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

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

**Date: 05/08/2024**

**Assignment 2**

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

**1. Perform encryption and decryption using following transposition techniques**

**a. Rail fence**

**Ans:**

The Rail Fence Cipher is a type of transposition cipher where the plain text is written in a zigzag pattern across multiple "rails" (rows) and then read row by row to create the cipher text. Decryption involves reconstructing the zigzag pattern to retrieve the original message.

**Python code:**

def rail\_fence\_encrypt(plain\_text, key):

    """

    Encrypt the plain text using the Rail Fence cipher.

    Parameters:

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

    key (int): The number of rails (rows) for the Rail Fence cipher.

    Returns:

    str: The encrypted text.

    """

    # Create a list of strings to represent each rail

    rail = ['' for \_ in range(key)]

    row, direction = 0, 1

    # Distribute the characters across the rails in a zigzag pattern

    for char in plain\_text:

        rail[row] += char

        row += direction

        # Reverse direction when we reach the top or bottom rail

        if row == 0 or row == key - 1:

            direction \*= -1

    # Concatenate all the rails to get the encrypted text

    return ''.join(rail)

def rail\_fence\_decrypt(cipher\_text, key):

    """

    Decrypt the cipher text using the Rail Fence cipher.

    Parameters:

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

    key (int): The number of rails (rows) for the Rail Fence cipher.

    Returns:

    str: The decrypted text.

    """

    # Determine the length of each rail in the zigzag pattern

    pattern = [0] \* len(cipher\_text)

    row, direction = 0, 1

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

        pattern[i] = row

        row += direction

        # Reverse direction when we reach the top or bottom rail

        if row == 0 or row == key - 1:

            direction \*= -1

    # Reconstruct the rails from the cipher text

    rail\_lengths = [pattern.count(i) for i in range(key)]

    rail\_chars = ['' for \_ in range(key)]

    pos = 0

    for i in range(key):

        rail\_chars[i] = cipher\_text[pos:pos + rail\_lengths[i]]

        pos += rail\_lengths[i]

    # Reconstruct the original message by following the zigzag pattern

    result = []

    row\_pointers = [0] \* key

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

        result.append(rail\_chars[pattern[i]][row\_pointers[pattern[i]]])

        row\_pointers[pattern[i]] += 1

    return ''.join(result)

def main():

    """

    The main function to run the menu-driven program.

    """

    while True:

        print("\nRail Fence 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: ").replace(" ", "")

            key = int(input("Enter the number of rails: "))

            encrypted\_text = rail\_fence\_encrypt(plain\_text, key)

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

        elif choice == '2':

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

            key = int(input("Enter the number of rails: "))

            decrypted\_text = rail\_fence\_decrypt(cipher\_text, key)

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

        elif choice == '3':

            print("Exiting the program.")

            break

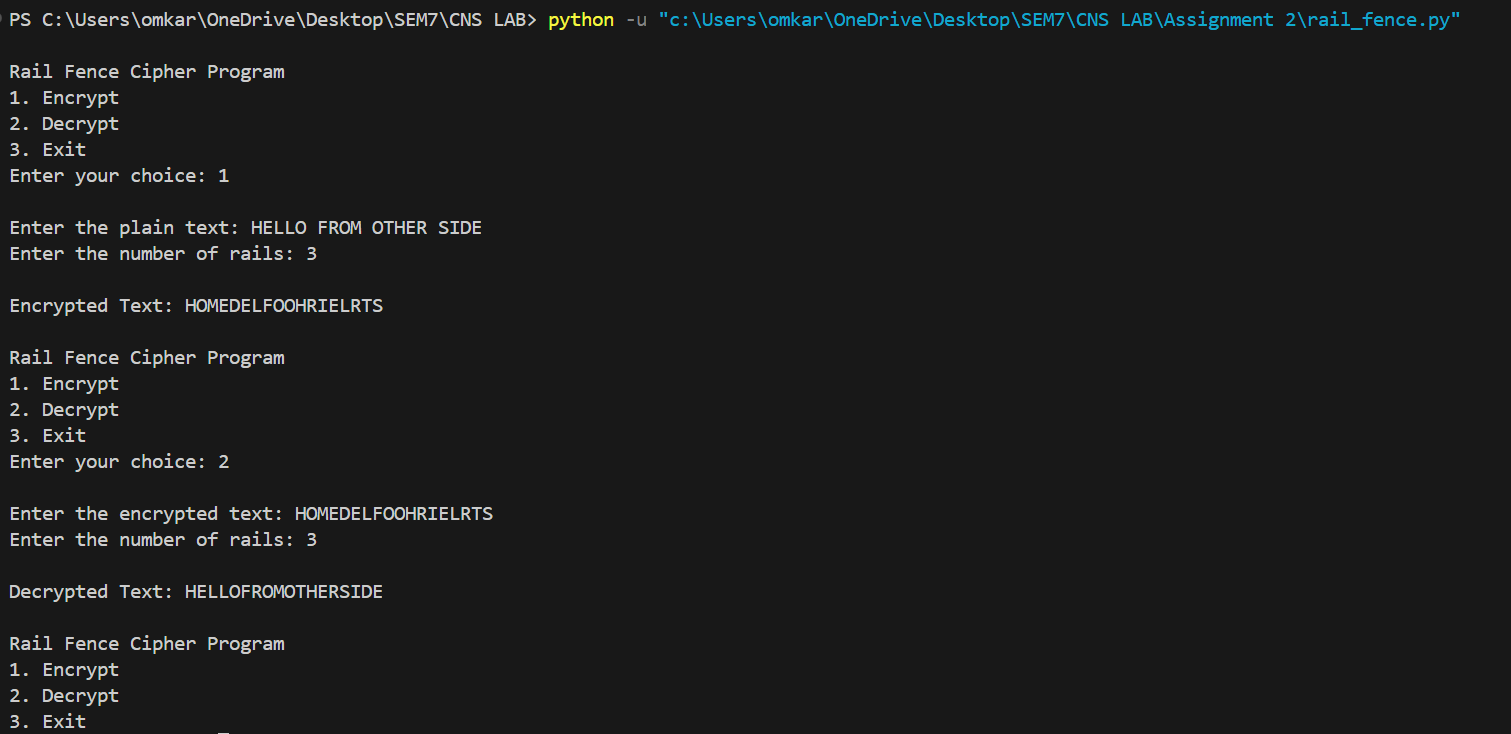
        else:

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

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

    main()

**Output:**

****

**Advantages:**

* **Simplicity**: Easy to understand and implement.
* **Low Computation**: Requires minimal computational resources for encryption and decryption.

**Disadvantages:**

* **Weak Security**: Very easy to break with simple analysis or known-plaintext attacks.
* **Pattern Recognition**: The regular zigzag pattern makes it susceptible to pattern recognition, which can be exploited to decode the message.

**b. row and Column Transformation**

**Ans:**

Row and column transformation is a type of transposition cipher where the message is written in a grid (matrix) and the order of rows and columns is changed according to a key.

**Row Transposition**: Encrypts text by writing it into rows of a grid, then permuting the columns according to a specific key.

**Column Transposition**: Encrypts text by writing it into columns of a grid, then permuting the rows according to a specific key.

**How It Works**:

1. **Write** the plaintext into a grid according to the number of rows or columns.
2. **Permute** the rows or columns based on the key.
3. **Read** off the text in the new order to get the ciphertext.

**Python code:**

import math

def create\_matrix(text, key\_len):

    """

    Create a matrix from the text with the specified number of columns (key length).

    """

    rows = math.ceil(len(text) / key\_len)

    matrix = [['' for \_ in range(key\_len)] for \_ in range(rows)]

    k = 0

    for i in range(rows):

        for j in range(key\_len):

            if k < len(text):

                matrix[i][j] = text[k]

                k += 1

            else:

                matrix[i][j] = 'X'  # Padding with 'X' if the matrix is not full

    return matrix

def row\_column\_encrypt(plain\_text, row\_key, col\_key):

    """

    Encrypt the plain text using row and column transformation.

    Parameters:

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

    row\_key (list): The key to rearrange rows.

    col\_key (list): The key to rearrange columns.

    Returns:

    str: The encrypted text.

    """

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

    key\_len = len(col\_key)

    # Create the matrix from the plain text

    matrix = create\_matrix(plain\_text, key\_len)

    # Apply the row key

    row\_matrix = [matrix[i] for i in row\_key]

    # Apply the column key

    encrypted\_text = ""

    for row in row\_matrix:

        encrypted\_row = [row[j] for j in col\_key]

        encrypted\_text += ''.join(encrypted\_row)

    return encrypted\_text

def row\_column\_decrypt(cipher\_text, row\_key, col\_key):

    """

    Decrypt the cipher text using row and column transformation.

    Parameters:

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

    row\_key (list): The key to rearrange rows.

    col\_key (list): The key to rearrange columns.

    Returns:

    str: The decrypted text.

    """

    key\_len = len(col\_key)

    rows = len(cipher\_text) // key\_len

    # Create the matrix to store the rearranged cipher text

    matrix = [['' for \_ in range(key\_len)] for \_ in range(rows)]

    k = 0

    # Arrange the cipher text in the matrix based on the column key

    for i in range(len(row\_key)):

        for j in col\_key:

            matrix[row\_key[i]][j] = cipher\_text[k]

            k += 1

    # Read the decrypted text row by row

    decrypted\_text = ""

    for i in range(rows):

        decrypted\_text += ''.join(matrix[i])

    return decrypted\_text

def main():

    """

    The main function to run the menu-driven program.

