Experiment No. _2_____

Title: Transposition Cipher

Batch: B2　　**Roll No.: 16010421119**　　**Experiment No.:2**

Aim: To implement transposition cipher – Row transposition and column transposition cipher.

Resources needed: Windows/Linux

Theory

Pre Lab/ Prior Concepts:

Symmetric-key algorithms are a class of algorithms for cryptography that use the same cryptographic keys for both encryption of plaintext and decryption of cipher text. The keys may be identical or there may be a simple transformation to go between the two keys. The keys, in practice, represent a shared secret between two or more parties that can be used to maintain a private information link. This requirement that both parties have access to the secret key is one of the main drawbacks of symmetric key encryption, in comparison to public-key encryption. Symmetric-key encryption can use either stream ciphers or block ciphers. Transposition Cipher is block cipher. Ancient cryptographic systems are classified as: Substitution and Permutation/Transposition Ciphers.

Transposition Cipher/Permutation Cipher

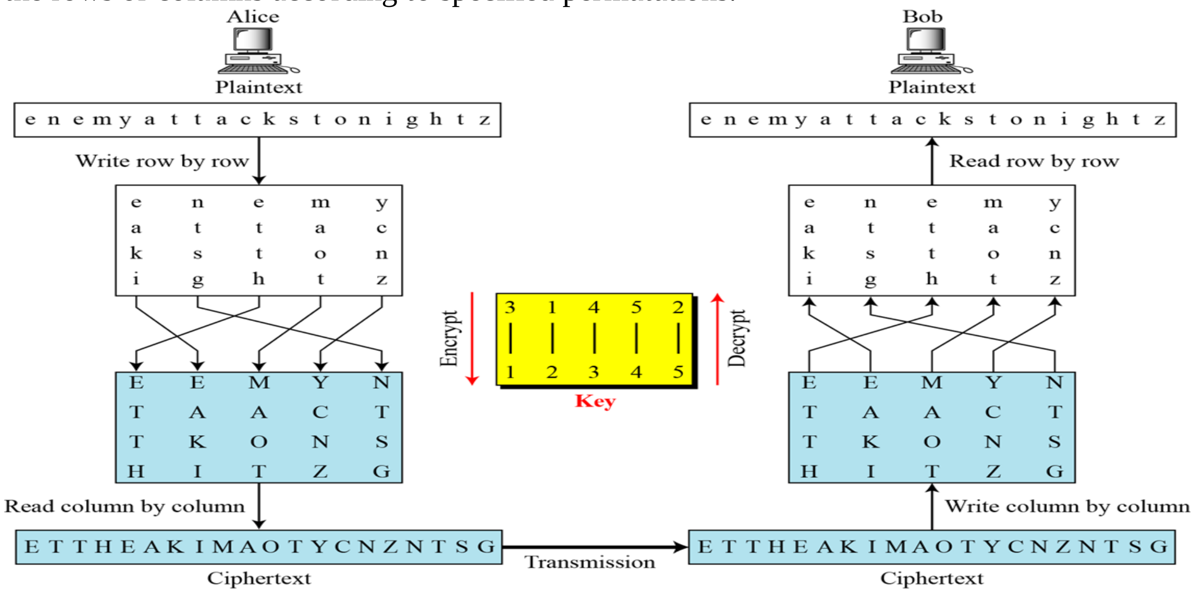
A transposition cipher rearranges (permutes) symbols in a block without altering actual values. It has the same frequency distribution as the original text .So it is easily recognizable.

EXAMPLE :

Plaintext: HELLO MY DEAR
Cipher text: ELHLMDOYAER

There are varieties of transposition ciphers like: keyless and keyed transposition ciphers.

Following figure shows the combination of both keyed and keyless. To encrypt with a transposition cipher, we first write the plaintext into a matrix of a given size and then permute the rows or columns according to specified permutations.



For the transposition, the key consists of the size of the matrix and the row or column permutations. The recipient who knows the key can simply put the cipher text into the appropriate sized matrix and undo the permutations to recover the plaintext.
Unlike a simple substitution, the transposition does nothing to disguise the letters that appear in the message But it does appear to thwart an attack that relies on the statistical information contained in the plaintext, since the plaintext statistics are disbursed throughout the cipher text. The double transposition is not a trivial cipher to break.

