**Param Patil**

**2019130049**

**TE Comps**

**Batch C**

**Experiment 1: Traditional Crypto Methods**

**Github Link:** [**https://github.com/pparam-2610/CSS\_LAB\_5thSem**](https://github.com/pparam-2610/CSS_LAB_5thSem)

**Aim:**

**To implement Substitution, ROT 13, Transposition, Double Transposition and Vernam Cipher in Python.**

**Code:**

import math

print('1 Substitution')

print('2 ROT 13')

print('3 Transpose')

print('4 Double Transposition')

print('5 Vernam Cipher')

a =int(input('Enter the cryptography method:'))

def encryptBySubstitution(text,shift = 1):

cipher = ""

shift = shift%26

for letter in text:

if(ord(letter)>96 and ord(letter)<=122):

new\_value = (ord(letter) + shift)

if(new\_value > 122):

new\_value -= 26

cipher += chr((96 + new\_value)%96 + 96)

elif(ord(letter)>64 and ord(letter)<=90):

new\_value = (ord(letter) + shift)

if(new\_value > 90):

new\_value -= 26

cipher += chr((65 + new\_value)%65 + 65)

else:

cipher +=chr(ord(letter) + shift)

return cipher

def decryptBySubstitution(text,shift = 1):

cipher = ""

shift = shift%26

for letter in text:

if(ord(letter)>96 and ord(letter)<=122):

new\_value = (ord(letter) - shift)

if(new\_value < 97):

new\_value += 26

cipher += chr((96 + new\_value)%96 + 96)

elif(ord(letter)>64 and ord(letter)<=90):

new\_value = (ord(letter) - shift)

if(new\_value < 65):

new\_value +=26

cipher += chr((64 + new\_value)%64 + 64)

else:

cipher +=chr(ord(letter) - shift)

return cipher

def encryptByVernam(text,key):

cipher = ""

for i in range(len(text)):

cipher += chr(((ord(text[i])-65)^(ord(key[i])-65))+65)

return cipher

def decryptByVernam(cipher,key):

text = ""

for i in range(len(cipher)):

text += chr(((ord(cipher[i]) - 65)^(ord(key[i]) - 65)) + 65)

return text

def encryptByTranspose(text,key):

col = len(key)

row = math.ceil(len(text)/col)

# print('The dimensions are: ',row,col,text)

matrix = []

for i in range(0,row):

string = []

for j in range(0,col):

string.append("\_")

matrix.append(string)

# print('The constructed matrix is: ', matrix)

text\_index = 0

for i in range(0,row):

for j in range(0,col):

matrix[i][j] = text[text\_index]

text\_index += 1

if(text\_index >= len(text)):

break

if(text\_index >= len(text)):

break

# print('The column matrix is: ', matrix)

refer\_key = sorted(key)

# print('The sorted list is: ',refer\_key)

cipher = ""

for i in range(0,col):

iterate\_col\_index = key.index(refer\_key[i])

key = key[:iterate\_col\_index] + '-' + key[iterate\_col\_index+1:]

for j in range(0,row):

cipher += matrix[j][iterate\_col\_index]

return cipher

def decryptByTranspose(cipher,key):

col = len(key)

row = math.ceil(len(cipher)/col)

matrix = []

for i in range(0,row):

string = []

for j in range(0,col):

string.append("\_")

matrix.append(string)

refer\_key = sorted(key)

cipher\_index = 0

for i in range(0,col):

iterate\_col\_index = key.index(refer\_key[i])

key = key[:iterate\_col\_index] + '-' + key[iterate\_col\_index+1:]

for j in range(0,row):

matrix[j][iterate\_col\_index] = cipher[cipher\_index]

cipher\_index += 1

if(cipher\_index >= len(cipher)):

break

if(cipher\_index >= len(cipher)):

break

text = ""

for i in range(0,row):

for j in range(0,col):

