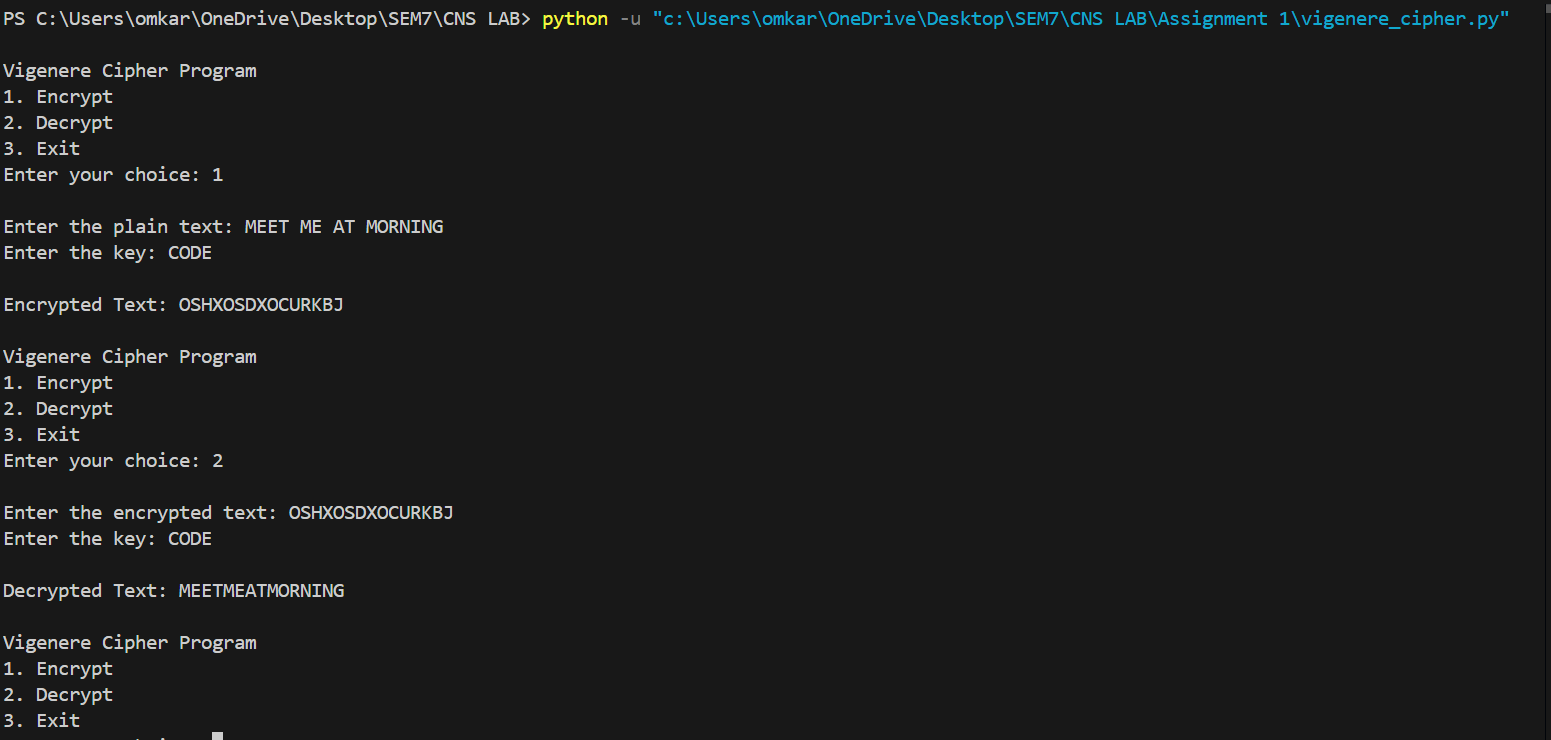
        else:

            print("Invalid choice. Please try again.")

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

    main()

**Output:**

****

**Advantages:**

* **Polyalphabetic**: Uses a keyword to shift letters, making it more secure than Caesar Cipher.
* **Improved Security**: Harder to crack with frequency analysis if the keyword is long and complex.

**Disadvantages:**

* **Keyword Management**: Security depends on the keyword length and complexity.
* **Vulnerabilities**: Can be broken with techniques like the Kasiski examination or frequency analysis if the keyword is short.