Cryptography and Network Security Lab

Assignment 1
Implementation and Understanding of Caesar Cipher

2019BTECS00058
Devang K

Batch: B2

Title: Implementation and Understanding of Caesar Cipher

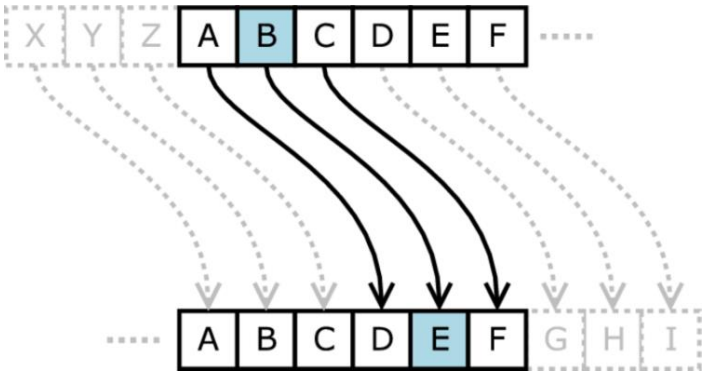
Aim: To Study, Implement and Demonstrate the Caesar Cipher Algorithm

Theory:

Caesar Cipher is named after ‘Julius Caesar’ who used to use this cipher algorithm in his private correspondence. It is a form of Substitution Cipher wherein we replace each letter in the plaintext by a letter some fixed number of positions ahead/behind. Let us called these number of positions as ‘shift key’.

Illustration:

Say we wish to encrypt ‘ABC’ with a shift key of 4. Then, the cipher would work as:



Taking some examples:
Say ‘DEVANG’ with a shift-key 3
Our logic would be like:

For every character:
ASCII of D = 68

To make a general case, we subtract from 65 – ASCII of A.
We get, 68-65 = 3 (0-based index)

We add 3 (shift-key) to it, 3+3 = 3

Then, we take a mod with 26 (all the alphabets, it’s a circular repetition)

6 % 26 = 6

Then add, 65 again and take ASCII
We get 71, which converting to Character is – G.

Similar to this, we go on doing for each character. Thus, the Ciphertext for ‘DEVANG’ would be ‘GHYDQJ’.

To decrypt this, we run the algorithm in reverse order:

So, for every character, we know the shift-key is 3
Say ‘G’, ASCII – 71

We subtract 65 and also the shift-key value:
71 – 65 – 3 = 3

Then taking a mod with 26 for general case:
3 % 26 = 3

We then add 65 and convert it to character.
3 + 65 = 68
which is ‘D’

Similarly, repeating for every character, we finally get – ‘DEVANG’.

Thus, we can encrypt and decrypt in Caesar Cipher.
We shall look at more examples in the implementation.

Code:

The main crux of the code is the function that handles this shift-key for each character and returns its shifted value.

```
def shiftCharWithKeyDecrypt(c, step):
    amt = ord(c)
    amt -= 65
    amt -= step
    amt %= 26
    amt += 65
    return chr(amt)
```

This function can we used for both, encryption and decryption, just by reversing the sign of key in decryption.
Beyond this, we write the code to take input and traverse character-by-character and encrypt and decrypt.

Code for Encryption:

```
def shiftCharWithKey(c, step):
    amt = ord(c)
    amt -= 65
    amt += step
    amt %= 26
    amt += 65
    return chr(amt)

def chooseTypeOfInput():
    print("Choose your method:")
    print("1. Encrypt a File.")
    print("2. Encrypt an Input.")
    print("Please Enter Type of Input: ", end='')
```

```
    n = int(input())
    if n == 1 or n == 2:
        return n
    print("Incorrect choice. Try Again!")
    chooseTypeOfInput()

choice = chooseTypeOfInput()

value=0
shift = 0

if choice == 1:
    f = open("enc.txt", "r")
    value = f.readline()

else:
    print("Enter the Plaintext: ", end='')
    value = input()

print("Enter the Shift Key Value: ", end='')
shift = int(input())

encryptedValue = ""

for c in value:
    encryptedValue += shiftCharWithKey(c, shift)

print("\nEncryption Result:")
print("PlainText: "+value)
print("Ciphertext: "+encryptedValue)
```

Code for Decryption:

```
def shiftCharWithKeyDecrypt(c, step):
    amt = ord(c)
    amt -= 65
    amt -= step
    amt %= 26
    amt += 65
    return chr(amt)

def chooseTypeOfInput():
    print("Choose your method:")
    print("1. Decrypt a File.")
    print("2. Decrypt an Input.")
    print("Please Enter Type of Input: ", end='')
    n = int(input())
    if n == 1 or n == 2:
```

```
        return n
    print("Incorrect choice. Try Again!")
    chooseTypeOfInput()

choice = chooseTypeOfInput()

ciphertext=0
shift = 0

if choice == 1:
    f = open("dec.txt", "r")
    ciphertext = f.readline()

else:
    print("Enter the Plaintext: ", end='')
    ciphertext = input()

print("Enter the Shift Key Value: ", end='')
shift = int(input())

plaintext = ""

for i in ciphertext:
    plaintext += shiftCharWithKeyDecrypt(i, shift)

print("Ciphertext: "+ciphertext)
print("Decrypted Plaintext: "+plaintext)
```

We now solve some examples with the code.
Say we wish to encrypt: ‘CEASEFIRETILLDAWN’

We choose the option to directly type the string.
And let’s take shift-key of 12.

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\CaesarCipher> py .\encryption.py
Choose your method:
1. Encrypt a File.
2. Encrypt an Input.
Please Enter Type of Input: 2
Enter the Plaintext:
```

We enter the plaintext and shift-key

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\CaesarCipher> py .\encryption.py
Choose your method:
1. Encrypt a File.
2. Encrypt an Input.
Please Enter Type of Input: 2
Enter the Plaintext: CEASEFIRETILLDAWN
Enter the Shift Key Value: 12

Encryption Result:
PlainText: CEASEFIRETILLDAWN
Ciphertext: OQMEQRUDQFUXXPMIZ
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\CaesarCipher>
```

We get that encryption of ‘CEASEFIRETILLDAWN’ with shift-key of 12 is: ‘OQMEQRUDQFUXXPMIZ’.

Now, let’s decrypt it using our decryption code.

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\CaesarCipher> py .\decryption.py
Choose your method:
1. Encrypt a File.
2. Encrypt an Input.
Please Enter Type of Input: 2
```

We choose 2 and then enter the Ciphertext and shift-key.

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\CaesarCipher> py .\decryption.py
Choose your method:
1. Encrypt a File.
2. Encrypt an Input.
Please Enter Type of Input: 2
Enter the Plaintext: OQMEQRUDQFUXXPMIZ
Enter the Shift Key Value: 12
Ciphertext: OQMEQRUDQFUXXPMIZ
Decrypted Plaintext: CEASEFIRETILLDAWN
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\CaesarCipher>
```

Thus, we get the result, ‘CEASEFIRETILLDAWN’.

Taking another example, we can encrypt and decrypt the content of a file.
We take enc.txt and dec.txt

Putting the example ‘ATTACKWITHALLWEHAVE’ with a shift-key of -10.

