Experiment 2

Date of Performance : 28-03-2022 **Date of Submission:** 28-03-2022

SAP Id: 60004190016 **Name : Aunali Patel**

Div: A **Batch : A1**

Aim of Experiment

Design Single Columnar Transposition Technique. Your program must Create two functions Encrypt() and Decrypt(). Your Program Must Input Image in Gray Scale / Color Image. Key is any line of string entered by the user. Number the key positions of letters in key string and apply encryption technique. You can club similar positional rows as per numbered key sequence.

Passed intermediate output of above step to another level of encryption. May be you can select simple Transpose of a Matrix or swapping upper triangular part with lower triangular part or any other innovative step. Care must be taken to map both encryption and decryption functions.

(CO1)

Theory / Algorithm / Conceptual Description

Simple Columnar Transposition cipher:

Columnar Transposition involves writing the plaintext out in rows, and then reading the ciphertext off in columns. In its simplest form, it is the Route Cipher where the route is to read down each column in order. For example, the plaintext "a simple transposition" with 5 columns looks like the grid below

A	S	I	M	P
L	E	T	R	A
N	S	P	O	S
I	T	I	O	N

Plaintext written across 5 columns.

So far this is no different to a specific route cipher. Columnar Transposition builds in a keyword to order the way we read the columns, as well as to ascertain how many columns to use.

Encryption

We first pick a keyword for our encryption. We write the plaintext out in a grid where the number of columns is the number of letters in the keyword. We then title each column with the respective letter from the keyword. We take the letters in the keyword in alphabetical order, and read down the columns in this order. If a letter is repeated, we do the one that appears first, then the next and so on.
As an example, let's encrypt the message "The tomato is a plant in the nightshade family" using the keyword tomato. We get the grid given below.

T	O	M	A	T	O
5	3	2	1	6	4
T	H	E	T	O	M
A	T	O	I	S	A
P	L	A	N	T	I
N	T	H	E	N	I
G	H	T	S	H	A
D	E	F	A	M	I
L	Y	X	X	X	X

We have written the keyword above the grid of the plaintext, and also the numbers telling us which order to read the columns in. Notice that the first "O" is 3 and the second "O" is 4, and the same thing for the two "T"s.

Starting with the column headed by "A", our ciphertext begins "TINESAX" from this column. We now move to the column headed by "M", and so on through the letters of the keyword in alphabetical order to get the ciphertext "TINESAX / EOAHTFX / HTLTHEY / MAIIAIX / TAPNGDL / OSTNHMX" (where the / tells you where a new column starts). The final ciphertext is thus "TINES AXEOA HTFXH TLTHE YMAII AIXTA PNGDL OSTNH MX".

Decryption

1. The decryption process is significantly easier if nulls have been used to pad out the message in the encryption process. Below we shall talk about how to go about decrypting a message in both scenarios.
2. Firstly, if nulls have been used, then you start by writing out the keyword and the alphabetical order of the letters of the keyword. You must then divide the length of the ciphertext by the length of the keyword. The answer to this is the number of rows you need to add to the grid. You then write the ciphertext down the first column until you reach the last row. The next letter becomes the first letter in the second column (by the alphabetical order of the keyword), and so on.
3. As an example, we shall decrypt the ciphertext "ARESA SXOST HEYLO IIAIE XPENG DLLTA HTFAX TENHM WX" given the keyword potato. We start by writing out the keyword and the order of the letters. There are 42 letters in the ciphertext, and the keyword has six letters, so we need $42 \div 6 = 7$ rows.

P	O	T	A	T	O
4	2	5	1	6	3
P	O	T	A	T	O
E	S	A	R	E	I
N	T	H	E	N	I
G	H	T	S	H	A
D	E	F	A	M	I
L	Y	A	S	W	E
L	L	X	X	X	X

The completely reconstructed grid.

Program Encrypt:

```
import numpy as np
import PIL  import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = [7.00, 3.50]
plt.rcParams["figure.autolayout"] = True

def imageImport(file):
  img = PIL.Image.open(file)
gray_img = img.convert("L")
print(gray_img.size)       imgArr
= np.array(gray_img)    return
imgArr

matrix=imageImport('aun.jpeg')



def encryption(image,key):
inputImageShape=image.shape
inputrows=inputImageShape[0]
  n=len(key)
```

```
 sortedKey=sorted(key)
divisible=inputImageShape[1]//len(sortedKey)

 numsRows=inputImageShape[0]
numCols=divisible*len(sortedKey)+inputImageShape[1]%len(sortedKey)
matrix1=np.zeros((numsRows,numCols))

 hashmap = { i : sortedKey[i] for i in range(0, len(sortedKey) ) }

 hashmap = {value:key for key, value in hashmap.items()}

 print(hashmap)

 transpose=matrix.T     transpose1=matrix1.T     for i in
range(0,numCols-inputImageShape[1]%len(sortedKey)):
   currentItem=key[i%len(key)]
   currentIndex=hashmap[currentItem]
   a = i//len(key)          b = len(key)*a
c = b + currentIndex
transpose1[c]=transpose[i]       final =
transpose1.T       plt.imshow(final)
flattening=final.flatten()
rolling=np.roll(flattening,numCols//2)
   reshaping=rolling.reshape(numsRows,numCols)
transposing = reshaping.T

 plt.imshow(transposing)
return transposing
```

```
Cipherimage = encryption(matrix,'AUNALI')
```

