Cryptography and Network Security Lab

Assignment 5.2  
Implementation and Understanding of Columnar Transposition Cipher

2019BTECS00058  
Devang K

Batch: B2

Title: Implementation and Understanding of Columnar Transposition Cipher

Aim: To Study, Implement and Demonstrate the Columnar Transposition Cipher Algorithm

Theory:

Columnar Transposition Cipher is a form of transpositional cipher which involves us to write the plaintext in rows and then read the ciphertext as the columns.

To encrypt in Columnar Cipher, we need the plaintext and the key. Firstly, we pad the plaintext so that it has elements such that their count is divisible by the key-size. This is performed so that all the columns are filled. Further, we give an order index to each key-character based on its ASCII value. Then, in a matrix of length(key) columns and (length(plaintext) / length(key)) rows. And we fill the matrix going row-wise with the plaintext. Then, based on the order ID of the keys, we fill the ciphertext by appending characters in columns in the order of the IDs.

To decrypt in Columnar Cipher, we first make the matrix of length(key) columns and (length(ciphertext) / length(key)) rows. Then we give an order index to each key-character based on its ASCII value. And proceed to fill our matrix column-wise in that order ID with the ciphertext characters. The generated matrix is then traversed row-wise to obtain the initial plaintext.

Illustration:

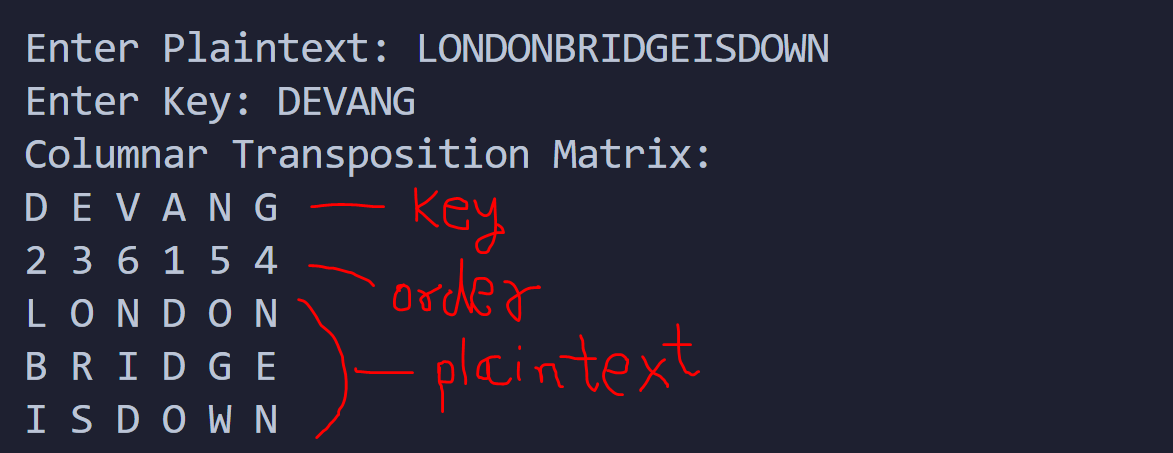
Let’s illustrate using the plaintext of ‘LONDONBRIDGEISDOWN’ and a key of ‘DEVANG’

Encryption

We would first pad the plaintext as needed. length(key) = 6 and length(plaintext) = 18. Since 18%6 = 0, no padding would be needed for this illustration. Then we give an order ID corresponding to the alphabetic precedence.

D E V A N G  
2 3 6 1 5 4

Now, we form the matrix by making a traversal row-wise.



Then, we traverse column-wise with the order ID to append to make it the cipher-text.  
Thus, the cipher text would be: ‘DDO’ + ‘LBI’ + ‘ORS’ + ‘NEN’ + ‘OGW’ + ‘NID’.  
Which is: ‘DDOLBIORSNENOGWNID’ – our ciphertext.

