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:

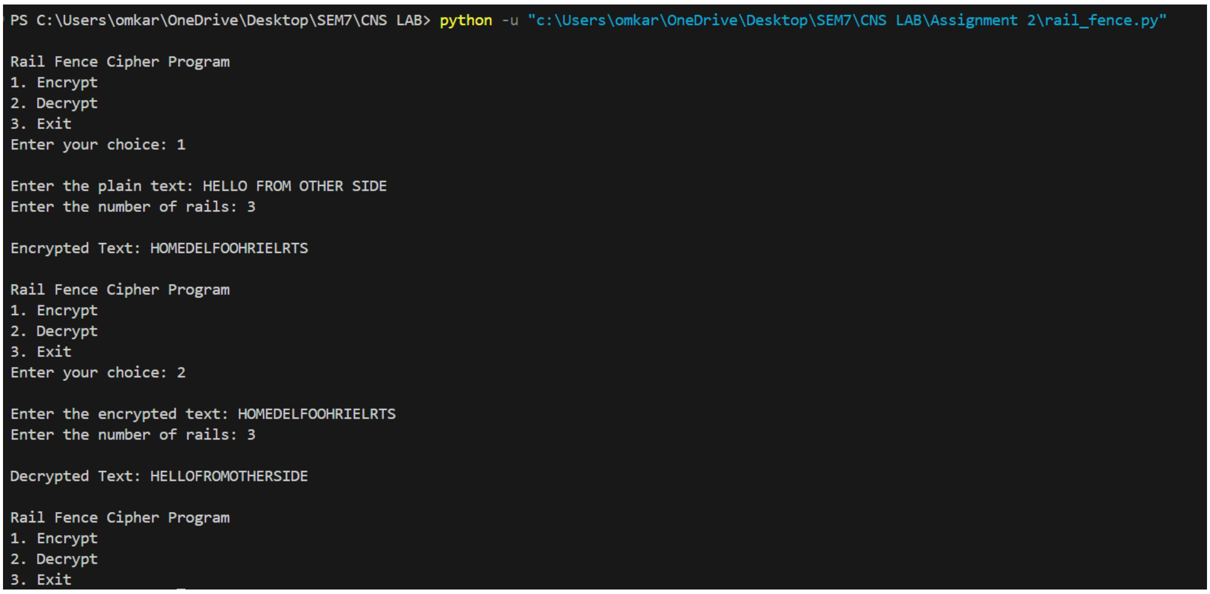
The Rail Fence Cipher is a type of transposition cipher where the plain text is written in a zigzag pattern across multiple "rails" (rows) and then read row by row to create the cipher text. Decryption involves reconstructing the zigzag pattern to retrieve the original message.

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:



```
PS C:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB> python -u "c:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB\Assignment 2\rail_fence.py"

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

Enter the plain text: HELLO FROM OTHER SIDE
Enter the number of rails: 3

Encrypted Text: HOMEDELFOOHRIELRTS

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

Enter the encrypted text: HOMEDELFOOHRIELRTS
Enter the number of rails: 3

Decrypted Text: HELLOFROMOTHERSIDE

Rail Fence Cipher Program
1. Encrypt
2. Decrypt
3. Exit
```

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.

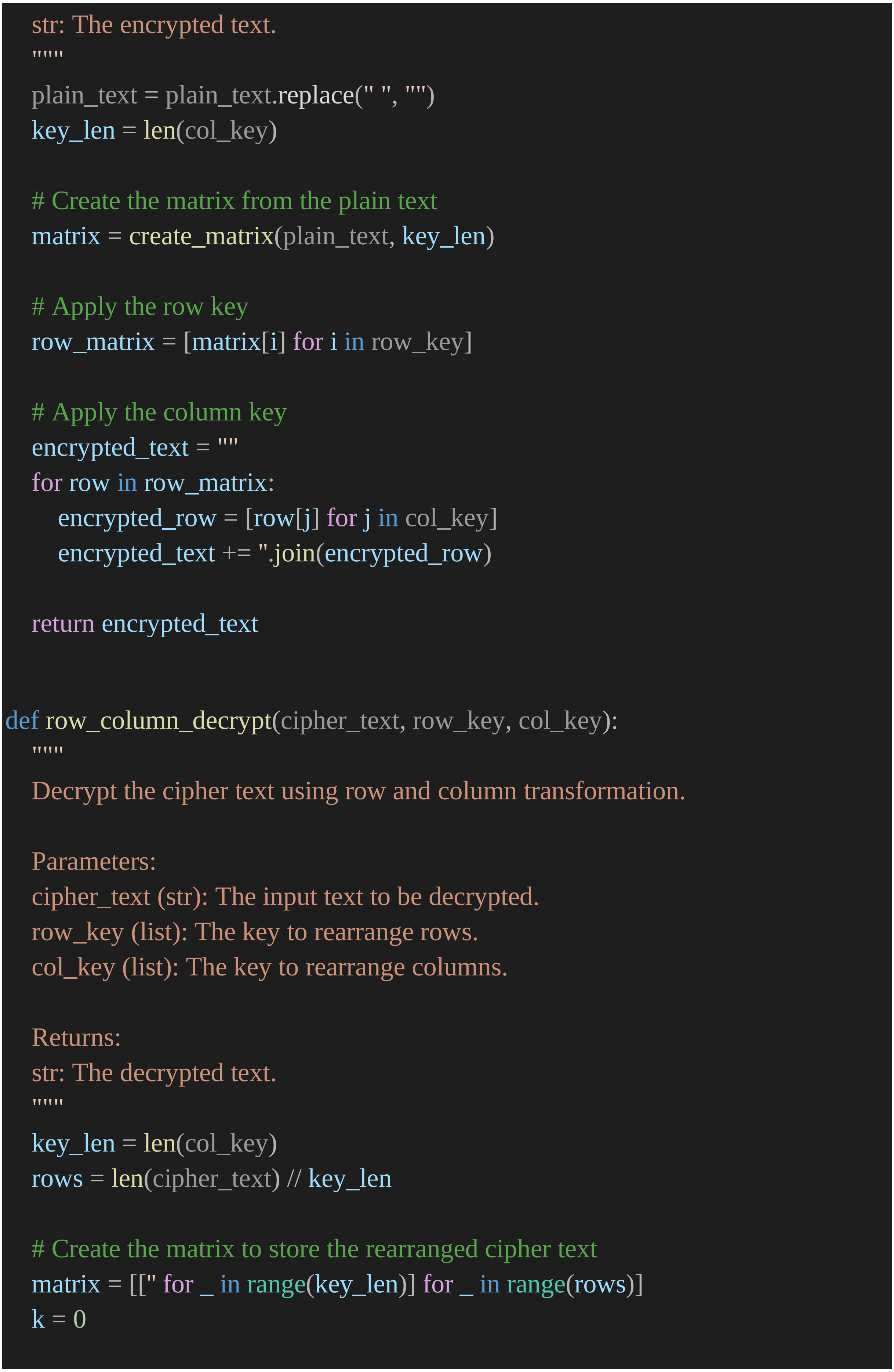
Column Transposition: Encrypts text by writing it into columns of a grid, then permuting the rows according to a specific key.