We now run our code

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\CaesarCipher> py .\encryption.py
ption.py
Choose your method:
1. Encrypt a File.
2. Encrypt an Input.
Please Enter Type of Input: 1
Enter the Shift Key Value: -10

Encryption Result:
PlainText: ATTACKWITHALLWEHAVE
Ciphertext: QJJQSAMYJXQBBMUXQLU
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\CaesarCipher>
```

Ciphertext: ‘QJJQSAMYJXQBBMUXQLU’

Then, we decrypt.

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\CaesarCipher> py .\decryption.py
Choose your method:
1. Decrypt a File.
2. Decrypt an Input.
Please Enter Type of Input: 1
Enter the Shift Key Value: -10
Ciphertext: QJJQSAMYJXQBBMUXQLU
Decrypted Plaintext: ATTACKWITHALLWEHAVE
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\CaesarCipher>
```

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

Conclusion:

Thus, the Caesar Cipher algorithm was studied and demonstrated with the code. It is observed that Caesar Cipher is really weak as the key can be any of 26 values and therefore is easy to crack in modern cryptography.

Cryptography and Network Security Lab

Assignment 2
Performing Cryptanalysis on Caesar Cipher

2019BTECS00058
Devang K

Batch: B2

Title: Performing Cryptanalysis on Caesar Cipher.

Aim: To perform cryptanalysis on Caesar Cipher and guess the shift-key value from an encryption.

Theory:

Caesar Cipher is named after 'Julius Caesar' who used to use this cipher algorithm in his private correspondence. It is a form of Substitution Cipher wherein we replace each letter in the plaintext by a letter some fixed number of positions ahead/behind. Let us called these number of positions as 'shift key'.

We studied the Caesar Cipher in Assignment 1 and implemented the logic to encrypt and decrypt it. The shift-key of the Caesar Cipher is cyclic. Meaning, a shift-key of 0 and a Shift-Key of 26 would give the same result. Therefore, the Cipher can have only 26 alternatives – one of which would be the right Shift-Key.

We exploit this vulnerability of the Cipher algorithm by performing a dictionary attack. For a given word of Ciphertext, we make a list of all the possible words and then check if they exist in our dictionary, the word among the list which does exist, would give us the corresponding key.

Code:

Python has a library called ‘enchant’ which has a large dictionary of words and has a function to check whether a word exists in that dictionary. We shall make use of the same.

For the given Ciphertext word, we generate all the combinations for all shift-keys and then one-by-one compare if the generated word exists in the enchant dictionary.

Here is the code implementation of breaking the Caesar Cipher.

```
import enchant

d = enchant.Dict("en_US")

def shiftCharWithKeyDecrypt(c, step):
    amt = ord(c)
    amt -= 65
    amt -= step
    amt %= 26
    amt += 65
    return chr(amt)

print("Enter the Cipherword: ",end='')
cipherword = input()

# Get all shift combos to guess the shift amt
# From 0 to 26
possibleShiftedCombos = []
for shift in range(26):
    theString = ""
    for i in cipherword:
        theString += shiftCharWithKeyDecrypt(i, shift)
    possibleShiftedCombos.append(theString)

shiftValue = 0

for i in range(len(possibleShiftedCombos)):
    if d.check(possibleShiftedCombos[i]):
        shiftValue = i
        break

decryptedText = ""
for i in cipherword:
    decryptedText += shiftCharWithKeyDecrypt(i, shiftValue)

print("Possible Values of Plaintext:")
```

```
print(possibleShiftedCombos)
print()
print("Ciphertext: "+cipherword)
print("Key:", shiftValue)
print("Decrypted Plaintext: "+decryptedText)
```

We now solve some examples with the code.
Say we have an encrypted word ‘OVSPKHF’. Let’s put it through our code.

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Cryptanalysis> py .\guessKeyOfCae
sarCipher.py
Enter the Cipherword: OVSPKHF
Possible Values of Plaintext:
['OVSPKHF', 'NUROJGE', 'MTQNIFD', 'LSPMHEC', 'KROLGDB', 'JQNKFCA', 'IPMJEBZ', 'HOLIDAY', 'GNK
HCZX', 'FMJGBYW', 'ELIFAXV', 'DKHEZWU', 'CJGDYVT', 'BIFCXUS', 'AHEBWTR', 'ZGDAVSQ', 'YFCZURP'
, 'XEBYTQO', 'WDAXSPN', 'VCZWROM', 'UBYVQNL', 'TAXUPMK', 'SZWTOLJ', 'RYVSNKI', 'QXURMJH', 'PW
TQLIG']

Ciphertext: OVSPKHF
Key: 7
Decrypted Plaintext: HOLIDAY
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Cryptanalysis>
```

We thus generated 26 possible values and checked to find that word at index 7 is valid – ‘HOLIDAY’. Therefore, the shift-key was 7.

We take another example:
Say we have an encrypted word ‘YKILQPAN’. Let’s put it through our code.

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Cryptanalysis> py .\guessKeyOfCaesarCiphe
r.py
Enter the Cipherword: YKILQPAN
Possible Values of Plaintext:
['YKILQPAN', 'XJHKPOZM', 'WIGJONYL', 'VHFINMXK', 'UGEHMLWJ', 'TFDGLKVI', 'SECFKJUH', 'RDBEJIT
G', 'QCADIHSF', 'PBZCHGRE', 'OAYBGFQD', 'NZXAFEPC', 'MYWZEDOB', 'LXVYDCNA', 'KWUXCBMZ', 'JVTW
BALY', 'IUSVAZKX', 'HTRUZYJW', 'GSQTYXIV', 'FRPSXWHU', 'EQORWVGT', 'DPNQVUFS', 'COMPUTER', 'B
NLOTSDQ', 'AMKNSRCP', 'ZLJMRQBO']

Ciphertext: YKILQPAN
Key: 22
Decrypted Plaintext: COMPUTER
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Cryptanalysis>
```

We thus generated 26 possible values and checked to find that word at index 22 is valid – ‘COMPUTER’. Therefore, the shift-key was 22.

Conclusion:

Thus, we cryptanalyzed the Caesar Cipher and broke the algorithm using dictionary attack. Thus, Caesar Cipher is not secure in modern cryptography. Also, this can only be done on a Cipher word and not on any sentence.

A potential drawback of this approach is also that, we may receive ambiguous answer, if 2 or more words are in the dictionary among the possible combinations.

Cryptography and Network Security Lab

Assignment 3
Implementation and Understanding of Vigenere Cipher

2019BTECS00058
Devang K

Batch: B2

Title: Implementation and Understanding of Vigenere Cipher

Aim: To Study, Implement and Demonstrate the Vigenere Cipher Algorithm

Theory:

Vigenere Cipher is an encryption method that is a form of poly-alphabetic substitution. It uses the Vigenere matrix to perform the encryption and decryption.

A Vigenere matrix is a table where alphabets are written in 26 possible rows that are Caesar Ciphered with increasing the key. This table alongside the key character and the character to encrypt/decrypt, gives us the answer.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Illustration:

Encryption

Say we wish to encrypt – ‘ATTACKATONCE’ and say with a key of ‘DEVANG’. Firstly, we generate a cycle-repeating key of the size of the plaintext. So, length of ‘ATTACKATONCE’ = 12. Therefore, the Cyclic Repeating Key would be ‘DEVANGDEVANG’.

Then, we now go character by character and for each character, we take the row element of the plaintext character and the column element of key character. The corresponding element is our Cipher-text.

Therefore, traversing our plaintext, we get row(‘A’) and col(‘D’) i.e. (0,3) which would be ‘A’. Similarly next, row(‘T’) and col(‘E’) i.e. (19, 4) which is ‘X’. Thus, by the end, we would get the generated Ciphertext of ‘DXOAPQDXJNPK’.