    """

    while True:

        print("\nRow and Column Transformation 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: ")

            row\_key = list(map(int, input("Enter the row key as a sequence of numbers (e.g., 2 0 1): ").split()))

            col\_key = list(map(int, input("Enter the column key as a sequence of numbers (e.g., 1 0 2): ").split()))

            encrypted\_text = row\_column\_encrypt(plain\_text, row\_key, col\_key)

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

        elif choice == '2':

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

            row\_key = list(map(int, input("Enter the row key as a sequence of numbers (e.g., 2 0 1): ").split()))

            col\_key = list(map(int, input("Enter the column key as a sequence of numbers (e.g., 1 0 2): ").split()))

            decrypted\_text = row\_column\_decrypt(cipher\_text, row\_key, col\_key)

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

        elif choice == '3':

            print("Exiting the program.")

            break

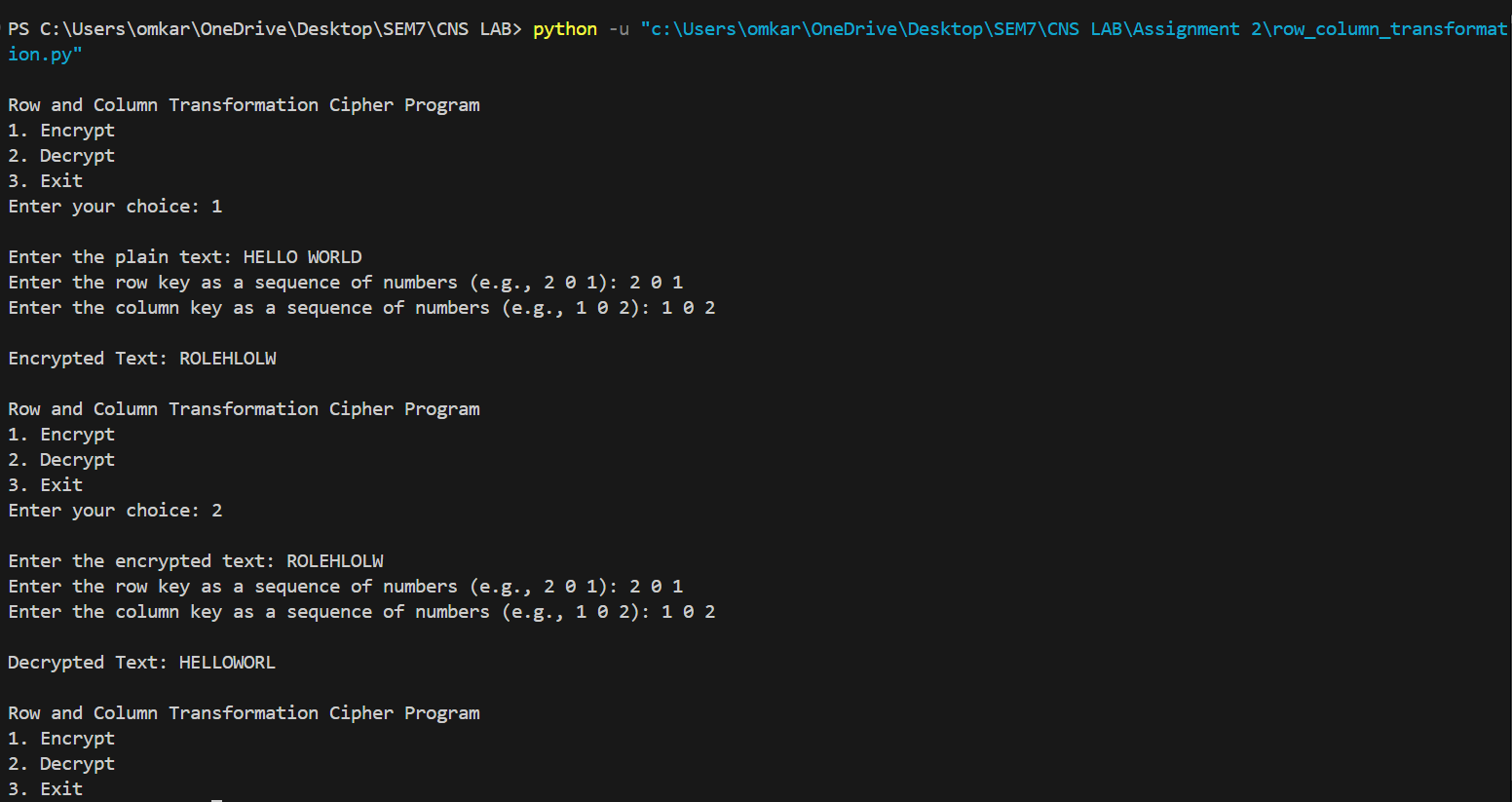
        else:

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

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

    main()

**Output:**

****

**Advantages**:

* **Increased Security**: More complex than simple transpositions.
* **Flexibility**: Key-based rearrangement can add security.

**Disadvantages**:

* **Complexity**: Can be more complex to implement and manage compared to simple ciphers.
* **Pattern Recognition**: Still susceptible to pattern analysis if not combined with other encryption methods.