Decryption

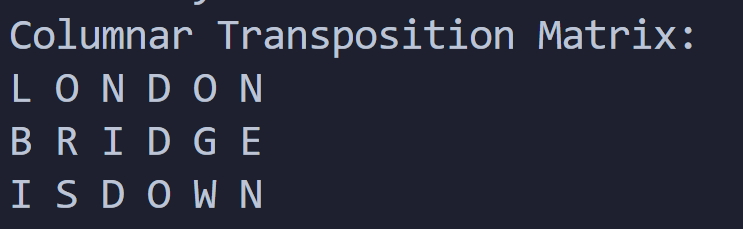
To decrypt, we first determine the key and the corresponding order ID. We then divide the encrypted text into equal sizes of length(encText) / length(key). Then, we add them to the columns of the matrix in the order of their Order IDs.

Say, we wish to decrypt ‘DDOLBIORSNENOGWNID’ with key ‘DEVANG’,

We first divide in strings of size 3 – 18/6.

We get, ‘DDO’, ‘LBI’, ‘ORS’, ‘NEN’, ‘OGW’, ‘NID’.

We arrange them in columns with respect to the key order ID.

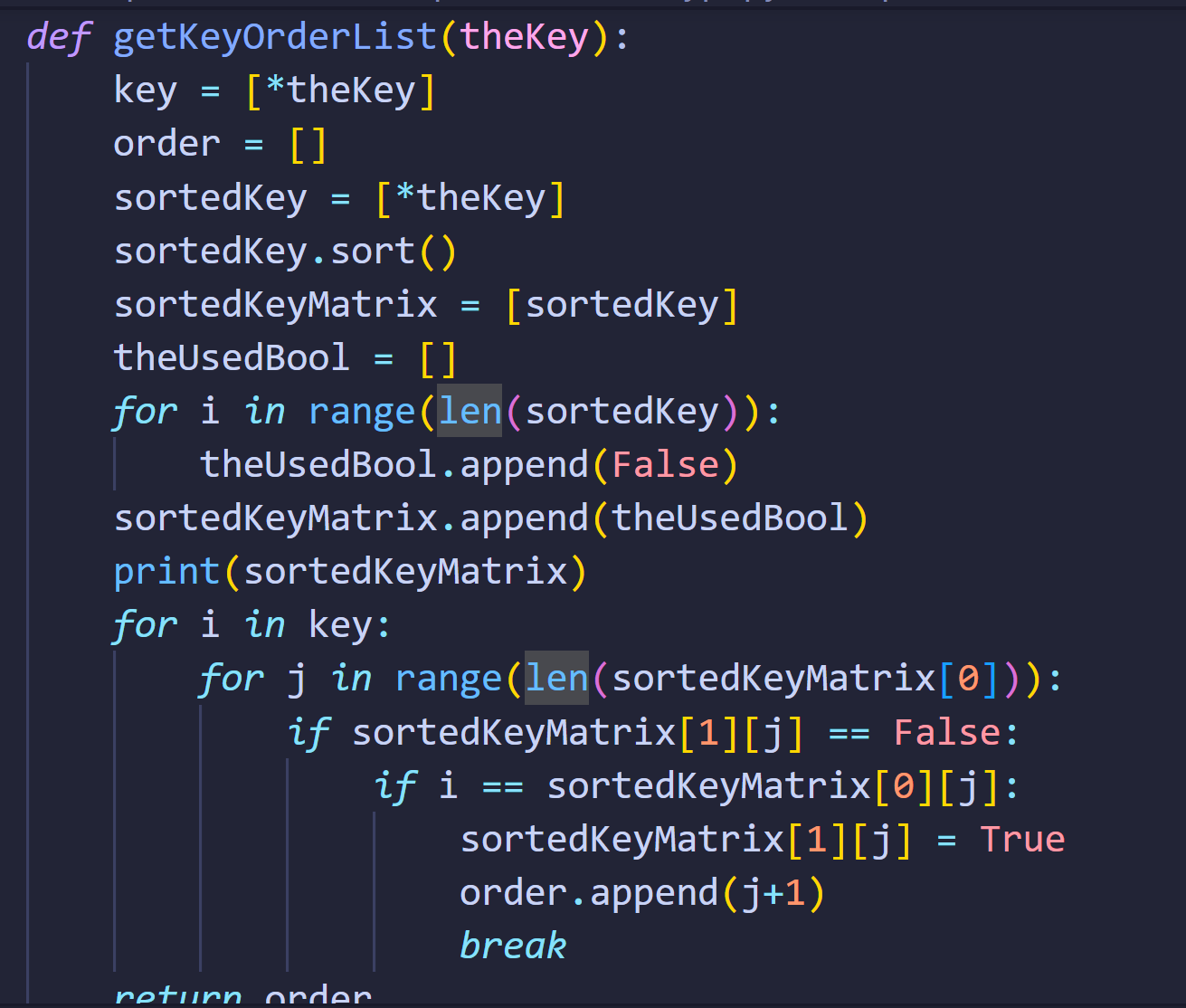


Then, we read the plaintext row-wise from the matrix.  
Thus, our plaintext is ‘LONDONBRIDGEISDOWN’.

We shall see more examples through the code.

Code:

The crux of the application is in the function that computes the Order ID of the key.



Elsewhere, the code to write in the column and read from it is straightforward.

Code for Encryption: *import* math

*def* getOrderFromOrderList(orderList):

    theOrderList = []

*for* i *in* range(len(orderList)):

        theOrderList.append(0)

    count = 0

*for* i *in* range(len(orderList)):

        theOrderList[orderList[i]-1] = count

        count += 1

*return* theOrderList

*def* generateEncryptedTextFromColumnarTranspositionMatrix(theMatrix):

    encrpytedText = ""

    theOrderList = getOrderFromOrderList(theMatrix[1])

*for* i *in* theOrderList:

*for* j *in* range(len(theMatrix)-2):

            encrpytedText += theMatrix[j+2][i]

*return* encrpytedText

*def* printMatrix(fenceMatrix):

*for* i *in* range(len(fenceMatrix)):

*for* j *in* range(len(fenceMatrix[0])):

            print(fenceMatrix[i][j], end=' ')

        print()

*def* getKeyOrderList(theKey):

    key = [\*theKey]