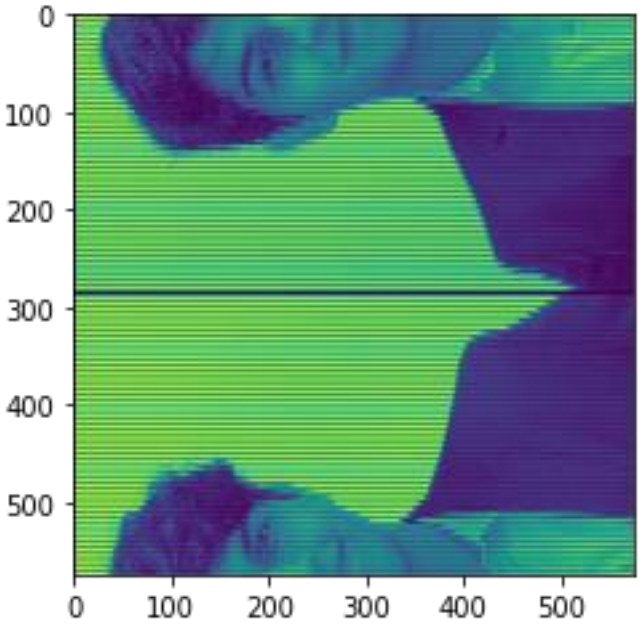
Decrypt:

```
def decryption(matrix,key):


  decrypttranspose = matrix.T
inputImageShape=decrypttranspose.shape
inputrows=inputImageShape[0]
  n=len(key)
  sortedKey=sorted(key)
divisible=inputImageShape[1]//len(sortedKey)


  numsRows=inputImageShape[0]
numCols=divisible*len(sortedKey)+inputImageShape[1]%len(sortedKey)
matrix1=np.zeros((numsRows,numCols))


  hashmap = { i : sortedKey[i] for i in range(0, len(sortedKey) ) }


  hashmap = {value:key for key, value in hashmap.items()}


  reflattening=decrypttranspose.flatten()
rerolling=np.roll(reflattening,numCols//2)
retranspose=rerolling.reshape(numsRows,numCols)
plt.imshow(retranspose)



  finaldecrypt=np.zeros((numsRows,numCols))
```

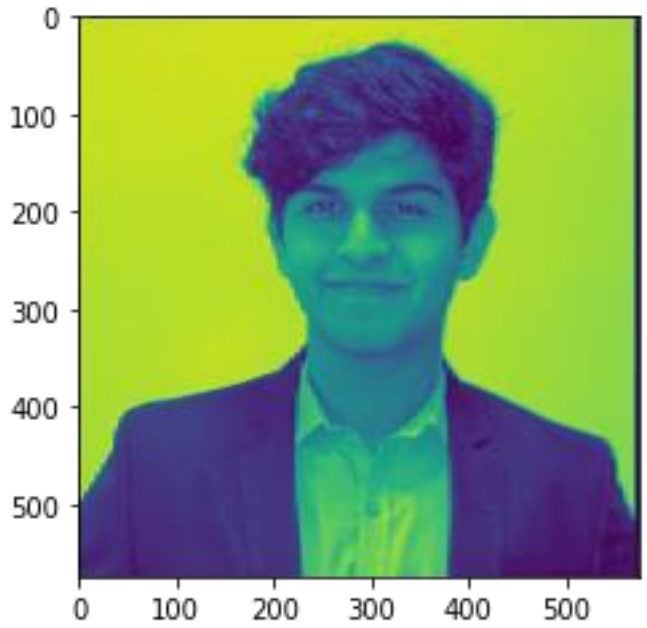
```
 retranspose1=retranspose.T      for i in range(0,numCols-
inputImageShape[1]%len(sortedKey)):
   currentItem=key[i%len(key)]
  currentIndex=hashmap[currentItem]
   a = i//len(key)          b =
len(key)*a          c = b +
currentIndex
finaldecrypt[i]=retranspose1[c]
finaldecrypt = finaldecrypt.T
plt.imshow(finaldecrypt)


decryption(Cipherimage,'AUNALI')
```

Output

After Encrypt:



After Decrypt:



Conclusion:

Hence, with the help of this experiment, we understood the conceptual, mathematical and practical aspect of Simple Columnar Transposition Technique. We implemented and performed Simple Columnar Transposition cipher on gray scale image using python.