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:

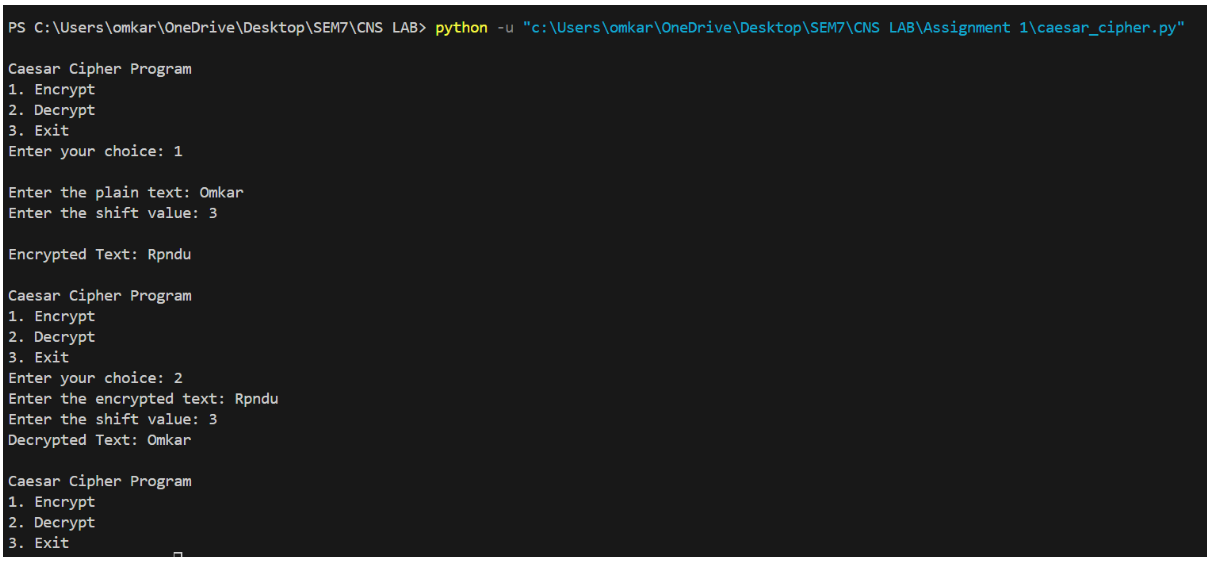
The Caesar Cipher is a simple encryption technique where each letter in a message is shifted by a fixed number of positions in the alphabet. For example, with a shift of 3, "A" becomes "D," "B" becomes "E," and so on. It's one of the oldest known ciphers and is easy to implement but also easy to break.

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'))
```

```python
            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:



```
PS C:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB> python -u "c:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB\Assignment 1\caesar_cipher.py"

Caesar Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1

Enter the plain text: Omkar
Enter the shift value: 3

Encrypted Text: Rpndu

Caesar Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2
Enter the encrypted text: Rpndu
Enter the shift value: 3
Decrypted Text: Omkar

Caesar Cipher Program
1. Encrypt
2. Decrypt
3. Exit
```

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:

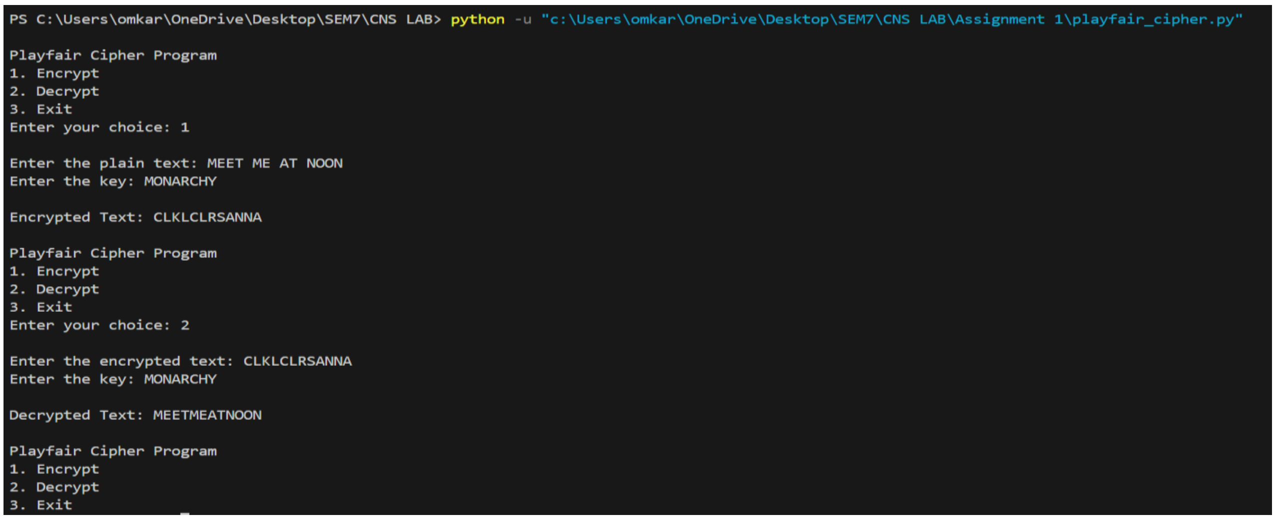
The Playfair Cipher is a digraph substitution cipher that encrypts pairs of letters. It uses a 5x5 matrix of letters created from a keyword. To encrypt, locate each letter pair in the matrix and swap or substitute based on their positions. It's more secure than simple substitution ciphers because it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

Python code:

```python
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.")
```

```python
if __name__ == "__main__":
    main()
```

Output:



```
PS C:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB> python -u "c:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB\Assignment 1\playfair_cipher.py"

Playfair Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1

Enter the plain text: MEET ME AT NOON
Enter the key: MONARCHY

Encrypted Text: CLKLCLRSANNA

Playfair Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2

Enter the encrypted text: CLKLCLRSANNA
Enter the key: MONARCHY

Decrypted Text: MEETMEATNOON

Playfair Cipher Program
1. Encrypt
2. Decrypt
3. Exit
```

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:

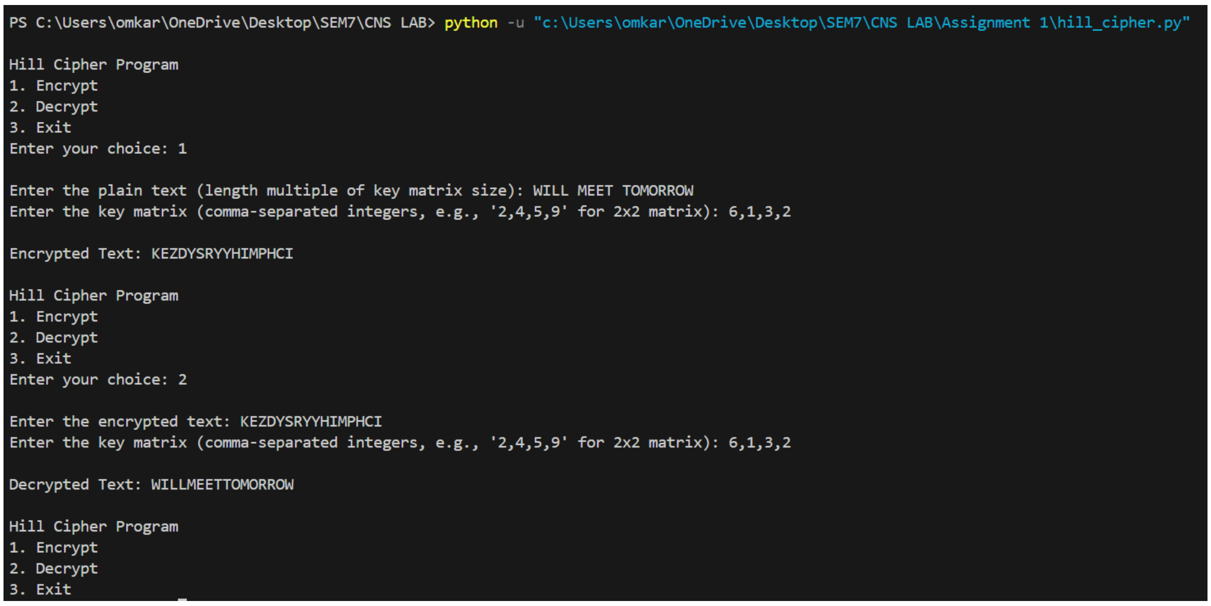
```python
import numpy as np
```

```python
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:



```
PS C:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB> python -u "c:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB\Assignment 1\hill_cipher.py"

Hill Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1

Enter the plain text (length multiple of key matrix size): WILL MEET TOMORROW
Enter the key matrix (comma-separated integers, e.g., '2,4,5,9' for 2x2 matrix): 6,1,3,2

Encrypted Text: KEZDYSRYYHIMPHCI

Hill Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2

Enter the encrypted text: KEZDYSRYYHIMPHCI
Enter the key matrix (comma-separated integers, e.g., '2,4,5,9' for 2x2 matrix): 6,1,3,2

Decrypted Text: WILLMEETTOMORROW

Hill Cipher Program
1. Encrypt
2. Decrypt
3. Exit
```

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).

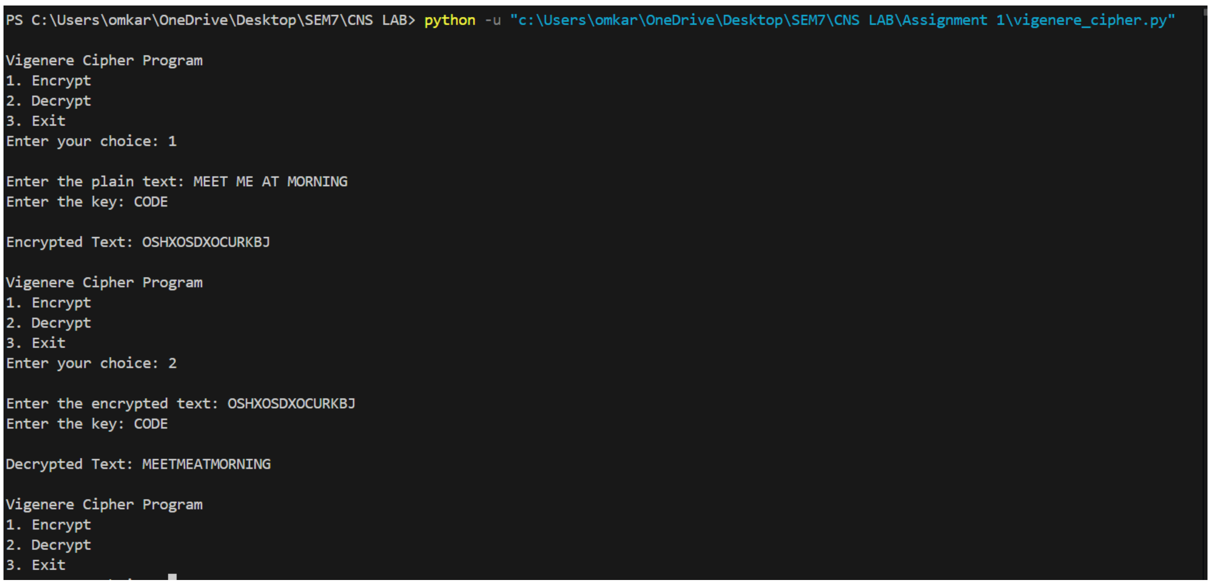
3. **Decryption**:
    - Use the same keyword to reverse the shifts and recover the plaintext.

Python Code:

```python
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: ")
```

```python
            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:



```
PS C:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB> python -u "c:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB\Assignment 1\vigenere_cipher.py"

Vigenere Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1

Enter the plain text: MEET ME AT MORNING
Enter the key: CODE

Encrypted Text: OSHXOSDXOCURKBJ

Vigenere Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2

Enter the encrypted text: OSHXOSDXOCURKBJ
Enter the key: CODE

Decrypted Text: MEETMEATMORNING

Vigenere Cipher Program
1. Encrypt
2. Decrypt
3. Exit
```

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.