    order = []

    sortedKey = [\*theKey]

    sortedKey.sort()

    sortedKeyMatrix = [sortedKey]

    theUsedBool = []

*for* i *in* range(len(sortedKey)):

        theUsedBool.append(False)

    sortedKeyMatrix.append(theUsedBool)

    print(sortedKeyMatrix)

*for* i *in* key:

*for* j *in* range(len(sortedKeyMatrix[0])):

*if* sortedKeyMatrix[1][j] == False:

*if* i == sortedKeyMatrix[0][j]:

                    sortedKeyMatrix[1][j] = True

                    order.append(j+1)

*break*

*return* order

*def* plaintextToMatrixOfKey(plaintext, keySize):

    keyMatrix = []

    numberOfCharsToHave = math.ceil(len(plaintext)/keySize) \* keySize

    numberOfCharsToAdd = numberOfCharsToHave - len(plaintext)

*for* i *in* range(numberOfCharsToAdd):

        plaintext += 'X'

    theMatrixRow = []

    counter = 0

*for* i *in* range(len(plaintext)):

        theMatrixRow.append(plaintext[i])

        counter += 1

*if* counter == keySize:

            keyMatrix.append(theMatrixRow)

            counter = 0

            theMatrixRow = []

*return* keyMatrix

print("Columnar Transposition Cipher\n")

print("Enter Plaintext: ", end='')

plaintext = input()

print("Enter Key: ", end='')

key = input()

columnarTranspositionMatrix = []

columnarTranspositionMatrix.append([\*key])

columnarTranspositionMatrix.append(getKeyOrderList(key))

*# print(columnarTranspositionMatrix)*

columnarTranspositionMatrix += plaintextToMatrixOfKey(plaintext, len(key))

print("Columnar Transposition Matrix:")

printMatrix(columnarTranspositionMatrix)

encryptedText = generateEncryptedTextFromColumnarTranspositionMatrix(columnarTranspositionMatrix)

print("Plaintext:", plaintext)

print("Encrypted Text:", encryptedText)