# if(matrix[i][j] == "\_"):

# continue

text += matrix[i][j]

return text

def filterText(text):

string = ""

for letter in text:

if(letter!="\_"):

string+=letter

return string

if a == 1:

text = input('Enter Plain Text: ')

key = int(input('No of position to be shifted: '))

encrpytedText = encryptBySubstitution(text,key)

print('Encrypted Message',encrpytedText)

print('Decrypted Message',decryptBySubstitution(encrpytedText,key))

elif a == 2:

text = input('Enter Plain Text to be encrypted: ')

encrpytedText = encryptBySubstitution(text, 13)

print('Encrypted Message', encrpytedText)

print('Decrypted Message', decryptBySubstitution(encrpytedText, 13))

elif a == 3:

text = input('Enter Plain Text to be encrypted: ')

key = input('Enter key: ')

encryptedText = encryptByTranspose(text,key)

decryptedText = decryptByTranspose(encryptedText,key)

print("The encrypted cipher is: ",filterText(encryptedText))

print("The decrypted text is: ",filterText(decryptedText))

elif a == 4:

text = input('Enter Plain Text to be encrypted: ')

key1 = input('Enter key 1: ')

key2 = input('Enter key 2: ')

encryptedText1 = (encryptByTranspose(text,key1))

encryptedText2 = (encryptByTranspose(text,key2))

decryptedText1 = decryptByTranspose(encryptedText2,key2)

decryptedText2 = decryptByTranspose(encryptedText1,key1)

print("The encrypted cipher is: ",filterText(encryptedText2))

print("The decrypted text is: ",filterText(decryptedText2))

elif a == 5:

text = input('Enter Plain Text to be encrypted: ')

key = input('Enter key of the same length as the Plain Text:')

while(len(text)!=len(key)):

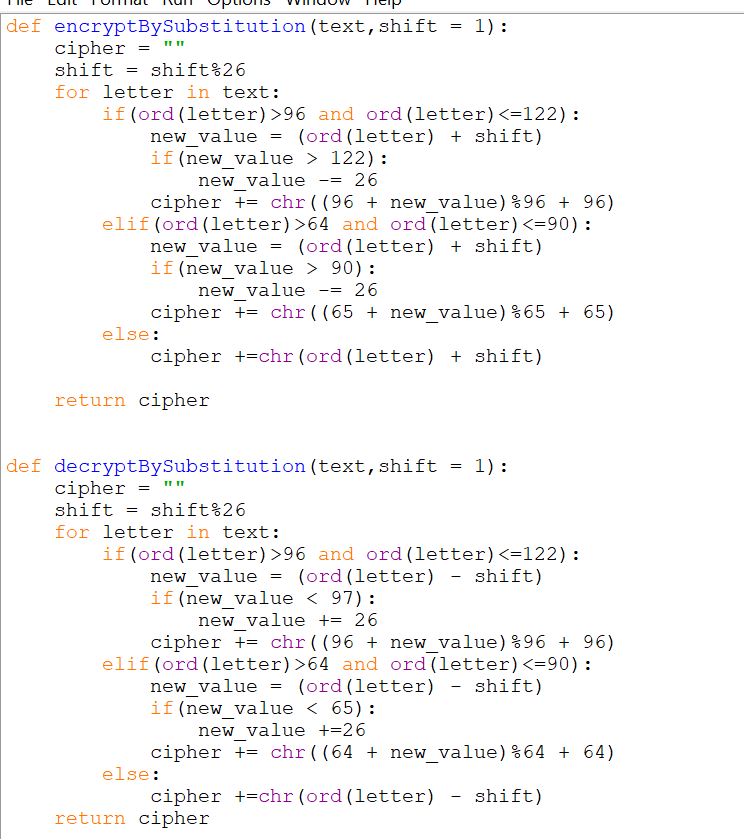
key = input('Enter key of the same length as the Plain Text: ')

encryptedText = (encryptByVernam(text,key))