Activity :
Implement

1) **Row transposition cipher**
2) **Column transposition cipher**
3) **Double transposition cipher (row followed by column)**

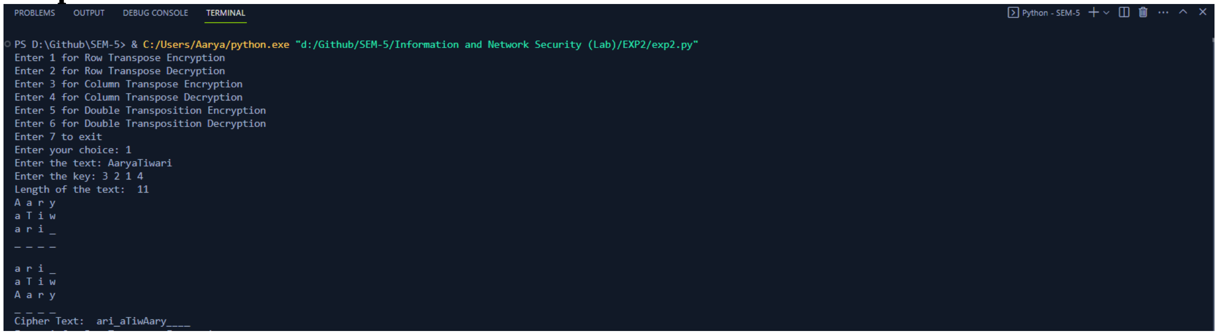
Implementation:
The program should have encryption function and decryption function for each cipher. Function should take message and a key as input from the user and display the expected output.

Results: (Program with output as per the format)

1. **Row transposition cipher (Encryption)**
   Code:

```
def row_transpose_encrypt(text, key):
    text_length = len(text)
    print("Length of the text: ", text_length)

    key1 = key.split(" ")

    for i in range(text_length):
        if i*i >= text_length:
            size = i
            break
    row = size
    col = size
    twoD = [[" " for i in range(col)] for j in range(row)]
    index = 0
    for i in range(row):
        for j in range(col):
            if index < text_length:
                twoD[i][j] = text[index]
                index += 1
            else:
                twoD[i][j] = '_'

    printArray(twoD, row)
    print()
    twoD_2 = twoD.copy()

    for i in range(len(key1)):  # [3,1,2]
        twoD_2[i] = twoD[int(key1[i])-1]

    printArray(twoD_2, row)
    cipher_text = textGen(twoD_2, row)
    print("Cipher Text: ", cipher_text)
    return cipher_text
```

Output:

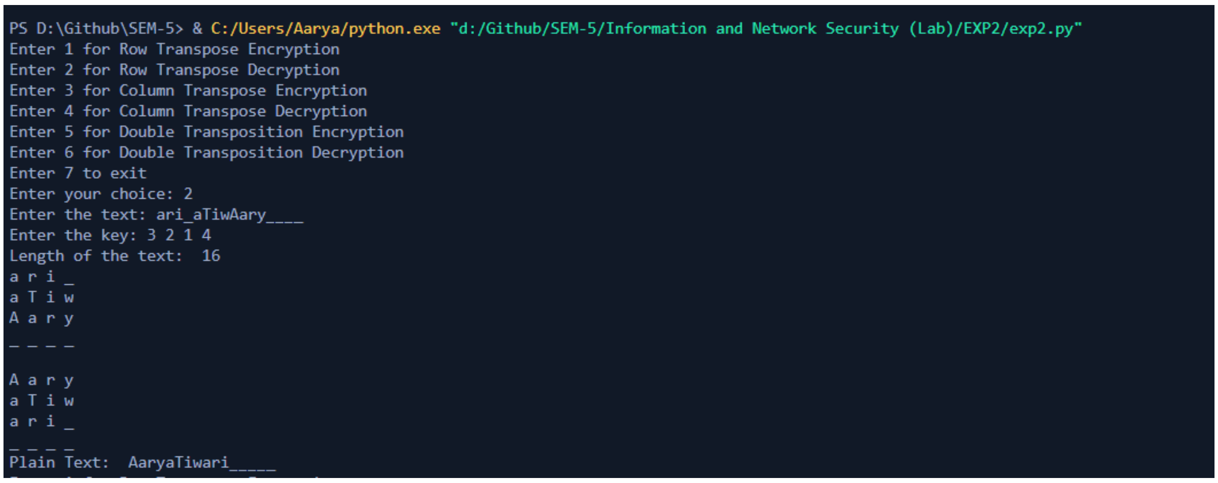


```
PS D:\Github\SEM-5> & C:/Users/Aarya/python.exe "d:/Github/SEM-5/Information and Network Security (Lab)/EXP2/exp2.py"
Enter 1 for Row Transpose Encryption
Enter 2 for Row Transpose Decryption
Enter 3 for Column Transpose Encryption
Enter 4 for Column Transpose Decryption
Enter 5 for Double Transposition Encryption
Enter 6 for Double Transposition Decryption
Enter 7 to exit
Enter your choice: 1
Enter the text: AaryaTiwari
Enter the key: 3 2 1 4
Length of the text:  11
A a r y
a T i w
a r i _
_ _ _ _

a r i _
a T i w
A a r y
_ _ _ _
Cipher Text:  ari_aTiwAary____
```