Decryption

Say we wish to decrypt our Ciphertext of ‘DXOAPQDXJNPK’ with the key of ‘DEVANG’. We shall first generate the cycle-repeating key, we get ‘DEVANGDEVANG’. Then we shall now traverse every character of the Ciphertext.

Starting with cipher-character ‘D’ and key-character ‘D’, we take the row of the table corresponding to the key-character and then find the column whose

value is the cipher-character ‘D’. Therefore, we get row(‘D’) – 3 and the column corresponding to ‘D’ in the 3rd row as 0. Hence, the value shall become (3, 0) – ‘A’. Similarly, next is key character ‘E’ and cipher character ‘X’. We look at row of ‘E’ – 4 and column corresponding to ‘X’ in ‘E’ – ‘T’. Thus, we continue to traverse till the end until we generate the plaintext ‘ATTACKATONCE’.

We illustrate more examples using the code.

Code:

Some important functions we need for the code, common for encryption and decryption are:

A function to generate the circular key of given length of plain/cipher-text.

```
def generateCircularKeyOfLength(key, size):
    # number of times to repeat the key
    timesToRepeat = math.ceil(size / len(key))
    theCircularKey = key
    for i in range(timesToRepeat):
        theCircularKey += key
    return theCircularKey[:size]
```

A function to generate Vigenere Matrix.

```
def generateVigenereMatrix():
    theVigenereMatrix = []
    for i in range(26):
        theMatrixRow = []
        for j in range(26):
            theMatrixRow.append(chr(((i+j)%26)+65))
        theVigenereMatrix.append(theMatrixRow)
    return theVigenereMatrix
```

Then corresponding to these, we would have the functions to traverse each character with key and compare with the Vigenere Table and generate the Ciphertext or to generate the Plaintext.

Code for Encryption:

```
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 generateVigenereMatrix():
    theVigenereMatrix = []
    for i in range(26):
        theMatrixRow = []
        for j in range(26):
            theMatrixRow.append(chr(((i+j)%26)+65))
        theVigenereMatrix.append(theMatrixRow)
    return theVigenereMatrix

def generateCircularKeyOfLength(key, size):
    # number of times to repeat the key
    timesToRepeat = math.ceil(size / len(key))
    theCircularKey = key
    for i in range(timesToRepeat):
        theCircularKey += key
    return theCircularKey[:size]

vigenereMatrix = generateVigenereMatrix()

print("Vigenere Cipher")
print("Enter Plaintext: ", end='')
plaintext = input()
print("Enter the Key: ", end='')
key = input()

circularKey = generateCircularKeyOfLength(key, len(plaintext))

# To encrypt -> Row Index -> Plaintext, Col Index -> CipherText

encryptedText = ""

for i in range(len(plaintext)):
    rowIndex = ord(plaintext[i])-65
```

```
    colIndex = ord(circularKey[i])-65
    encryptedText += vigenereMatrix[rowIndex][colIndex]

print("\nCircular Key: ", end='')
print(circularKey)

print("\nVigenere Table:")
printMatrix(vigenereMatrix)

print()
print()

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 generateVigenereMatrix():
    theVigenereMatrix = []
    for i in range(26):
        theMatrixRow = []
        for j in range(26):
            theMatrixRow.append(chr(((i+j)%26)+65))
        theVigenereMatrix.append(theMatrixRow)
    return theVigenereMatrix

def generateCircularKeyOfLength(key, size):
    # number of times to repeat the key
    timesToRepeat = math.ceil(size / len(key))
    theCircularKey = key
    for i in range(timesToRepeat):
        theCircularKey += key
    return theCircularKey[:size]

def getIndexFromRow(theRow, char):
    for i in range(len(theRow)):
        if(theRow[i] == char):
            return i

vigenereMatrix = generateVigenereMatrix()
```

```
print("Vigenere Cipher")
print("Enter Encrypted Text: ", end='')
encryptedText = input()
print("Enter the Key: ", end='')
key = input()

circularKey = generateCircularKeyOfLength(key, len(encryptedText))

# To decrypt -> Row Index -> Key, Col Index -> Where the Enc Text Char is
found - that col Index correspondign char

plaintext = ""

for i in range(len(encryptedText)):
    rowIndex = ord(circularKey[i])-65
    plainTextValue = chr(getIndexFromRow(vigenereMatrix[rowIndex],
encryptedText[i])+65)
    plaintext += plainTextValue

print("\nCircular Key: ", end='')
print(circularKey)

print("\nVigenere Table:")
printMatrix(vigenereMatrix)

print()
print()

print("Encrypted Text", encryptedText)
print("Plaintext:", plaintext)
```

We now solve some examples with the code.
Say we wish to encrypt: ‘CEASEFIRETILLDAWN’

We choose the option to directly type the string.
And let’s take key of ‘ELIZABETH’.

We run our code for encryption:

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\VigenereCipher> py .\encrypt.py
Vigenere Cipher
Enter Plaintext: CEASEFIRETILLDAWN
Enter the Key: ELIZABETH
```

It first generates the circular key:

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\VigenereCipher> py .\encrypt.py
Vigenere Cipher
Enter Plaintext: CEASEFIRETILLDAWN
Enter the Key: ELIZABETH

Circular Key: ELIZABETHELIZABET
```

Then generates the Vigenere Matrix

PROBLEMS OUTPUT GITLENS JUPYTER TERMINAL

Vigenere Table:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R

Then follows the algorithm to generate the cipher-text

```
P Q R S T U V W X Y Z A B C D E F G H I J K L M N O
Q R S T U V W X Y Z A B C D E F G H I J K L M N O P
R S T U V W X Y Z A B C D E F G H I J K L M N O P Q
S T U V W X Y Z A B C D E F G H I J K L M N O P Q R
T U V W X Y Z A B C D E F G H I J K L M N O P Q R S
U V W X Y Z A B C D E F G H I J K L M N O P Q R S T
V W X Y Z A B C D E F G H I J K L M N O P Q R S T U
W X Y Z A B C D E F G H I J K L M N O P Q R S T U V
X Y Z A B C D E F G H I J K L M N O P Q R S T U V W
Y Z A B C D E F G H I J K L M N O P Q R S T U V W X
Z A B C D E F G H I J K L M N O P Q R S T U V W X Y


Plaintext: CEASEFIRETILLDAWN
Encrypted Text GPIREGMKLXTTKDBAG
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\VigenereCipher>
```

Thus, cipher-text of ‘CEASEFIRETILLDAWN’ with key ‘ELIZABETH’ is ‘GPIREGMKLXTTKDBAG’.

Now, we decrypt using the code.