Code for Decryption:

*import* math

*def* printMatrix(fenceMatrix):

*for* i *in* range(len(fenceMatrix)):

*for* j *in* range(len(fenceMatrix[0])):

            print(fenceMatrix[i][j], end=' ')

        print()

*def* matrixTranspose(matrix):

    rowSize = len(matrix)

    colSize = len(matrix[0])

    transposedMatrix = []

*for* i *in* range(colSize):

        theRow = []

*for* j *in* range(rowSize):

            theRow.append(0)

        transposedMatrix.append(theRow)

*for* i *in* range(len(matrix)):

*for* j *in* range(len(matrix[0])):

            transposedMatrix[j][i] = matrix[i][j]

*return* transposedMatrix

*def* getKeyOrderList(theKey):

    key = [\*theKey]

    order = []

    sortedKey = [\*theKey]

    sortedKey.sort()

    sortedKeyMatrix = [sortedKey]

    theUsedBool = []

*for* i *in* range(len(sortedKey)):

        theUsedBool.append(False)

    sortedKeyMatrix.append(theUsedBool)

    print(sortedKeyMatrix)

*for* i *in* key:

*for* j *in* range(len(sortedKeyMatrix[0])):

*if* sortedKeyMatrix[1][j] == False:

*if* i == sortedKeyMatrix[0][j]:

                    sortedKeyMatrix[1][j] = True

                    order.append(j+1)

*break*

*return* order

*def* getMatrixFromEncryptedText(encText, keySize, keyOrderList):

    divisionSize = len(encText) // keySize

    theDividedStringList = [encText[start:start+divisionSize] *for* start *in* range(0, len(encText), divisionSize)]

    print(theDividedStringList)

    theColumnarMatrix = []

*for* i *in* theDividedStringList:

        theColumnarMatrix.append([\*i])

    theColumnarMatrixInOrder = []

*for* i *in* keyOrderList:

        theColumnarMatrixInOrder.append(theColumnarMatrix[i-1])

    print(theColumnarMatrixInOrder)

*# Transpose the matrix*

    theColumnarMatrixInOrder = matrixTranspose(theColumnarMatrixInOrder)

*return* theColumnarMatrixInOrder

*def* columnarMatrixToPlaintextString(colMatrix):

    thePlaintext = ""

*for* i *in* colMatrix:

        thePlaintext += "".join(i)

*return* thePlaintext

*def* listIntToString(lst):

    theLst = []

*for* i *in* lst:

        theLst.append(str(i))

*return* theLst

print("Columnar Transposition Cipher\n")

print("Enter Encrypted Text: ", end='')

encryptedText = input()

print("Enter Key: ", end='')

key = input()

*# We start by making the columnTranspositionMatrix*

keyOrderList = getKeyOrderList(key)

print("Key Order List:")

print(" ".join([\*key]))

print(" ".join(listIntToString(keyOrderList)))

print()

columnarMatrixInOrder = getMatrixFromEncryptedText(encryptedText, len(key), keyOrderList)

print("Columnar Transposition Matrix:")

printMatrix(columnarMatrixInOrder)

plaintext = columnarMatrixToPlaintextString(columnarMatrixInOrder)

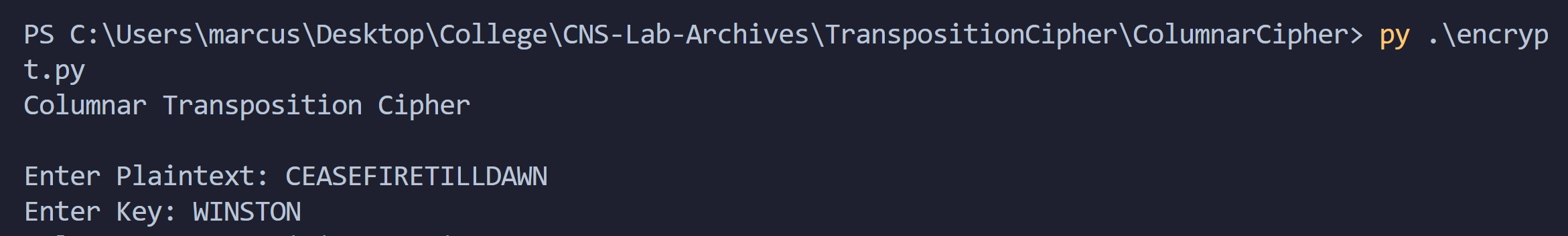
print("Encrypted Text:", encryptedText)

print("Plaintext:", plaintext)