2. **Row transposition cipher (Decryption)**
   Code:

```
def row_transpose_decrypt(text, key):
    text_length = len(text)
    print("Length of the text: ", text_length)

    key1 = key.split(" ")

    for i in range(text_length):
        if i*i >= text_length:
            size = i
            break
    row = size
    col = size
    twoD = [[" " for i in range(col)] for j in range(row)]
    index = 0
    for i in range(row):
        for j in range(col):
            if index < text_length:
                twoD[i][j] = text[index]
                index += 1
            else:
                twoD[i][j] = '_'
    printArray(twoD, row)
    print()
    twoD_2 = twoD.copy()

    for i in range(len(key1)):
        twoD_2[int(key1[i])-1] = twoD[i]

    printArray(twoD_2, row)

    plain_text = textGen(twoD_2, row)
    print("Plain Text: ", plain_text)
    return plain_text
```

Output:

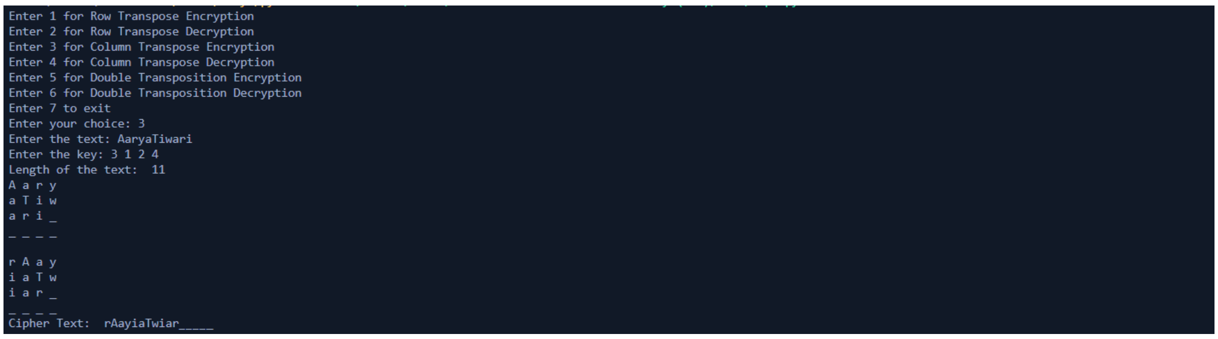


```
PS D:\Github\SEM-5> & C:/Users/Aarya/python.exe "d:/Github/SEM-5/Information and Network Security (Lab)/EXP2/exp2.py"
Enter 1 for Row Transpose Encryption
Enter 2 for Row Transpose Decryption
Enter 3 for Column Transpose Encryption
Enter 4 for Column Transpose Decryption
Enter 5 for Double Transposition Encryption
Enter 6 for Double Transposition Decryption
Enter 7 to exit
Enter your choice: 2
Enter the text: ari_aTiwAary____
Enter the key: 3 2 1 4
Length of the text:  16
a r i _
a T i w
A a r y
_ _ _ _

A a r y
a T i w
a r i _
_ _ _ _
Plain Text:  AaryaTiwari_____
```

3. **Column transposition cipher (Encrytion)**
   Code:

```
def column_transpose_encrypt(text, key):
    text_length = len(text)
    print("Length of the text: ", text_length)

    key1 = key.split(" ")

    for i in range(text_length):
        if i*i >= text_length:
            size = i
            break
    row = size
    col = size

    twoD = [[" " for i in range(col)] for j in range(row)]
    index = 0
    for i in range(row):
        for j in range(col):
            if index < text_length:
                twoD[i][j] = text[index]
                index += 1
            else:
                twoD[i][j] = '_'
    printArray(twoD, col)
    print()
    key1 = [int(i)-1 for i in key1]
    transposed_array = []
    for row in twoD:
        transposed_row = []
        for i in key1:
            transposed_row.append(row[i])
        transposed_array.append(transposed_row)

    printArray(transposed_array, col)
    cipher_text = textGen(transposed_array, col)
    print("Cipher Text: ", cipher_text)
    return cipher_text
```

Output:

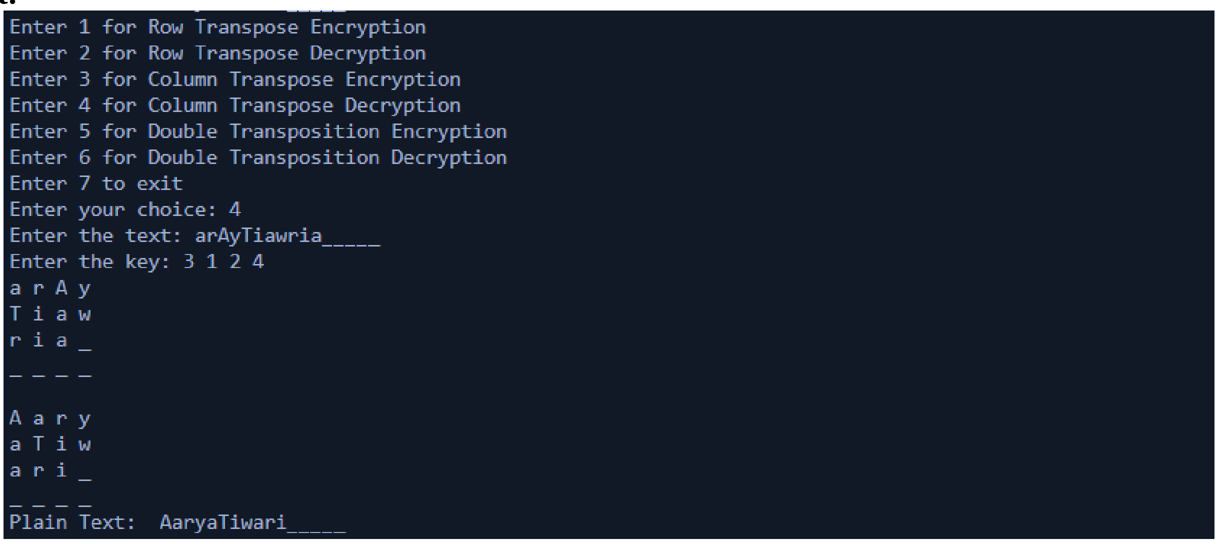


```
Enter 1 for Row Transpose Encryption
Enter 2 for Row Transpose Decryption
Enter 3 for Column Transpose Encryption
Enter 4 for Column Transpose Decryption
Enter 5 for Double Transposition Encryption
Enter 6 for Double Transposition Decryption
Enter 7 to exit
Enter your choice: 3
Enter the text: AaryaTiwari
Enter the key: 3 1 2 4
Length of the text:  11
A a r y
a T i w
a r i _
_ _ _ _

r A a y
i a T w
i a r _
_ _ _ _
Cipher Text:  rAayiaTwiar_____
```

4. **Column transposition cipher (Decryption)**
   Code:

```python
def column_transpose_decrypt(text, key):
    text_length = len(text)

    key1 = key.split(" ")

    for i in range(text_length):
        if i * i >= text_length:
            size = i
            break
    row = size
    col = size

    twoD = [[" " for i in range(col)] for j in range(row)]
    index = 0
    for i in range(row):
        for j in range(col):
            if index < text_length:
                twoD[i][j] = text[index]
                index += 1
            else:
                twoD[i][j] = '_'
    printArray(twoD, col)
    print()
    key1 = [int(i) - 1 for i in key1]
    original_array = []
    for i in range(row):
        original_row = []
        for j in range(col):
            original_row.append(twoD[i][key1[j]])
        original_array.append(original_row)
    printArray(original_array, col)

    plain_text = ""
    for row in original_array:
        for char in row:
            plain_text += char
    print("Plain Text: ", plain_text)
    return plain_text
```

Output:

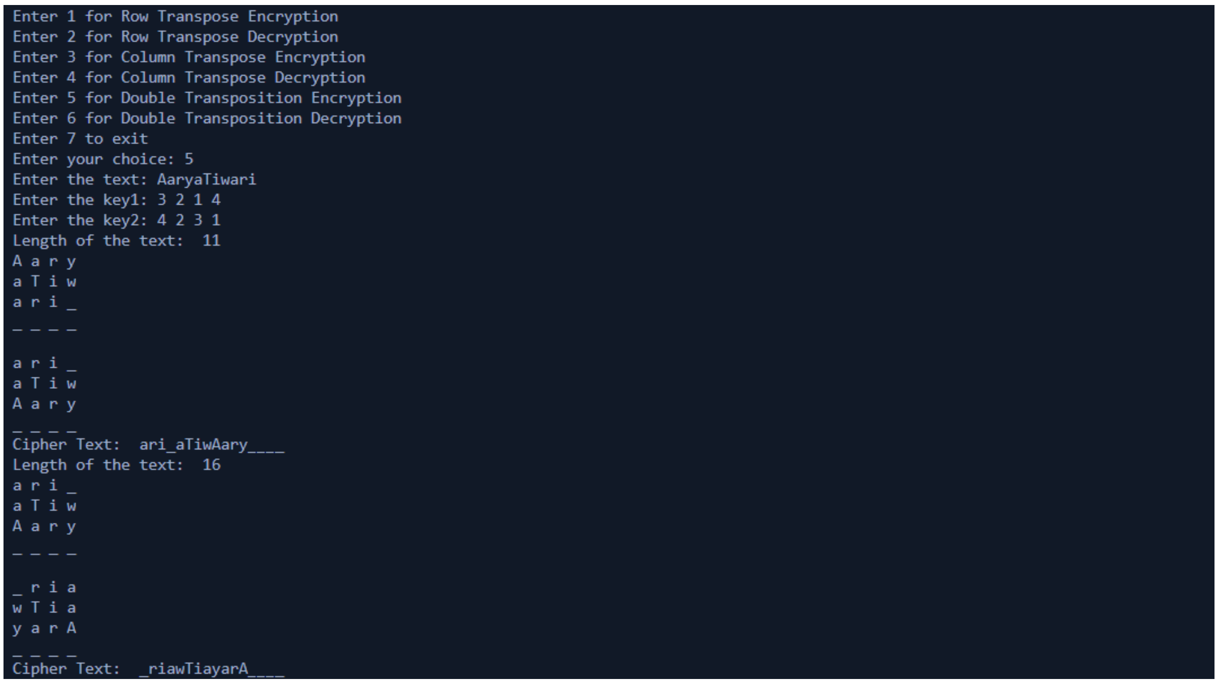


```
Enter 1 for Row Transpose Encryption
Enter 2 for Row Transpose Decryption
Enter 3 for Column Transpose Encryption
Enter 4 for Column Transpose Decryption
Enter 5 for Double Transposition Encryption
Enter 6 for Double Transposition Decryption
Enter 7 to exit
Enter your choice: 4
Enter the text: arAyTiawria_____
Enter the key: 3 1 2 4
a r A y
T i a w
r i a _
_ _ _ _

A a r y
a T i w
a r i _
_ _ _ _
Plain Text:  AaryaTiwari_____
```

5. **Double transposition cipher (row followed by column) – (Encryption)**
   Code:

```
def double_transposition_encrypt(text, key1, key2):
    intermediate_cipher = row_transpose_encrypt(text, key1)
    final_cipher = column_transpose_encrypt(intermediate_cipher, key2)
    return final_cipher
```

Output:

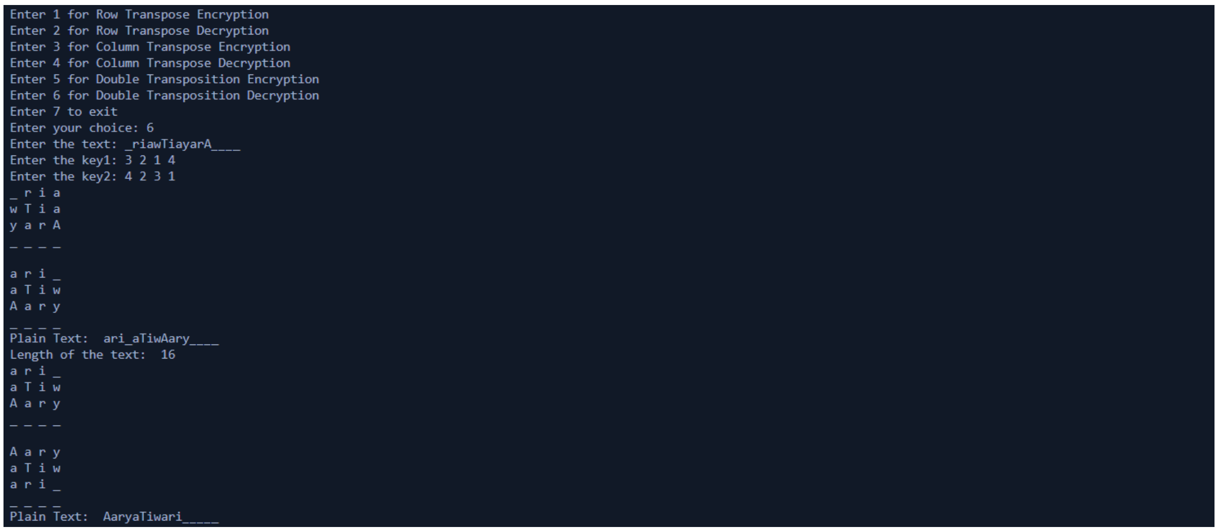


```
Enter 1 for Row Transpose Encryption
Enter 2 for Row Transpose Decryption
Enter 3 for Column Transpose Encryption
Enter 4 for Column Transpose Decryption
Enter 5 for Double Transposition Encryption
Enter 6 for Double Transposition Decryption
Enter 7 to exit
Enter your choice: 5
Enter the text: AaryaTiwari
Enter the key1: 3 2 1 4
Enter the key2: 4 2 3 1
Length of the text:  11
A a r y
a T i w
a r i _
_ _ _ _

a r i _
a T i w
A a r y
_ _ _ _
Cipher Text:  ari_aTiwAary____
Length of the text:  16
a r i _
a T i w
A a r y
_ _ _ _

_ r i a
w T i a
y a r A
_ _ _ _
Cipher Text:  _riawTiayarA____
```