Our Cipher-text of ‘GPIREGMKLXTTKDBAG’ with key ‘ELIZABETH’:

It generates the Circular Key

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\VigenereCipher> py .\decrypt.py
Vigenere Cipher
Enter Encrypted Text: GPIREGMKLXTTKDBAG
Enter the Key: ELIZABETH

Circular Key: ELIZABETHELIZABET
```

Then generates the Vigenere Table:

```
Vigenere Table:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
B C D E F G H I J K L M N O P Q R S T U V W X Y Z A
C D E F G H I J K L M N O P Q R S T U V W X Y Z A B
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
E F G H I J K L M N O P Q R S T U V W X Y Z A B C D
F G H I J K L M N O P Q R S T U V W X Y Z A B C D E
G H I J K L M N O P Q R S T U V W X Y Z A B C D E F
H I J K L M N O P Q R S T U V W X Y Z A B C D E F G
I J K L M N O P Q R S T U V W X Y Z A B C D E F G H
J K L M N O P Q R S T U V W X Y Z A B C D E F G H I
K L M N O P Q R S T U V W X Y Z A B C D E F G H I J
L M N O P Q R S T U V W X Y Z A B C D E F G H I J K
M N O P Q R S T U V W X Y Z A B C D E F G H I J K L
N O P Q R S T U V W X Y Z A B C D E F G H I J K L M
O P Q R S T U V W X Y Z A B C D E F G H I J K L M N
P Q R S T U V W X Y Z A B C D E F G H I J K L M N O
Q R S T U V W X Y Z A B C D E F G H I J K L M N O P
R S T U V W X Y Z A B C D E F G H I J K L M N O P Q
S T U V W X Y Z A B C D E F G H I J K L M N O P Q R
```

And the decryption,

```
S T U V W X Y Z A B C D E F G H I J K L M N O P Q R
T U V W X Y Z A B C D E F G H I J K L M N O P Q R S
U V W X Y Z A B C D E F G H I J K L M N O P Q R S T
V W X Y Z A B C D E F G H I J K L M N O P Q R S T U
W X Y Z A B C D E F G H I J K L M N O P Q R S T U V
X Y Z A B C D E F G H I J K L M N O P Q R S T U V W
Y Z A B C D E F G H I J K L M N O P Q R S T U V W X
Z A B C D E F G H I J K L M N O P Q R S T U V W X Y


Encrypted Text GPIREGMKLXTTKDBAG
Plaintext: CEASEFIRETILLDAWN
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\VigenereCipher>
```

Thus, the plaintext of cipher-text ‘GPIREGMKLXTTKDBAG’ with key ‘ELIZABETH’ is ‘CEASEFIRETILLDAWN’.

We look at a final example:

Say for ‘LONDONBRIDGEHASFALLEN’ with key ‘CHARLES’

We get:

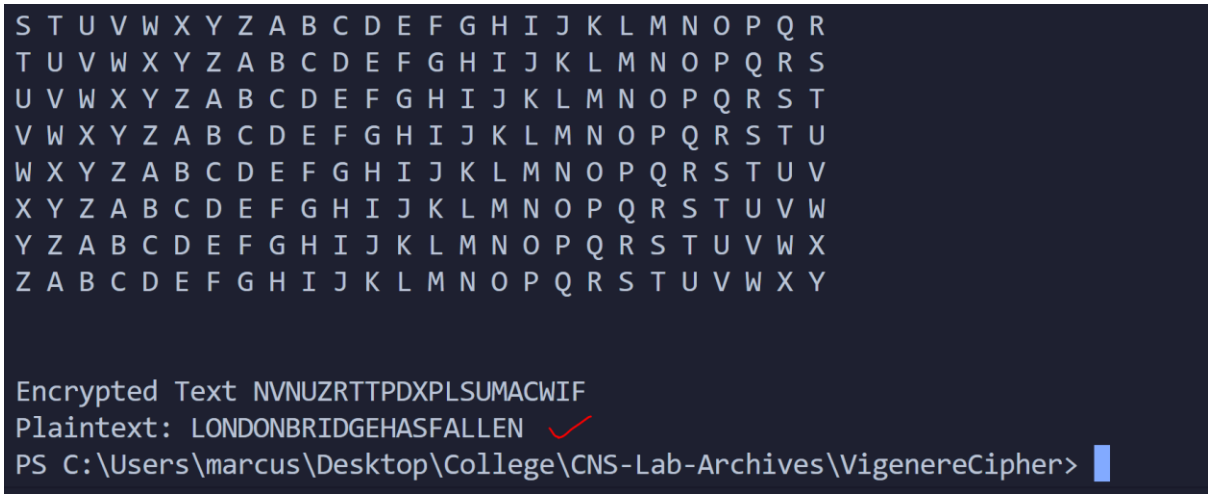
```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\VigenereCipher> py .\encrypt.py
Vigenere Cipher
Enter Plaintext: LONDONBRIDGEHASFALLEN
Enter the Key: CHARLES

Circular Key: CHARLESCHARLESCHARLES
```

```
T U V W X Y Z A B C D E F G H I J K L M N O P Q R S
U V W X Y Z A B C D E F G H I J K L M N O P Q R S T
V W X Y Z A B C D E F G H I J K L M N O P Q R S T U
W X Y Z A B C D E F G H I J K L M N O P Q R S T U V
X Y Z A B C D E F G H I J K L M N O P Q R S T U V W
Y Z A B C D E F G H I J K L M N O P Q R S T U V W X
Z A B C D E F G H I J K L M N O P Q R S T U V W X Y


Plaintext: LONDONBRIDGEHASFALLEN
Encrypted Text NVNUZRTTPDXPLSUMACWIF
```

Thus, the encrypted text would be, ‘NVNUZRTTPDXPLSUMACWIF’.

We can decrypt it as:

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\VigenereCipher> py .\decrypt.py
Vigenere Cipher
Enter Encrypted Text: NVNUZRTTPDXPLSUMACWIF
Enter the Key: CHARLES

Circular Key: CHARLESCHARLESCHARLES
```



We get our plaintext back.

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

Conclusion:

Thus, the Vigenere Cipher algorithm was studied and demonstrated with the code. It is observed that Vigenere Cipher is much stronger and harder to crack than the Caesar Cipher as the key can be anything and not one of 26 values as in Caesar Cipher.

Cryptography and Network Security Lab

Assignment 4
Implementation and Understanding of Playfair Cipher

2019BTECS00058
Devang K

Batch: B2

Title: Implementation and Understanding of Playfair Cipher

Aim: To Study, Implement and Demonstrate the Playfair Cipher Algorithm

Theory:

Playfair Cipher is a manual symmetric encryption technique invented by Wheatstone and brought in use by Playfair. The technique encrypts pairs of letters – diagrams and the algorithm is harder to break. It requires the Plaintext and the Encryption Key.

To encrypt in Playfair Cipher, we take the key and remove any ‘J’ instance. Then, in the plaintext, if we have an instance of ‘J’ we replace it with ‘I’. Then, we construct a PlayfairMatrix a 5*5 square – which starts by putting all the unique characters of the key in that order. Then, it is filled by all the characters not in the key (except ‘J’).

After this, the plaintext is divided into set of diagraphs, with condition that both characters have to be unique, if not, bogus character ‘Z’ is added. Same to be done at the end to balance all the diagraphs.

Then, for each plaintext diagraph, we shall generate the cipher as:
We take the characters on diagraphs and plot it on the PlayfairMatrix. Then, if:

Both are in the same column - Take the letter below each one (going back to the top if at the bottom).

Both are in the same row - Take the letter to the right of each one (going back to the leftmost if at the rightmost position).

For all other cases - Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

Then, combining the generated diagraphs would give us our cipher-text.

To decrypt the cipher, we again generate the PlayfairMatrix based off the key. Then we divide our encrypted text in a list of diagraphs. Then, for each diagraph if:

Both letters are in the same column - Take the letter above each one (going back to the bottom if at the top).

Both letters are in the same row - Take the letter to the left of each one (going back to the rightmost if at the leftmost position).

If neither - Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

Basically, doing the opposite.

Then, we finally combine these digraphs to generate the plain-text.

Illustration.

Say we wish to encrypt ‘ALLFORCESATTACK’ with key ‘ELIZABETH’.