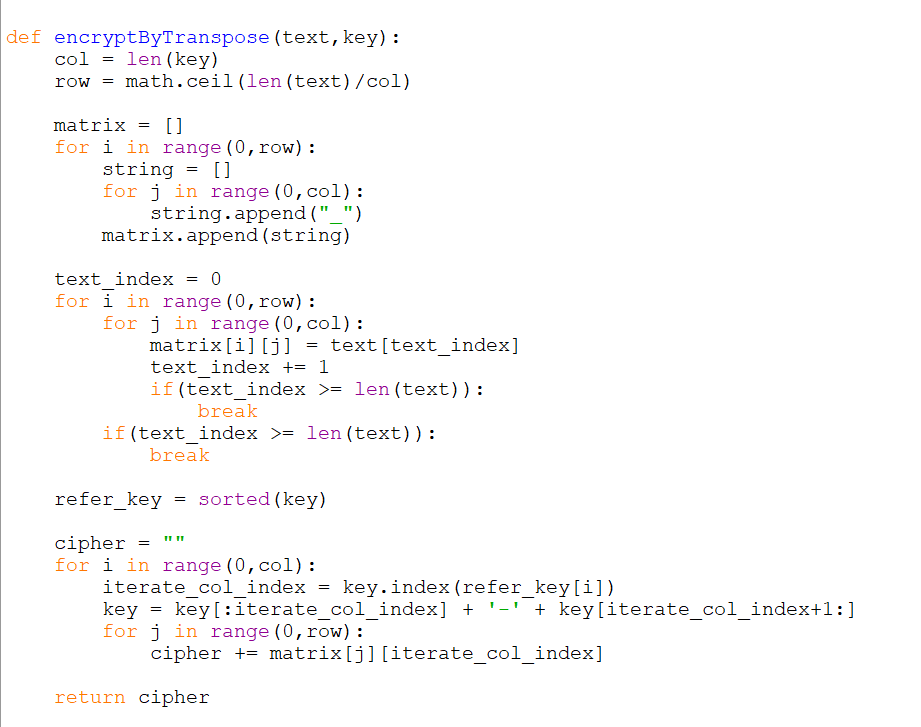
decryptedText = decryptByVernam(encryptedText,key)

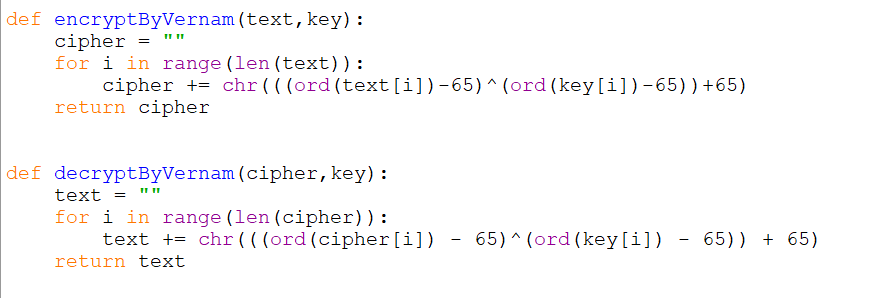
print("The encrypted cipher is: ",(encryptedText))

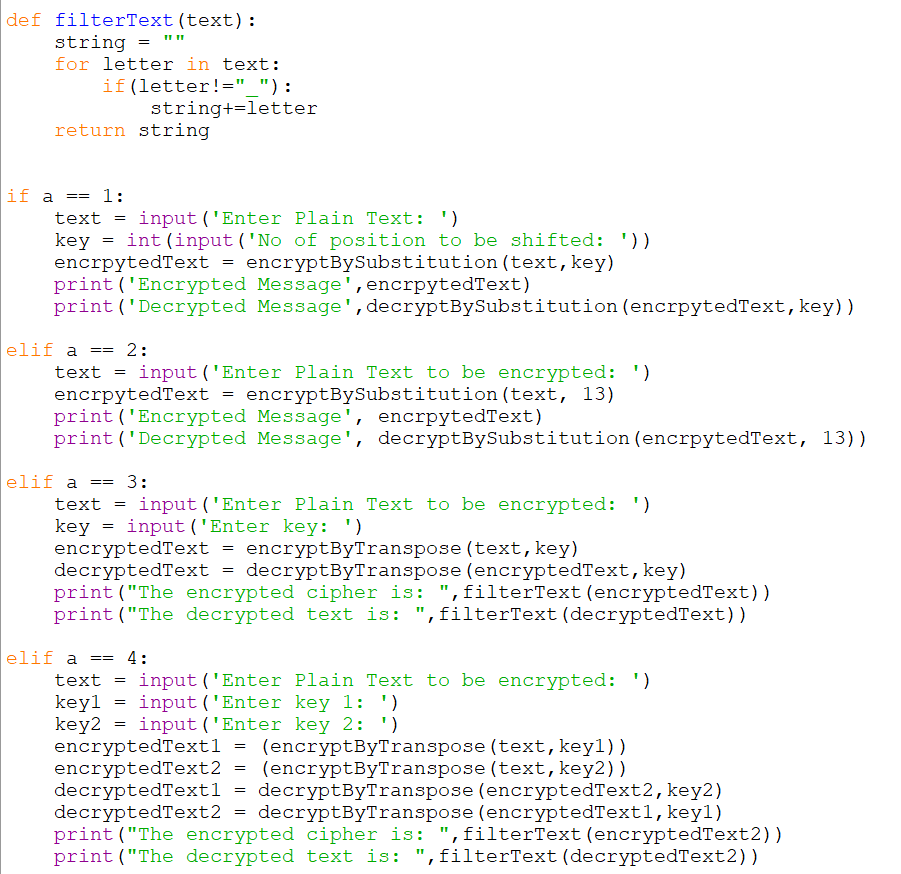
print("The decrypted text is: ",(decryptedText))