6. **Double transposition cipher (column followed by row) - (Decryption)**
   Code:

```python
def double_transposition_decrypt(cipher_text, key1, key2):
    intermediate_plain = column_transpose_decrypt(cipher_text, key2)
    original_plain = row_transpose_decrypt(intermediate_plain, key1)
    return original_plain
```

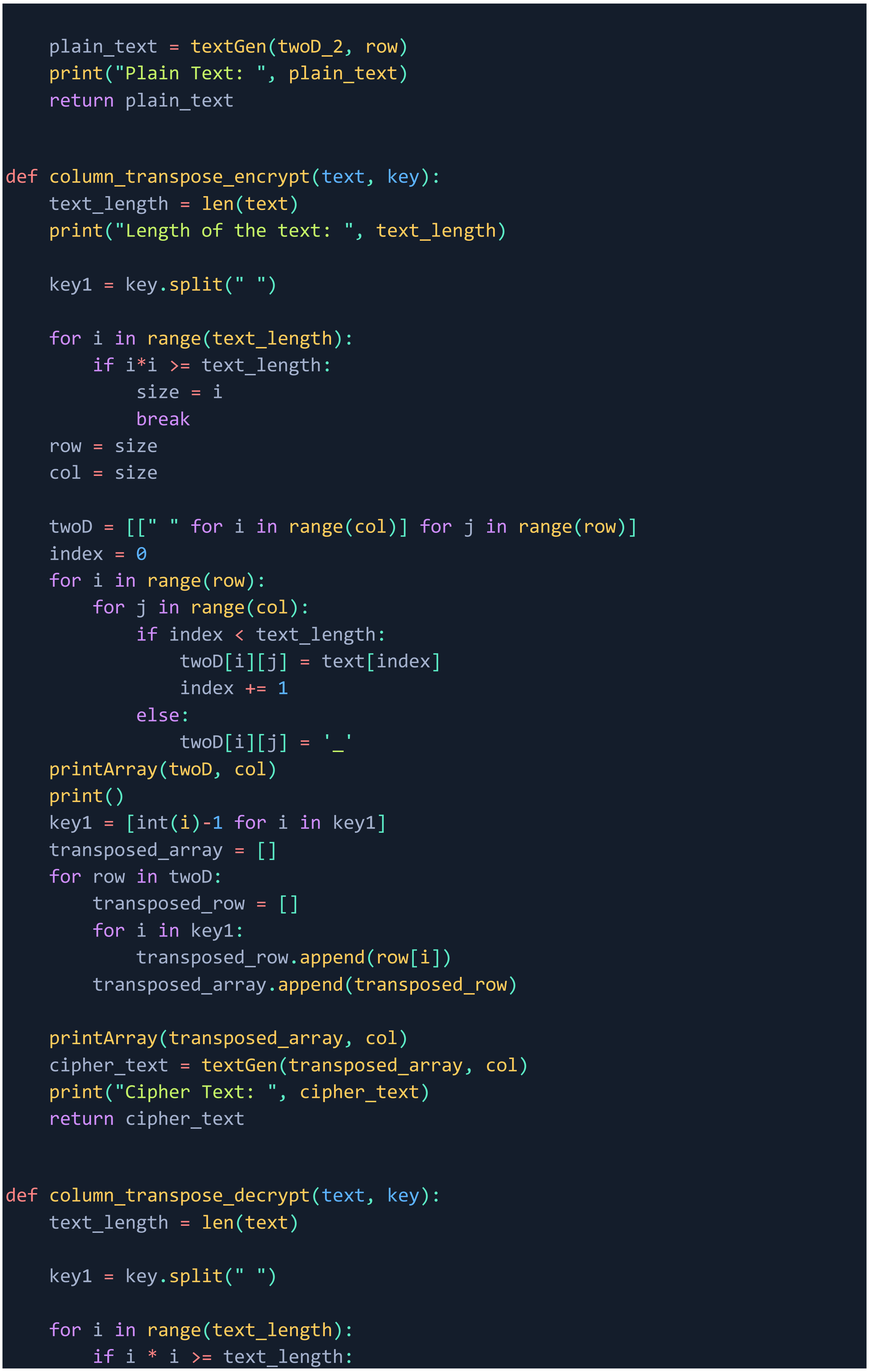
Output:



```
Enter 1 for Row Transpose Encryption
Enter 2 for Row Transpose Decryption
Enter 3 for Column Transpose Encryption
Enter 4 for Column Transpose Decryption
Enter 5 for Double Transposition Encryption
Enter 6 for Double Transposition Decryption
Enter 7 to exit
Enter your choice: 6
Enter the text: _riawTiayarA____
Enter the key1: 3 2 1 4
Enter the key2: 4 2 3 1
_ r i a
w T i a
y a r A
_ _ _ _

a r i _
a T i w
A a r y
_ _ _ _
Plain Text:  ari_aTiwAary____
Length of the text:  16
a r i _
a T i w
A a r y
_ _ _ _

A a r y
a T i w
a r i _
_ _ _ _
Plain Text:  AaryaTiwari_____
```