We start by making the Playfair square. Filling with unique characters of the key eliminating ‘J’:

E L I Z A
B T H C D
F G K M N
O P Q R S
U V W X Y

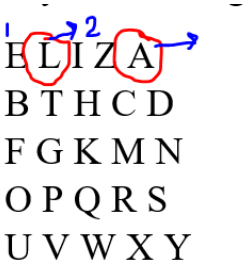
Then, we make the Playfair Diagraphs of the Plaintext.
First, we replace all ‘J’ with ‘I’. Then,

We take ‘AL’, they are unique and therefore can be diagraphs
Next, we take ‘LF’, ‘OR’ and so on. A case we encounter at ‘TTACK’, ‘TT’ cannot be taken, so we convert it to ‘TX’ and ‘TA’. Therefore, we get:

‘AL’, ‘LF’, ‘OR’, ‘CE’, ‘SA’, ‘TX’, ‘TA’, ‘CK’

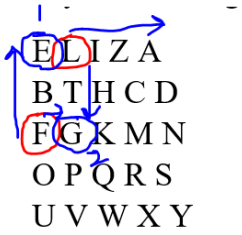
Then, for every diagraph, we encrypt using our Playfair Matrix.

Start with ‘AL’, same row



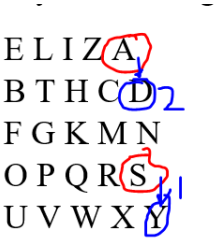
Therefore, the diagraph becomes ‘EI’.

Next, we find, ‘LF’, different rows and columns:



The encrypted diagraph becomes ‘EG’.

Similarly, we keep going till ‘SA’ – same column.



The encrypted diagraph becomes ‘YD’.

Thus, all the diagraphs can be encrypted and resultant ciphertext would be – ‘EIEGPSBZYDCVDLHM’.

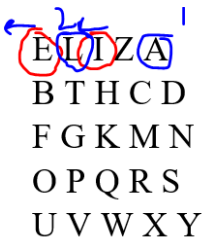
Now to decrypt, we again make the Playfair Matrix from the key and divide our ciphertext into diagraphs.

We get:
E L I Z A
B T H C D
F G K M N
O P Q R S
U V W X Y

‘EI’, ‘EG’, ‘PS’, ‘BZ’, ‘YD’, ‘CV’, ‘DL’, ‘HM’.

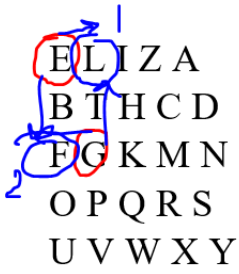
Now let us take a look at decryption cases of ‘EI’, ‘EG’ and ‘YD’ to cover our 3 cases:

For ‘EI’,



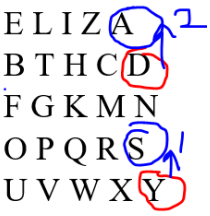
The Plaintext diagraph becomes ‘AL’.

For ‘EG’,



The Plaintext diagraph becomes ‘LF’

Finally, for ‘YD’,



The Plaintext diagraph becomes ‘SA’

Our Plaintext diagraphs are: ‘AL’, ‘LF’, ‘OR’, ‘CE’, ‘SA’, ‘TX’, ‘TA’, ‘CK’

Thus, our final plaintext would be: ‘ALLFORCESATXTACK’.

Code:

The crux of the application is in the functions that compute the Playfair Matrix and the Diagraph Generator. Rest of the functions are straightforward.

Code for Encryption:

```
alphabets = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

alphabetsList = []
for a in alphabets:
    alphabetsList.append(a)

def printMatrix(fenceMatrix):
    for i in range(len(fenceMatrix)):
        for j in range(len(fenceMatrix[0])):
            print(fenceMatrix[i][j], end=' ')
        print()


def generatekeyMatrix(key):
    # We are to make a 5*5 matrix

    # Splitting the key to a list
    keySet = []
    for k in key:
        keySet.append(k)
    keySet = list(dict.fromkeys(keySet))
    try:
        keySet.remove('J')
    except:
        pass
    for a in alphabetsList:
        if a != 'J':
            if a not in keySet:
                keySet.append(a)

    keyMatrix = []
    keyMatrix.append(keySet[0:5])
    keyMatrix.append(keySet[5:10])
    keyMatrix.append(keySet[10:15])
```

```
    keyMatrix.append(keySet[15:20])
    keyMatrix.append(keySet[20:25])

    return keyMatrix

def generatePlainTextDiagraphs(plaintext):
    # Replace all J with I
    plaintext = plaintext.replace("J", "I")
    lstOfDraphs = []
    aDiagraph = ""
    for i in range(len(plaintext)):
        if len(aDiagraph) == 0:
            aDiagraph+=plaintext[i]
        elif len(aDiagraph) == 1:
            if plaintext[i] == aDiagraph:
                aDiagraph += "X"
                lstOfDraphs.append(aDiagraph)
                aDiagraph = plaintext[i]
            else:
                aDiagraph += plaintext[i]
                lstOfDraphs.append(aDiagraph)
                aDiagraph = ""
    if len(aDiagraph) == 1:
        aDiagraph+="Z"
        lstOfDraphs.append(aDiagraph)
    return lstOfDraphs

def getPositionOfACharInKeyMatrix(keyMatrix, c):
    for i in range(5):
        for j in range(5):
            if keyMatrix[i][j] == c:
                return i,j

print("Playfair Cipher")
print("Enter Plaintext: ", end='')
plaintext = input()
print("Enter Key: ", end='')
key = input()

keyMatrix = generatekeyMatrix(key)

print("\nKey Matrix")
print(keyMatrix)

plainTextDiagraphs = generatePlainTextDiagraphs(plaintext)

print("\nPlaintext Diagraphs")
encryptedDigraphs = []
```

```
print(plainTextDiagraphs)
printMatrix(keyMatrix)

for ptd in plainTextDiagraphs:
    c1Row, c1Col = getPositionOfACharInKeyMatrix(keyMatrix, ptd[0])
    c2Row, c2Col = getPositionOfACharInKeyMatrix(keyMatrix, ptd[1])

    newDiagraph = ""

    # if both are in same column
    if c1Col == c2Col:
        newDiagraph+=keyMatrix[(c1Row+1)%5][c1Col]
        newDiagraph+=keyMatrix[(c2Row+1)%5][c2Col]
        encryptedDigraphs.append(newDiagraph)

    # if both are in same row
    elif c1Row == c2Row:
        newDiagraph+=keyMatrix[c1Row][(c1Col+1)%5]
        newDiagraph+=keyMatrix[c2Row][(c2Col+1)%5]
        encryptedDigraphs.append(newDiagraph)

    # else
    else:
        newDiagraph+=keyMatrix[c1Row][c2Col]
        newDiagraph+=keyMatrix[c2Row][c1Col]
        encryptedDigraphs.append(newDiagraph)

print("Plaintext: "+plaintext)
print("Encrypted Text: "+ "".join(encryptedDigraphs))
```