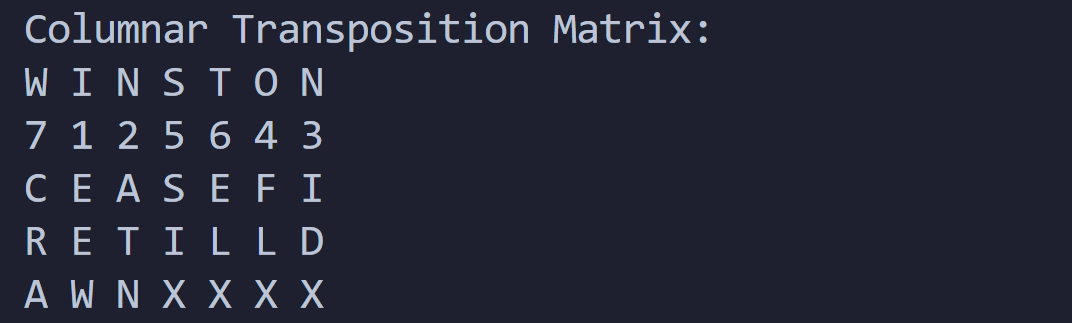
We now solve some examples with the code.

Say we wish to encrypt: ‘CEASEFIRETILLDAWN’  
and we take our key as ‘WINSTON’

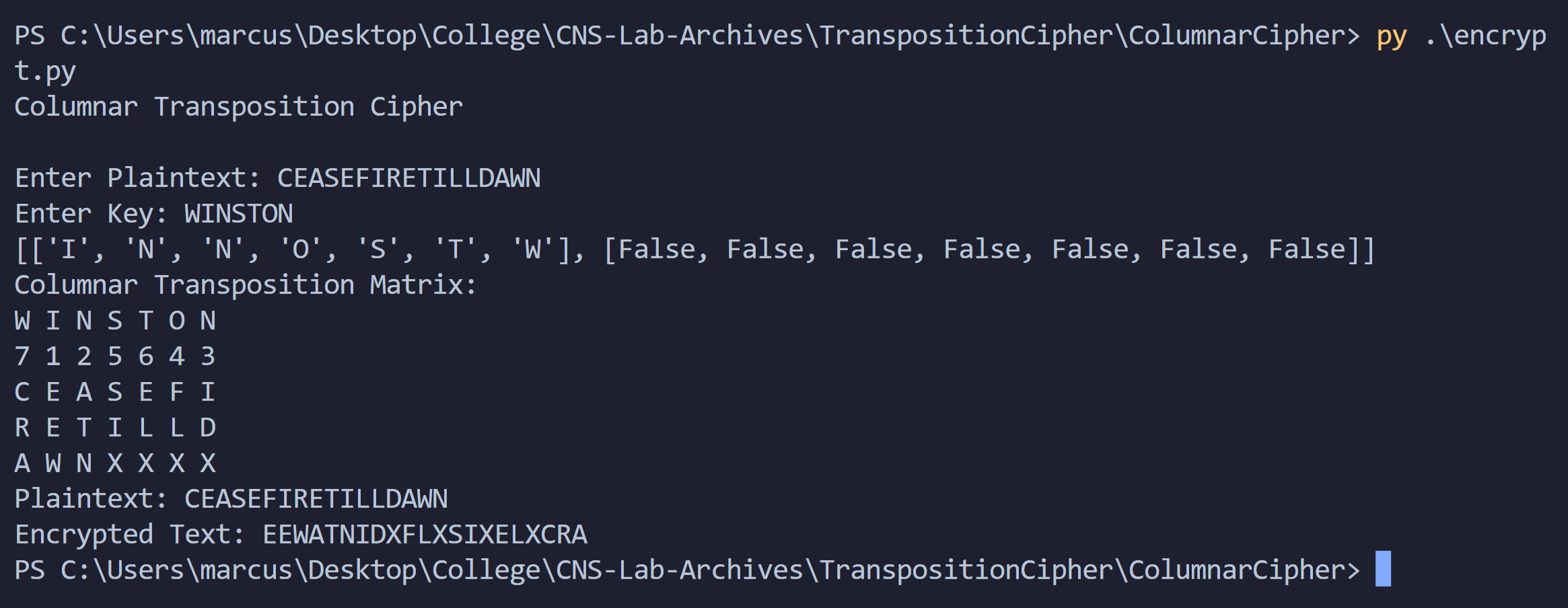
To perform encryption:



The Generated Columnar Matrix:



Therefore, the ciphertext would be:

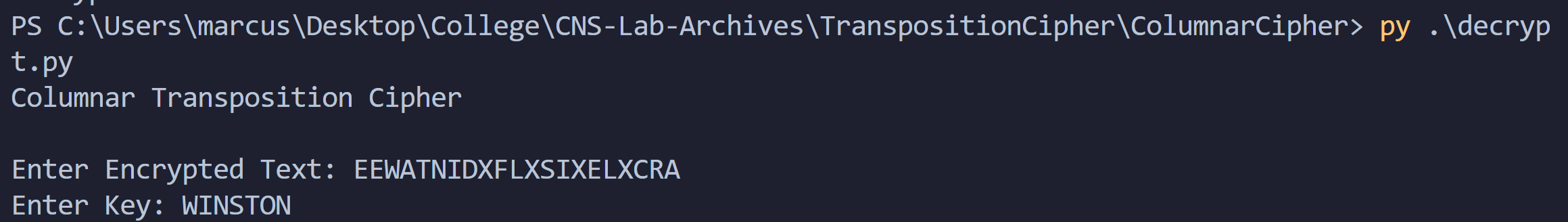


Ciphertext when plaintext – ‘CEASEFIRETILLDAWN’ and key – ‘WINSTON’ is ‘EEWATNIDXFLXSIXELXCRA’.

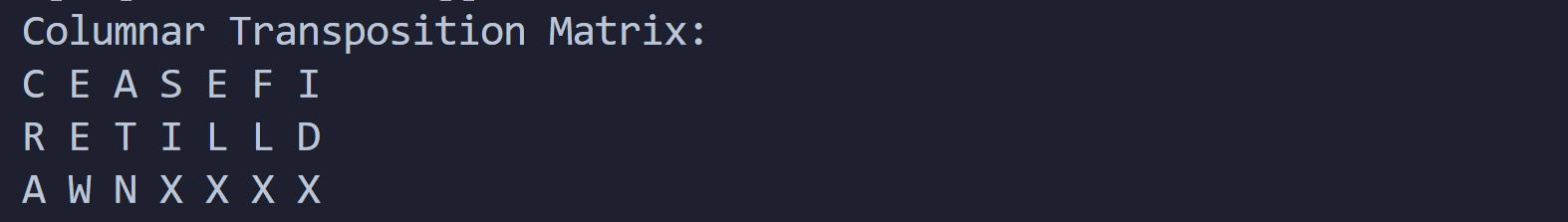
To decrypt:

EncText – ‘EEWATNIDXFLXSIXELXCRA’ and key – ‘WINSTON’

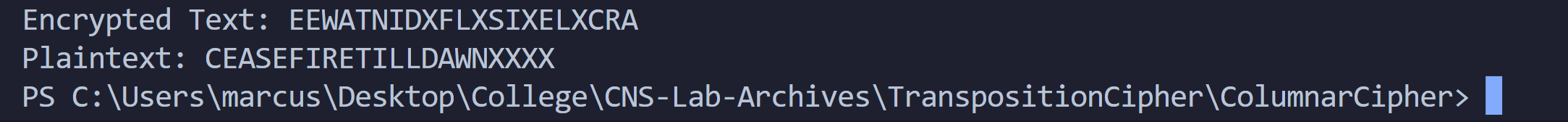
We first get the key order ID



Then, we make the columnar matrix with respect to the order ID.