7. **Final Menu-Driven Program.**
   Code:

```python
def printArray(arr, n):
    for i in range(n):
        for j in range(n):
            print(arr[i][j], end=" ")
        print()


def textGen(arr, n):
    text = ""
    for i in range(n):
        for j in range(n):
            # print(arr[i][j], end=" ")
            text += arr[i][j]
    return text


def row_transpose_encrypt(text, key):
    text_length = len(text)
    print("Length of the text: ", text_length)

    key1 = key.split(" ")

    for i in range(text_length):
        if i*i >= text_length:
            size = i
            break
    row = size
    col = size
    twoD = [[" " for i in range(col)] for j in range(row)]
    index = 0
    for i in range(row):
        for j in range(col):
            if index < text_length:
                twoD[i][j] = text[index]
                index += 1
            else:
                twoD[i][j] = '_'

    printArray(twoD, row)
    print()
    twoD_2 = twoD.copy()

    for i in range(len(key1)):  # [3,1,2]
        twoD_2[i] = twoD[int(key1[i])-1]

    printArray(twoD_2, row)
    cipher_text = textGen(twoD_2, row)
    print("Cipher Text: ", cipher_text)
    return cipher_text


def row_transpose_decrypt(text, key):
    text_length = len(text)
    print("Length of the text: ", text_length)

    key1 = key.split(" ")

    for i in range(text_length):
        if i*i >= text_length:
            size = i
            break
    row = size
    col = size
    twoD = [[" " for i in range(col)] for j in range(row)]
    index = 0
    for i in range(row):
        for j in range(col):
            if index < text_length:
                twoD[i][j] = text[index]
                index += 1
            else:
                twoD[i][j] = '_'
    printArray(twoD, row)
    print()
    twoD_2 = twoD.copy()

    for i in range(len(key1)):
        twoD_2[int(key1[i])-1] = twoD[i]

    printArray(twoD_2, row)
```



```
    plain_text = textGen(twoD_2, row)
    print("Plain Text: ", plain_text)
    return plain_text


def column_transpose_encrypt(text, key):
    text_length = len(text)
    print("Length of the text: ", text_length)

    key1 = key.split(" ")

    for i in range(text_length):
        if i*i >= text_length:
            size = i
            break
    row = size
    col = size

    twoD = [[" " for i in range(col)] for j in range(row)]
    index = 0
    for i in range(row):
        for j in range(col):
            if index < text_length:
                twoD[i][j] = text[index]
                index += 1
            else:
                twoD[i][j] = '_'
    printArray(twoD, col)
    print()
    key1 = [int(i)-1 for i in key1]
    transposed_array = []
    for row in twoD:
        transposed_row = []
        for i in key1:
            transposed_row.append(row[i])
        transposed_array.append(transposed_row)

    printArray(transposed_array, col)
    cipher_text = textGen(transposed_array, col)
    print("Cipher Text: ", cipher_text)
    return cipher_text


def column_transpose_decrypt(text, key):
    text_length = len(text)

    key1 = key.split(" ")

    for i in range(text_length):
        if i * i >= text_length:
            size = i
            break
    row = size
    col = size

    twoD = [[" " for i in range(col)] for j in range(row)]
    index = 0
    for i in range(row):
        for j in range(col):
            if index < text_length:
                twoD[i][j] = text[index]
                index += 1
            else:
                twoD[i][j] = '_'
    printArray(twoD, col)
    print()
    key1 = [int(i) - 1 for i in key1]
    original_array = []
    for i in range(row):
        original_row = []
        for j in range(col):
            original_row.append(twoD[i][key1[j]])
        original_array.append(original_row)
    printArray(original_array, col)

    plain_text = ""
    for row in original_array:
        for char in row:
            plain_text += char
    print("Plain Text: ", plain_text)
    return plain_text


def double_transposition_encrypt(text, key1, key2):
    intermediate_cipher = row_transpose_encrypt(text, key1)
    final_cipher = column_transpose_encrypt(intermediate_cipher, key2)
    return final_cipher


def double_transposition_decrypt(cipher_text, key1, key2):
    intermediate_plain = column_transpose_decrypt(cipher_text, key2)
    original_plain = row_transpose_decrypt(intermediate_plain, key1)
    return original_plain


x = True
while x:
    print("Enter 1 for Row Transpose Encryption")
    print("Enter 2 for Row Transpose Decryption")
    print("Enter 3 for Column Transpose Encryption")
    print("Enter 4 for Column Transpose Decryption")
    print("Enter 5 for Double Transposition Encryption")
    print("Enter 6 for Double Transposition Decryption")
    print("Enter 7 to exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        user_text = input("Enter the text: ")
        user_key = input("Enter the key: ")
        cipher_text = row_transpose_encrypt(user_text, user_key)
    elif choice == 2:
        user_text = input("Enter the text: ")
        user_key = input("Enter the key: ")
        plain_text = row_transpose_decrypt(user_text, user_key)
    elif choice == 3:
        user_text = input("Enter the text: ")
        user_key = input("Enter the key: ")
        cipher_text = column_transpose_encrypt(user_text, user_key)
    elif choice == 4:
        user_text = input("Enter the text: ")
        user_key = input("Enter the key: ")
        plain_text = column_transpose_decrypt(user_text, user_key)
    elif choice == 5:
        user_text = input("Enter the text: ")
        user_key1 = input("Enter the key1: ")
        user_key2 = input("Enter the key2: ")
        cipher_text = double_transposition_encrypt(user_text, user_key1,
user_key2)
    elif choice == 6:
        user_text = input("Enter the text: ")
        user_key1 = input("Enter the key1: ")
        user_key2 = input("Enter the key2: ")
        plain_text = double_transposition_decrypt(user_text, user_key1,
user_key2)
    elif choice == 7:
        x = False
    else:
        print("Invalid Choice. Please choose a correct input!!")
```