Code for Decryption:

```
alphabets = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

alphabetsList = []
for a in alphabets:
    alphabetsList.append(a)

def printMatrix(fenceMatrix):
    for i in range(len(fenceMatrix)):
        for j in range(len(fenceMatrix[0])):
            print(fenceMatrix[i][j], end=' ')
        print()

def generatekeyMatrix(key):
    # We are to make a 5*5 matrix
```

```
    # Splitting the key to a list
    keySet = []
    for k in key:
        keySet.append(k)
    keySet = list(dict.fromkeys(keySet))
    try:
        keySet.remove('J')
    except:
        pass
    for a in alphabetsList:
        if a != 'J':
            if a not in keySet:
                keySet.append(a)
    keyMatrix = []
    keyMatrix.append(keySet[0:5])
    keyMatrix.append(keySet[5:10])
    keyMatrix.append(keySet[10:15])
    keyMatrix.append(keySet[15:20])
    keyMatrix.append(keySet[20:25])
    return keyMatrix

def generatePlainTextDiagraphs(plaintext):
    # Replace all J with I
    plaintext = plaintext.replace("J", "I")
    lstOfDraphs = []
    aDiagraph = ""
    for i in range(len(plaintext)):
        if len(aDiagraph) == 0:
            aDiagraph+=plaintext[i]
        elif len(aDiagraph) == 1:
            if plaintext[i] == aDiagraph:
                aDiagraph += "X"
                lstOfDraphs.append(aDiagraph)
                aDiagraph = plaintext[i]
            else:
                aDiagraph += plaintext[i]
                lstOfDraphs.append(aDiagraph)
                aDiagraph = ""
    if len(aDiagraph) == 1:
        aDiagraph+="Z"
        lstOfDraphs.append(aDiagraph)
    return lstOfDraphs

def getPositionOfACharInKeyMatrix(keyMatrix, c):
    for i in range(5):
        for j in range(5):
            if keyMatrix[i][j] == c:
                return i,j
```

```
print("Playfair Cipher")
print("Enter Encrypted Text: ", end='')
enctext = input()
print("Enter Key: ", end='')
key = input()

keyMatrix = generatekeyMatrix(key)

print("\nKey Matrix")
printMatrix(keyMatrix)

encTextDiagraphs = generatePlainTextDiagraphs(enctext)
print("\nEnc Text Diagraphs")
print(encTextDiagraphs)

decryptedDigraphs = []

for ptd in encTextDiagraphs:
    c1Row, c1Col = getPositionOfACharInKeyMatrix(keyMatrix, ptd[0])
    c2Row, c2Col = getPositionOfACharInKeyMatrix(keyMatrix, ptd[1])

    newDiagraph = ""

    # if both are in same column
    if c1Col == c2Col:
        newDiagraph+=keyMatrix[(c1Row-1)%5][c1Col]
        newDiagraph+=keyMatrix[(c2Row-1)%5][c2Col]
        decryptedDigraphs.append(newDiagraph)

    # if both are in same row
    elif c1Row == c2Row:
        newDiagraph+=keyMatrix[c1Row][(c1Col-1)%5]
        newDiagraph+=keyMatrix[c2Row][(c2Col-1)%5]
        decryptedDigraphs.append(newDiagraph)

    # else
    else:
        newDiagraph+=keyMatrix[c1Row][c2Col]
        newDiagraph+=keyMatrix[c2Row][c1Col]
        decryptedDigraphs.append(newDiagraph)

print("Encrypted Text: "+enctext)
print("Plain Text: "+ "".join(decryptedDigraphs))
```

We now solve some examples with the code.

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

Running our code for encryption:

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\PlayfairCipher> py .\enc
rypt.py
Playfair Cipher
Enter Plaintext: CEASEFIRETILLDAWN
Enter Key: WINSTON
```

We generate the Playfair Matrix and Plaintext Diagraphs with respect to our conditions:

```
Plaintext Diagraphs
['CE', 'AS', 'EF', 'IR', 'ET', 'IL', 'LD', 'AW', 'NZ']
W I N S T
O A B C D
E F G H K
L M P Q R
U V X Y Z
```

Then, checking with the matrix for every plaintext diagraph, we get:

```
Plaintext Diagraphs
['CE', 'AS', 'EF', 'IR', 'ET', 'IL', 'LD', 'AW', 'NZ']
W I N S T
O A B C D
E F G H K
L M P Q R
U V X Y Z
Plaintext: CEASEFIRETILLDAWN
Encrypted Text: OHCIFGTMKWWMROOITX
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\PlayfairCipher>
```

Thus, our ciphertext is ‘OHCIFGTMKWWMROOITX’

Now, to decrypt:

With encrypted text – ‘OHCIFGTMKWWMROOITX’ and key – ‘WINSTON’

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\PlayfairCipher> py .\dec
rypt.py
Playfair Cipher
Enter Encrypted Text: OHCIFGTMKWWMROOITX
Enter Key: WINSTON
```

We generate the key matrix and the encrypted text diagraphs:

```
Key Matrix
W I N S T
O A B C D
E F G H K
L M P Q R
U V X Y Z

Enc Text Diagraphs
['OH', 'CI', 'FG', 'TM', 'KW', 'WM', 'RO', 'OI', 'TX']
```

Then,

```
Enter Encrypted Text: OHCIFGTMKWWMROOITX
Enter Key: WINSTON

Key Matrix
W I N S T
O A B C D
E F G H K
L M P Q R
U V X Y Z

Enc Text Diagraphs
['OH', 'CI', 'FG', 'TM', 'KW', 'WM', 'RO', 'OI', 'TX']
Encrypted Text: OHCIFGTMKWWMROOITX
Plain Text: CEASEFIRETILLDAWNZ
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\PlayfairCipher>
```

Thus, we get our plaintext back.

We take another example:

Say we wish to encrypt ‘ATTACKWITHFULLFORCE’ with key ‘MANTLE’.

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\PlayfairCipher> ^C
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\PlayfairCipher> py .\encrypt.py
Playfair Cipher
Enter Plaintext: ATTACKWITHFULLFORCE
Enter Key: MANTLE

KeyMarix
[['M', 'A', 'N', 'T', 'L'], ['E', 'B', 'C', 'D', 'F'], ['G', 'H', 'I', 'K', 'O'], ['P', 'Q', 'R', '
S', 'U'], ['V', 'W', 'X', 'Y', 'Z']]

Plaintext Diagraphs
['AT', 'TA', 'CK', 'WI', 'TH', 'FU', 'LX', 'LF', 'OR', 'CE']
M A N T L
E B C D F
G H I K O
P Q R S U
V W X Y Z
Plaintext: ATTACKWITHFULLFORCE
Encrypted Text: NLLNDIXHAKOZNZFOIUDB
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\PlayfairCipher>
```

Encrypted Text – ‘NLLNDIXHAKOZNZFOIUDB’.

To decrypt:

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\PlayfairCipher> py .\decrypt.py
Playfair Cipher
Enter Encrypted Text: NLLNDIXHAKOZNZFOIUDB
Enter Key: MANTLE

Key Matrix
M A N T L
E B C D F
G H I K O
P Q R S U
V W X Y Z

Enc Text Diagraphs
['NL', 'LN', 'DI', 'XH', 'AK', 'OZ', 'NZ', 'FO', 'IU', 'DB']
Encrypted Text: NLLNDIXHAKOZNZFOIUDB
Plain Text: ATTACKWITHFULXLFORCE
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\PlayfairCipher>
```

We get our plaintext back.

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

Conclusion:

Thus, the Playfair Cipher algorithm was studied and demonstrated with the code. The algorithm is observed to be difficult to break. Bogus characters can come in the middle of the decrypted plaintext.

Cryptography and Network Security Lab

Assignment 5.1
Implementation and Understanding of Rail Fence Cipher

2019BTECS00058
Devang K

Batch: B2

Title: Implementation and Understanding of Rail Fence Cipher

Aim: To Study, Implement and Demonstrate the Rail Fence Cipher Algorithm

Theory:

Rail fence cipher is a type of classical transpositional cipher which is also called zig-zag cipher. It gets name from the way in which it is encoded.

