LABORATORY

CEL62: Cryptography and System Security
Winter 2021

Experiment 1:	**Traditional Crypto Methods and Key Exchange**

Note: Students are advised to read through this lab sheet before doing experiment. On-the-spot evaluation may be carried out during or at the end of the experiment. Your performance, teamwork/Personal effort, and learning attitude will count towards the marks.

Name:: Chirag Rana
UID: 2018130043
TE COMPS

Experiment 1: Traditional Crypto Methods and Key Exchange

1 OBJECTIVE

This experiment will be in two parts:

1) To implement Substitution, ROT 13, Transposition, Double Transposition, and Vernam Cipher in Scilab/C/Python/R. 2) Implement Diffie Hellman key exchange algorithm in Scilab/C/Python/R.

2. INTROUCTION TO CRYTO AND RELEVANT ALGORITHMS

Cryptography:
In cryptography, encryption is the process of transforming information (referred to as plaintext) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key. The result of the process is encrypted information (in cryptography, referred to as cipher text). In many contexts, the word encryption also implicitly refers to the reverse process, decryption (e.g. “software for encryption” can typically also perform decryption), to make the encrypted information readable again (i.e. to make it unencrypted). Encryption is used to protect data in transit, for example data being transferred via networks (e.g. the Internet, e-commerce), mobile telephones, wireless microphones, wireless intercom systems, Bluetooth devices and bank automatic teller machines. There have been numerous reports of data in transit being intercepted in recent years/ Encrypting data in transit also helps to secure it as it is often difficult to physically secure all access to networks

Substitution Technique:
In cryptography, a substitution cipher is a method of encryption by which units of plaintext are replaced with ciphertext according to a regular system; the "units" may be single letters (the most common), pairs of letters, triplets of letters, mixtures of the above, and so forth. The receiver deciphers the text by performing an inverse substitution.

There are a number of different types of substitution cipher. If the cipher operates on single letters, it is termed a simple substitution cipher; a cipher that operates on larger groups of letters is termed polygraphic. A monoalphabetic cipher uses fixed substitution over the entire message, whereas a polyalphabetic cipher uses a number of substitutions at different times in the message, where a unit from the plaintext is mapped to one of several possibilities in the ciphertext and vice-versa.

Transposition Technique:
In cryptography, a transposition cipher is a method of encryption by which the positions held by units of plaintext (which are commonly characters or groups of characters) are shifted according to a regular system, so that the ciphertext constitutes a permutation of the plaintext. That is, the order of the units is changed. Mathematically a bijective function is used on the characters' positions to encrypt and an inverse function to decrypt.

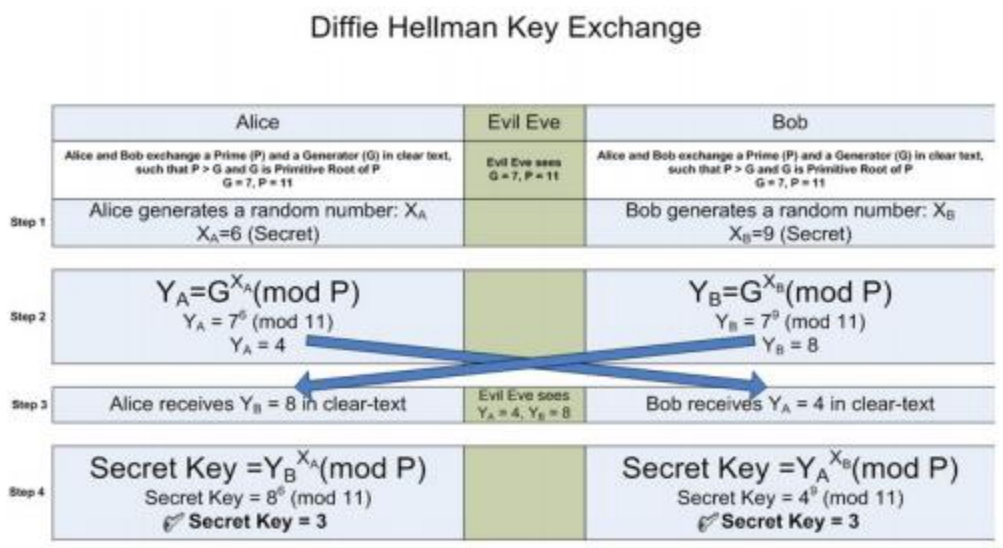
In a columnar transposition, the message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order. Both the width of the rows and the permutation of the columns are usually defined by a keyword. For example, the word ZEBRAS is of length 6 (so the rows are of length 6), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be "6 3 2 4 1 5".

In a regular columnar transposition cipher, any spare spaces are filled with nulls; in an irregular columnar transposition cipher, the spaces are left blank. Finally, the message is read off in columns, in the order specified by the keyword.

<u>Double Transposition</u>:
A single columnar transposition could be attacked by guessing possible column lengths, writing the message out in its columns (but in the wrong order, as the key is not yet known), and then looking for possible anagrams. Thus to make it stronger, a double transposition was often used. This is simply a columnar transposition applied twice. The same key can be used for both transpositions, or two different keys can be used.