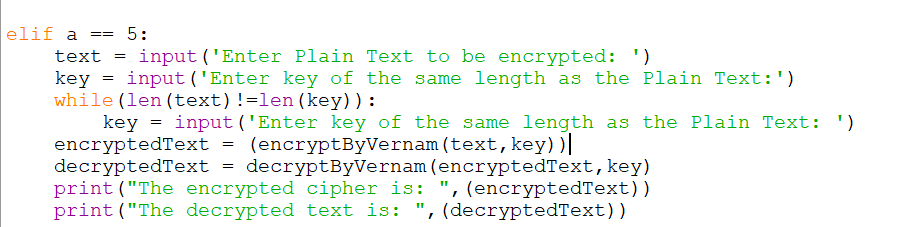




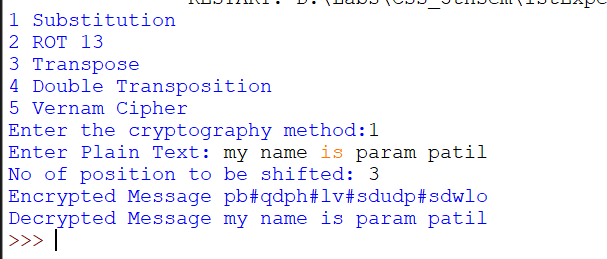




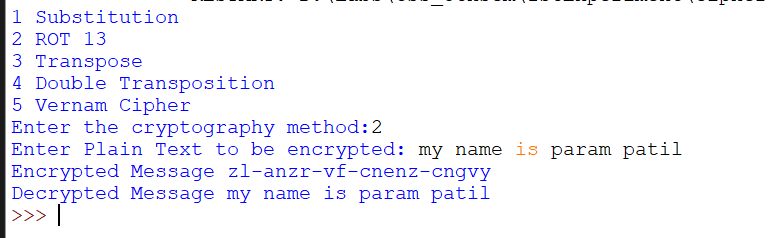




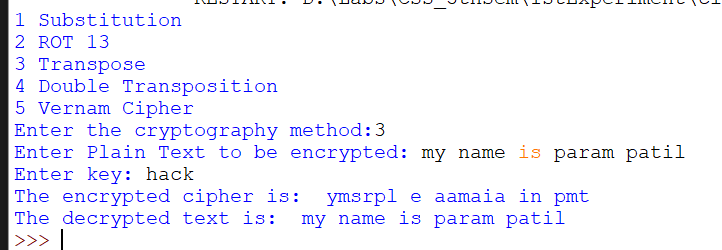
1. **Substitution Cipher**



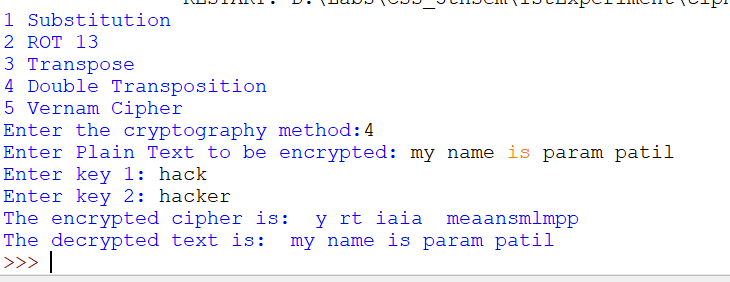
1. **ROT 13**



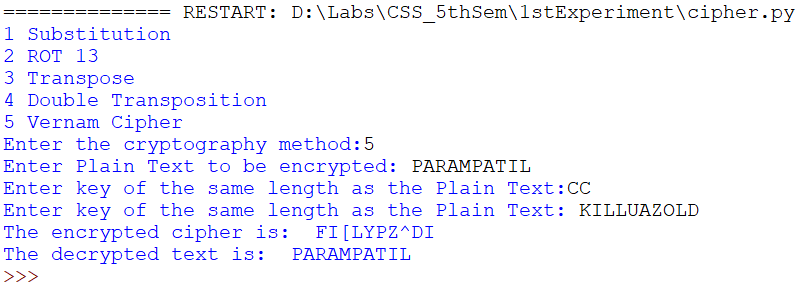
1. **Transpose Cipher**



1. **Double Transposition**



1. **Vernam Cipher**



**Conclusion:**

**In this experiment I have implemented 5 different traditional methods of encryption:**

**Substitution:** This algorithm substitutes all alphabets by a character representation of a certain number obtained by adding the shift (taken as an input in my implementation) with the ASCII value of the current alphabet. Here, the shift of the value can be infinite but to limit to alphabetic representation after encryption, the shift/key that was taken as an input is first modded by 26, as a result the maximum shift an alphabet can take is 26. Therefore, even if we consider all the ASCII values, there are finite values needed to crack this algorithm. Hence, this algorithm is not directly used in modern cryptography, but it lays the foundation for creation of complex cryptography algorithms.

**Rot 13:** This algorithm rotates the ASCII value of each alphabet by 13, in simple words it is a simple letter substitution cipher that replaces a letter with the letter that comes 13 letters after it in the alphabet., and in a cycle i.e z would be substituted with M. Therefore, the implementation of this algorithm is similar to substitution algorithm whose observations are mentioned above. Here the shift/key value is fixed to 13 and as a result this algorithm could easily be cracked by a normal human by simply observing the pattern and hence the applications of this algorithm is limited.