To encrypt using rail fence, we need the plaintext and the 'number of rails' i.e., the number of rows. The plaintext is written downwards diagonally on successive 'rails' – moving down till the last rail and then climbing up. It follows this until the plaintext is written out. Then, the Cipher-text is generated by traversing the rows.

To decrypt, we generate the table of the given fence size – length of size of encrypted text and height of the number of rails. We replace the position of a character with a special character – say '*'. Then we traverse our encrypted text character-by-character and replace '*' with the encrypted-text character. Finally, we traverse the shape in the zig-zag order to get the corresponding plaintext.

Illustration:

Encryption

Say we wish to encrypt – ‘ATTACKATONCE’ and say with 3 rails. We first make our matrix filled with blanks ‘-’ and then in the zig-zag pattern, plot our plaintext characters one-by-one. Finally, we then take append the characters going row-wise to generate our Cipher text.

Example:

Say, we wish to encrypt ‘DONTOPENTHEGATES’ with rail size 3. We first make the structure, and then plot the text in zig-zag pattern.

```
Fence Matrix with the Plaintext:
D---O---T---A------
-O-T-P-N-H-G-T-S---
--N---E---E---E----
```

We then take the characters row-wise to generate the Cipher-text.

Here, it would become – ‘DOTAOTPNHGTSNEEE’.

Decryption

Say we wish to decrypt our Ciphertext of ‘DOTAXOTPNHGTSNEEE’ with the rail size of 3. We shall first make the structure of our fence, replacing the characters with stars.

```
Initial Fence Matrix:
*---*---*---*------
-*-*-*-*-*-*-*-*---
--*---*---*---*----
```

Then we replace the ‘*’ with the cipher-text character going row-wise. This would make it:

```
Fence Matrix with the Encrypted Text:
D---O---T---A------
-O-T-P-N-H-G-T-S---
--N---E---E---E----
```

Now, in the pattern we made the '*', we shall traverse the fence to get our initial plaintext. Therefore, we get, 'DONTOPENTHEGATES'.

We illustrate more examples using the code.

Code:

The crux of the code would be the code to traverse in the rail fence pattern. The function looks like:

```
rowIndex = 0
colIndex = 0
slopeDown = True
for c in plaintext:
    fenceMatrix[rowIndex][colIndex] = c
    if slopeDown:
        rowIndex += 1
        if (rowIndex+1) > rails_count:
            slopeDown = False
            rowIndex -= 2
    else:
        rowIndex -= 1
        if (rowIndex) < 0:
            slopeDown = True
            rowIndex += 2
    colIndex += 1
```

Besides, the code to generate the matrix, traverse the elements and generating the cipher-text would be easy to implement. Similar is the case with decryption.

Code for Encryption:

```
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 getEncryptedTextFromFenceMatrix(fenceMatrix):
    encText = ""
    for i in range(len(fenceMatrix)):
        for j in range(len(fenceMatrix[0])):
            if fenceMatrix[i][j] != '-':
                encText += fenceMatrix[i][j]
    return encText

print("Rail Fence Encrypt\n")
print("Enter Plaintext: ", end='')
plaintext = input()
print("Enter Fence Size: ", end='')
rails_count = int(input())

fenceMatrix = []
# No. of Rows -> No. of Rails
# No. of Cols -> Size of Plaintext + No. Rails

noOfCols = len(plaintext) + rails_count

for i in range(rails_count):
    individualFence = []
    for j in range(noOfCols):
        individualFence.append("-")
    fenceMatrix.append(individualFence)

print("\nInitial Fence Matrix:")
printMatrix(fenceMatrix)

rowIndex = 0
colIndex = 0
slopeDown = True
for c in plaintext:
    fenceMatrix[rowIndex][colIndex] = c
    if slopeDown:
        rowIndex += 1
        if (rowIndex+1) > rails_count:
            slopeDown = False
            rowIndex -= 2
    else:
        rowIndex -= 1
        if (rowIndex) < 0:
            slopeDown = True
            rowIndex += 2
    colIndex += 1
```

```
while True:
    fenceMatrix[rowIndex][colIndex] = '-'
    if slopeDown:
        rowIndex += 1
        if (rowIndex+1) > rails_count:
            break
    else:
        rowIndex -= 1
        if (rowIndex) < 0:
            break
    colIndex += 1

print("\nFence Matrix with the Plaintext:")
printMatrix(fenceMatrix)
print()
print("Plaintext:", plaintext)
print("Encrypted Text: ",getEncryptedTextFromFenceMatrix(fenceMatrix))
```

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()

print("Rail Fence Decrypt\n")
print("Enter Encrypted Text: ", end='')
encryptedText = input()
print("Enter Fence Size: ", end='')
rails_count = int(input())

fenceMatrix = []
# No. of Rows -> No. of Rails
# No. of Cols -> Size of encryptedText + No. Rails

noOfCols = len(encryptedText) + rails_count

for i in range(rails_count):
    individualFence = []
    for j in range(noOfCols):
        individualFence.append("-")
    fenceMatrix.append(individualFence)

# We first mark the indices with '*'
```

```
rowIndex = 0
colIndex = 0
slopeDown = True
for c in range(len(encryptedText)):
    fenceMatrix[rowIndex][colIndex] = '*'
    if slopeDown:
        rowIndex += 1
        if (rowIndex+1) > rails_count:
            slopeDown = False
            rowIndex -= 2
    else:
        rowIndex -= 1
        if (rowIndex) < 0:
            slopeDown = True
            rowIndex += 2
    colIndex += 1

print("\nInitial Fence Matrix:")
printMatrix(fenceMatrix)

# Replace * with the Encrypted Text
encTextIndex = 0
for i in range(rails_count):
    for j in range(noOfCols):
        if fenceMatrix[i][j] == '*':
            fenceMatrix[i][j] = encryptedText[encTextIndex]
            encTextIndex += 1

plaintext = ''

rowIndex = 0
colIndex = 0
slopeDown = True
# Now we iterate in the rails and create the plain-text
for c in range(len(encryptedText)):
    plaintext+=fenceMatrix[rowIndex][colIndex]
    if slopeDown:
        rowIndex += 1
        if (rowIndex+1) > rails_count:
            slopeDown = False
            rowIndex -= 2
    else:
        rowIndex -= 1
        if (rowIndex) < 0:
            slopeDown = True
            rowIndex += 2
    colIndex += 1
```

```
print("\nFence Matrix with the Encrypted Text:")
printMatrix(fenceMatrix)
print()

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 rail-size as 5

We run our code for encryption:

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\RailfenceCipher> py .\encry
pt.py
Rail Fence Encrypt

Enter Plaintext: CEASEFIRETILLDAWN
Enter Fence Size: 5
```

Then, it shows us our initial fence:

```
Initial Fence Matrix:
----------------------
----------------------
----------------------
----------------------
----------------------
```

Then, it traverses and generates the zig-zag pattern:

```
Fence Matrix with the Plaintext:
C-------E-------N-----
-E-----R-T-----W------
--A---I---I---A-------
---S-F-----L-D--------
----E-------L---------
```

Finally, we get the encrypted string as:

```
Plaintext: CEASEFIRETILLDAWN
Encrypted Text:  CENERTWAIIASFLDEL
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Tr
```

Therefore, the ciphertext for ‘CEASEFIRETILLDAWN’ with rail-size 5 would be ‘CENERTWAIIASFLDEL’.

Now, we decrypt using the code.

We run using encrypted text ‘CENERTWAIIASFLDEL’ and rail-size 5.

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\RailfenceCipher> py .\decry
pt.py
Rail Fence Decrypt

Enter Encrypted Text: CENERTWAIIASFLDEL
Enter Fence Size: 5
```

We look at our initial fence matrix:

```
Initial Fence Matrix:
*_______*_______*_____
_*_____*_*_____*______
__*___*___*___*_______
___*_*_____*_*________
____*_______*_________
```

Then one that is filled with our encrypted text:

```
Fence Matrix with the Encrypted Text:
C-------E-------N-----
-E-----R-T-----W------
--A---I---I---A-------
---S-F-----L-D--------
----E-------L---------
```

Finally, we get our plaintext:

```
Fence Matrix with the Encrypted Text:
C-------E-------N-----
-E-----R-T-----W------
--A---I---I---A-------
---S-F-----L-D--------
----E-------L---------

Encrypted Text: CENERTWAIIASFLDEL
Plaintext: CEASEFIRETILLDAWN
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\RailfenceCipher>
```

Therefore, decrypting ‘CENERTWAIIASFLDEL’ with rail-size 5 we get ‘CEASEFIRETILLDAWN’.

Let’s take another example:

We wish to encrypt ‘HOLDTHEDOORHODOR’ with rail-size of 3.

Using our encryption code:

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\RailfenceCipher> py .\encry
pt.py
Rail Fence Encrypt

Enter Plaintext: HOLDTHEDOORHODOR
Enter Fence Size: 3

Initial Fence Matrix:
------------------
------------------
------------------

Fence Matrix with the Plaintext:
H---T---O---O------
-O-D-H-D-O-H-D-R---
--L---E---R---O----

Plaintext: HOLDTHEDOORHODOR
Encrypted Text:  HTOOODHDOHDRLERO
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\RailfenceCipher>
```

The encrypted text would be: ‘HTOOODHDOHDRLERO’.

Now, we decrypt:

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\RailfenceCipher> py .\decry
pt.py
Rail Fence Decrypt

Enter Encrypted Text: HTOOODHDOHDRLERO
Enter Fence Size: 3

Initial Fence Matrix:
*---*---*---*------
-*-*-*-*-*-*-*-*---
--*---*---*---*----

Fence Matrix with the Encrypted Text:
H---T---O---O------
-O-D-H-D-O-H-D-R---
--L---E---R---O----

Encrypted Text: HTOOODHDOHDRLERO
Plaintext: HOLDTHEDOORHODOR
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\RailfenceCipher>
```

We get our plaintext back.

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

Conclusion:

Thus, the Rail Fence Cipher algorithm was studied and demonstrated with the code. It is observed that it is a difficult algorithm to decipher when the number of words increases with a large value of key.

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:

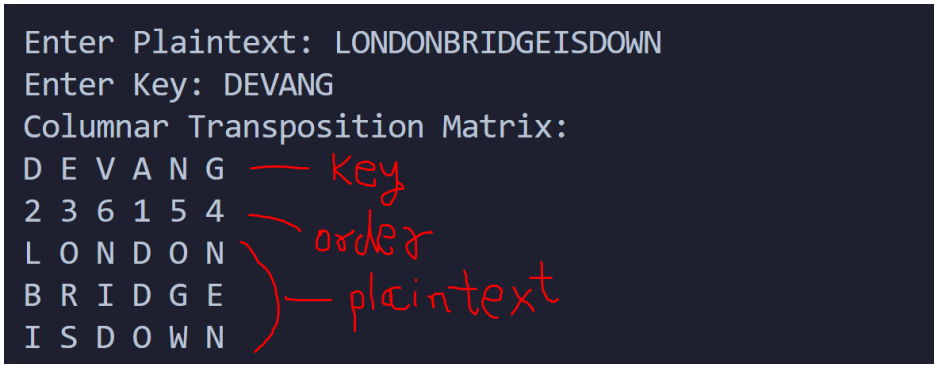
Let's illustrate using the plaintext of 'LONDONBRIDGEISDOWN' and a key of 'DEVANG'

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.

```
Columnar Transposition Matrix:
L O N D O N
B R I D G E
I S D O W N
```

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.

```
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
```

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(columnarTranspositionMatr
ix)

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:

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\ColumnarCipher> py .\encryp
t.py
Columnar Transposition Cipher

Enter Plaintext: CEASEFIRETILLDAWN
Enter Key: WINSTON
```

The Generated Columnar Matrix:

```
Columnar Transposition Matrix:
W I N S T O N
7 1 2 5 6 4 3
C E A S E F I
R E T I L L D
A W N X X X X
```

Therefore, the ciphertext would be:

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\ColumnarCipher> py .\encryp
t.py
Columnar Transposition Cipher

Enter Plaintext: CEASEFIRETILLDAWN
Enter Key: WINSTON
[['I', 'N', 'N', 'O', 'S', 'T', 'W'], [False, False, False, False, False, False, False]]
Columnar Transposition Matrix:
W I N S T O N
7 1 2 5 6 4 3
C E A S E F I
R E T I L L D
A W N X X X X
Plaintext: CEASEFIRETILLDAWN
Encrypted Text: EEWATNIDXFLXSIXELXCRA
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\ColumnarCipher>
```

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

To decrypt:

EncText – ‘EEWATNIDXFLXSIXELXCRA’ and key – ‘WINSTON’

We first get the key order ID

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\ColumnarCipher> py .\decryp
t.py
Columnar Transposition Cipher

Enter Encrypted Text: EEWATNIDXFLXSIXELXCRA
Enter Key: WINSTON
```

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

```
Columnar Transposition Matrix:
C E A S E F I
R E T I L L D
A W N X X X X
```

We finally get;

```
Encrypted Text: EEWATNIDXFLXSIXELXCRA
Plaintext: CEASEFIRETILLDAWNXXXX
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\ColumnarCipher>
```

Therefore, we get our plaintext back.

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

Encryption:

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\ColumnarCipher> py .\encryp
t.py
Columnar Transposition Cipher

Enter Plaintext: RETREATBACKTOBASE
Enter Key: MAJOR
[['A', 'J', 'M', 'O', 'R'], [False, False, False, False, False]]
Columnar Transposition Matrix:
M A J O R
3 1 2 4 5
R E T R E
A T B A C
K T O B A
S E X X X
Plaintext: RETREATBACKTOBASE
Encrypted Text: ETTETBOXRAKSRABXECAX
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\ColumnarCipher>
```

Ciphertext – ‘ETTETBOXRAKSRABXECAX’

Decryption:

```
t.py
Columnar Transposition Cipher

Enter Encrypted Text: ETTETBOXRAKSRABXECAX
Enter Key: MAJOR
[['A', 'J', 'M', 'O', 'R'], [False, False, False, False, False]]
Key Order List:
M A J O R
3 1 2 4 5

['ETTE', 'TBOX', 'RAKS', 'RABX', 'ECAX']
[['R', 'A', 'K', 'S'], ['E', 'T', 'T', 'E'], ['T', 'B', 'O', 'X'], ['R', 'A', 'B', 'X'], ['E', 'C',
 'A', 'X']]
Columnar Transposition Matrix:
R E T R E
A T B A C
K T O B A
S E X X X
Encrypted Text: ETTETBOXRAKSRABXECAX
Plaintext: RETREATBACKTOBASEXXX
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\TranspositionCipher\ColumnarCipher>
```

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.