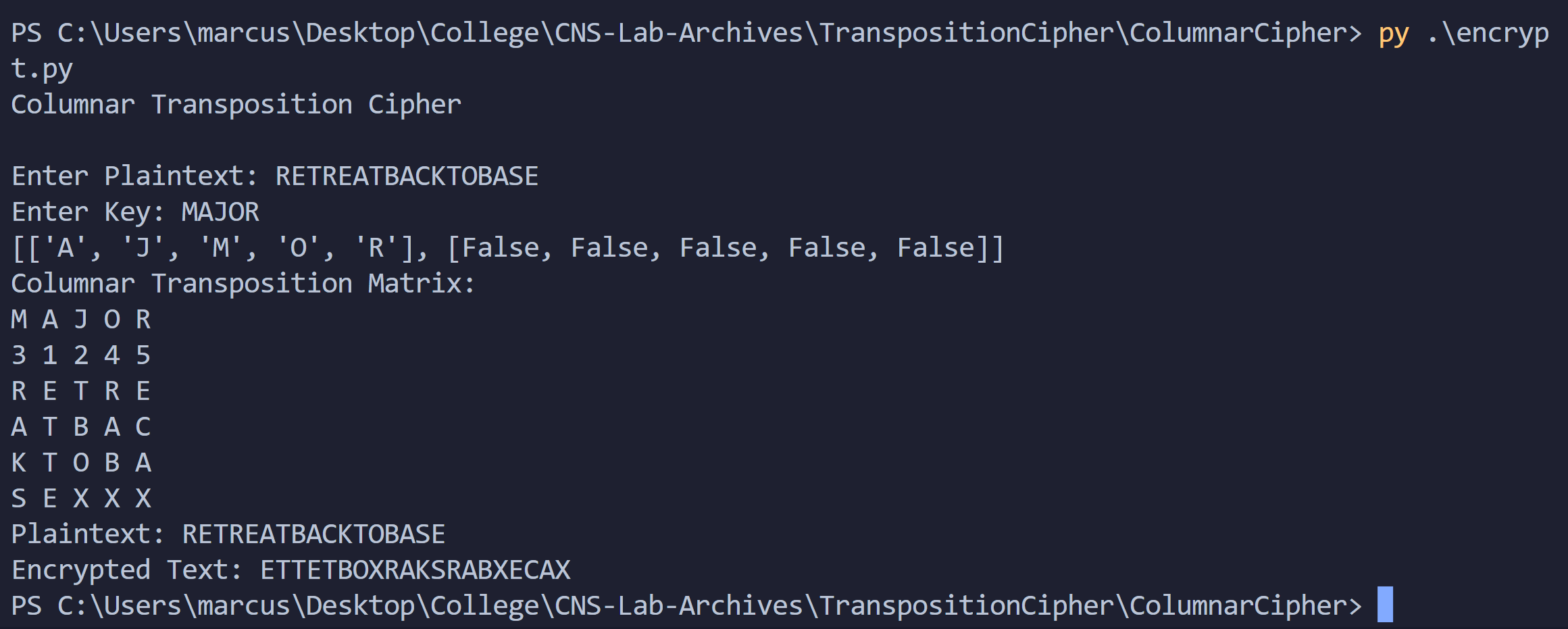
We finally get;