Questions:

1) Compare substitution ciphers and transposition/permutation ciphers.

   Ans:

   Substitution Ciphers:

   - Replace plaintext characters with other characters.
   - Caesar, monoalphabetic, polyalphabetic, and homophonic are examples.
   - Vulnerable to frequency analysis and known-plaintext attacks.
   - Simpler to implement.
   - Lower level of security compared to modern encryption.

   Transposition/Permutation Ciphers:

- Rearrange characters without changing their values.
- Columnar, rail fence, route, and double transposition are examples.
- Not as vulnerable to frequency analysis but can still be susceptible to pattern recognition.
- More complex to implement, especially for multi-dimensional arrays or routes.
- Moderate level of security, often used in combination with other techniques.

2) Define confusion and diffusion properties. Comment on of both substitution and transposition ciphers w.r.t. confusion and diffusion properties.
Ans:

Confusion:
Confusion refers to the idea of making the relationship between the plaintext and the ciphertext as complex and intricate as possible. In other words, it ensures that there is no simple or easily recognizable pattern between the input and output. Confusion aims to hide the statistical structure of the plaintext by introducing randomness and unpredictability in the transformation process.

Diffusion:
Diffusion involves spreading the influence of individual plaintext elements throughout the entire ciphertext. This ensures that small changes in the plaintext result in extensive changes in the ciphertext. Diffusion helps in preventing local patterns or correlations from persisting across the encryption process.

Substitution Ciphers:

Confusion: Substitution ciphers like the Vigenère cipher and modern block ciphers achieve confusion by introducing non-linear mappings between plaintext and ciphertext. Each plaintext character is replaced by a different ciphertext character based on a substitution key, making the relationship complex and difficult to deduce.
Diffusion: Substitution ciphers alone might not provide strong diffusion, as each character is replaced independently. This is why substitution ciphers are often combined with transposition techniques to achieve better diffusion. Without proper diffusion, repeated patterns in the plaintext could lead to recognizable patterns in the ciphertext.

Transposition Ciphers:

Confusion: Transposition ciphers provide confusion by rearranging the order of the plaintext characters. This can hide patterns and relationships in the original order.
Diffusion: Transposition ciphers excel at providing diffusion because they spread the influence of individual plaintext elements throughout the ciphertext. A small change in the input will lead to substantial changes in the output due to the reshuffling of characters.

Conclusion:
We can conclude that we have learnt about transposition ciphers (Row,Column and Double)

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References: Books/ Journals/ Websites:

1. Behrouz A. Forouzan, "Cryptography and Network Security", Tata McGraw Hill
2. Mark Stamp, "Information Security Principles and Practice", Wiley.
3. William Stalling, "Cryptography and Network Security", Prentice Hall