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.

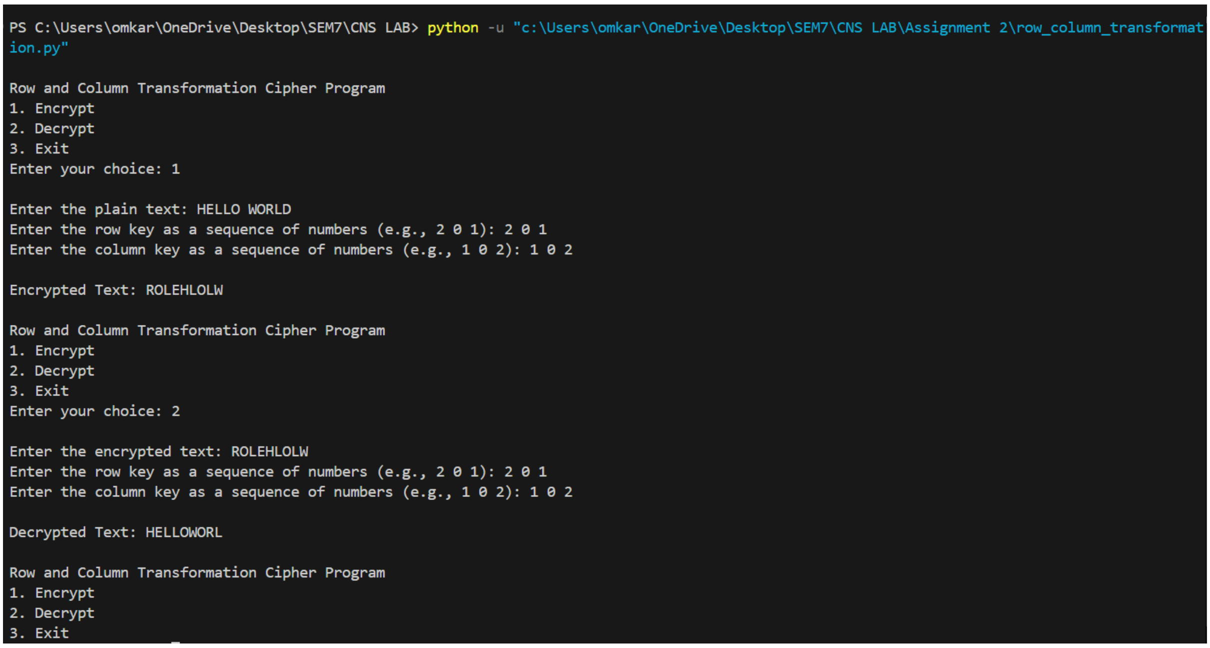
Python code:

```python
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:



```
PS C:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB> python -u "c:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB\Assignment 2\row_column_transformat
ion.py"

Row and Column Transformation Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1

Enter the plain text: HELLO WORLD
Enter the row key as a sequence of numbers (e.g., 2 0 1): 2 0 1
Enter the column key as a sequence of numbers (e.g., 1 0 2): 1 0 2

Encrypted Text: ROLEHLOLW

Row and Column Transformation Cipher Program
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2

Enter the encrypted text: ROLEHLOLW
Enter the row key as a sequence of numbers (e.g., 2 0 1): 2 0 1
Enter the column key as a sequence of numbers (e.g., 1 0 2): 1 0 2

Decrypted Text: HELLOWORL

Row and Column Transformation Cipher Program
1. Encrypt
2. Decrypt
3. Exit
```

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.