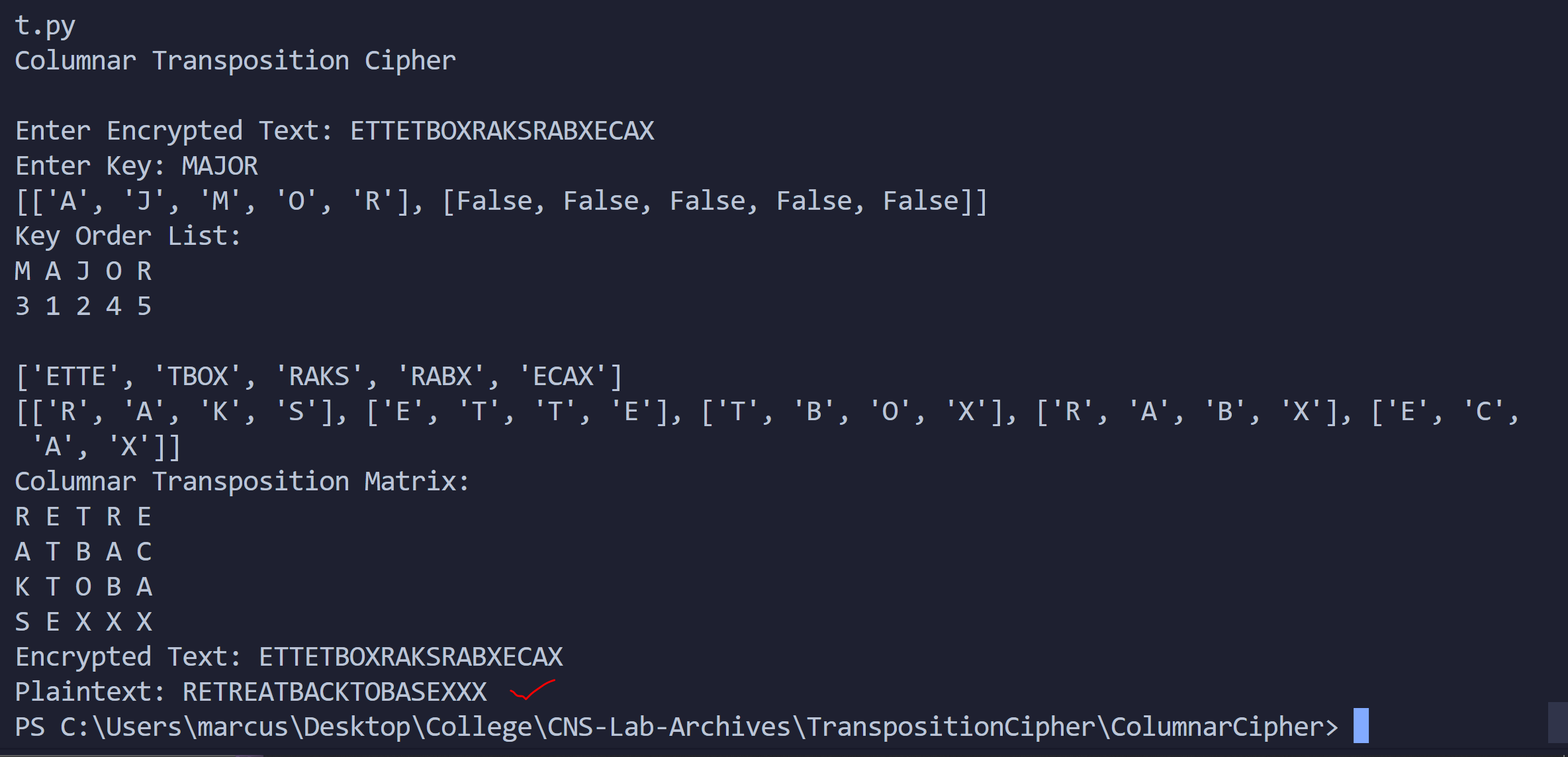
Therefore, we get our plaintext back.

Let’s look at another example:  
Say we wish to encrypt ‘RETREATBACKTOBASE’ with key ‘MAJOR’:

Encryption:



Ciphertext – ‘ETTETBOXRAKSRABXECAX’

Decryption:  


We get back our plaintext.

Thus, we demonstrated the working of the code with examples.

Conclusion:

Thus, the Columnar Transposition Cipher algorithm was studied and demonstrated with the code.