**Vernam:** This algorithm is polyalphabetic algorithm and encrypts each character with its corresponding character in the key, and therefore the length of the key must be the same as the length of the plain text. Since the value of the encrypted text is dependent on the key which is given as an input, the only way to crack the algorithm would be to try out various combinations until you decipher the key and during this process one would require strong computational power depending upon the length of the text. Because of all the factors mentioned above the applications of Vernam cipher is wide and active even in modern cryptography where the 8 bits word of a computer are encrypted in the similar manner as a 5-bit word of a teletype system( Vernam cipher was first used in early 1920s to encode the messages in teletype system).

**Transpose and Double Transpose:** In this algorithm the input text is transformed into a matrix whose column length would depend on the length of the key which is taken as an input. During conversion, it may not be necessary that the dimensions of the matrix would match with the length of the string where rows = len(plain text)/len(key). To overcome this problem, I added ‘\_’ at the end of the matrix (basically to handle the null spaces). Later on, we permute the matrix such that we can select the columns in ascending order of the key. To implement this feature, I sort the key and select the index from the actual input key of the first element in sorted key and append it to the main encryption text. To handle common alphabets in the key eg: hello, I maintained a flag for each letter to determine if the following letter has already been chosen or not. One important observation that could be noted is that the difficulty to crack the encrypted text would depend upon the length of the key and plain text i.e long plain text and short key would be difficult to crack then short text and short key. To make this algorithm more robust one could add another layer of encryption by using double transpose method. In modern times transpose cipher is not used directly but there are certain variations in transpose cipher to make it more difficult to crack eg: rail fence cipher, route cipher