<u>Vernam cipher</u>:
In modern terminology, a Vernam cipher is a symmetrical stream cipher in which the plaintext is XORed with a random or pseudo random stream of data (the "keystream") of the same length to generate the ciphertext. If the keystream is truly random and used only once, this is effectively a one-time pad. Substituting pseudorandom data generated by a cryptographically secure pseudo-random number generator is a common and effective construction for a stream cipher.

<u>Diffie –Hellman Key exchange algorithm:</u>
The Diffie–Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher. Although Diffie–Hellman key agreement itself is an anonymous (non-authenticated) key-agreement protocol, it provides the basis for a variety of authenticated protocols, and is used to provide perfect forward secrecy in Transport Layer Security's ephemeral modes (referred to as EDH or DHE depending on the cipher suite).



Diffie Hellman Key Exchange
Alice
Evil Eve
Bob
Alice and Bob exchange a Prime (P) and a Generator (G) in clear text, such that P > G and G is Primitive Root of P G = 7, P = 11
Evil Eve sees G = 7, P = 11
Alice and Bob exchange a Prime (P) and a Generator (G) in clear text, such that P > G and G is Primitive Root of P G = 7, P = 11
Step 1
Alice generates a random number: X_A X_A=6 (Secret)
Bob generates a random number: X_B X_B=9 (Secret)
Step 2
$Y_A=G^{X_A}(\mathrm{mod}\ P)$
$Y_A = 7^6\ (\mathrm{mod}\ 11)$
$Y_A = 4$
$Y_B=G^{X_B}(\mathrm{mod}\ P)$
$Y_B = 7^9\ (\mathrm{mod}\ 11)$
$Y_B = 8$
Step 3
Alice receives Y_B = 8 in clear-text
Evil Eve sees Y_A = 4, Y_B = 8
Bob receives Y_A = 4 in clear-text
Step 4
Secret Key $=Y_B^{X_A}(\mathrm{mod}\ P)$
Secret Key = 8^6 (mod 11)
Secret Key = 3
Secret Key $=Y_A^{X_B}(\mathrm{mod}\ P)$
Secret Key = 4^9 (mod 11)
Secret Key = 3

3 LAB TASKS

Write a single program which fits all algorithms. YOU should generate output in following manner:

1. Select the Cryptography Method Provide Choice 1…5 for subjected crypto methods
   a. Substitution
      i. Your choice
      ii. Enter Plain text to be encrypted
      iii. Enter the no. of Position shift
      iv. Encrypted Message
      v. Decrypted Message
   b. ROT 13
      i. Your choice
      ii. Enter Plain text to be encrypted
      iii. Encrypted Message
      iv. Decrypted Message
   c. Transpose
      i. Your choice
      ii. Enter Plain text to be encrypted
      iii. Encrypted Message
      iv. Decrypted Message
   d. Double Transposition
      i. Your choice
      ii. Enter Plain text to be encrypted
      iii. Encrypted Message
      iv. Decrypted Message
   e. Vernam Cipher
      i. Your choice
      ii. Enter Plain text to be encrypted
      iii. Input Key
      iv. Encrypted Message
      v. Decrypted Message
   f. Diffie Hellman
      i. Enter the Prime Number g:
      ii. Enter second Prime Number n:
      iii. Enter the Secret x:
      iv. Enter the Secret y
      v. $K_{1:}$
      vi. K_2:

4 SUBMISSION

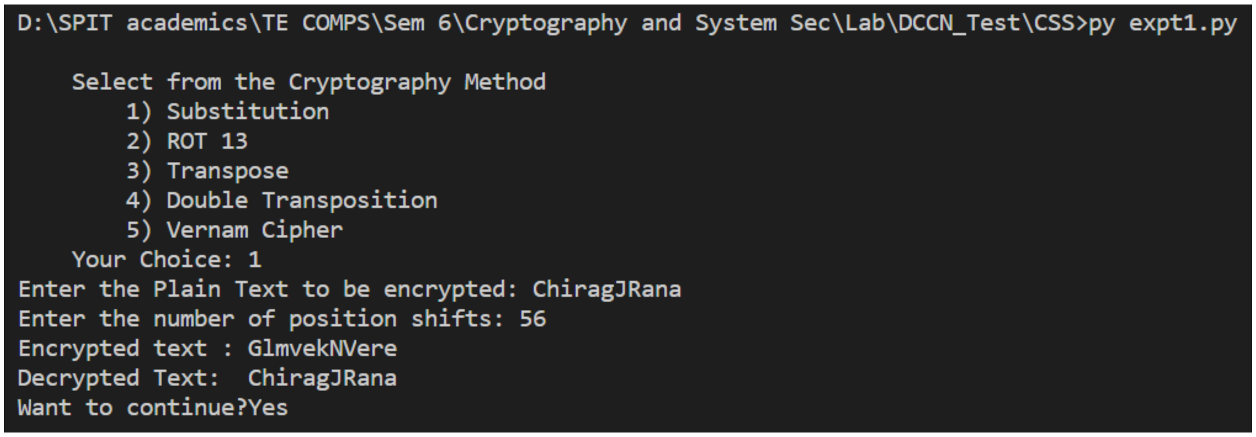
You need to submit a detailed lab report to describe what you have done and what you have observed as per the suggested output format for all method ; you also need to provide explanation to the observations that are interesting or surprising. In your report, you need to answer all the questions as per the suggested formant listed above.

Code:

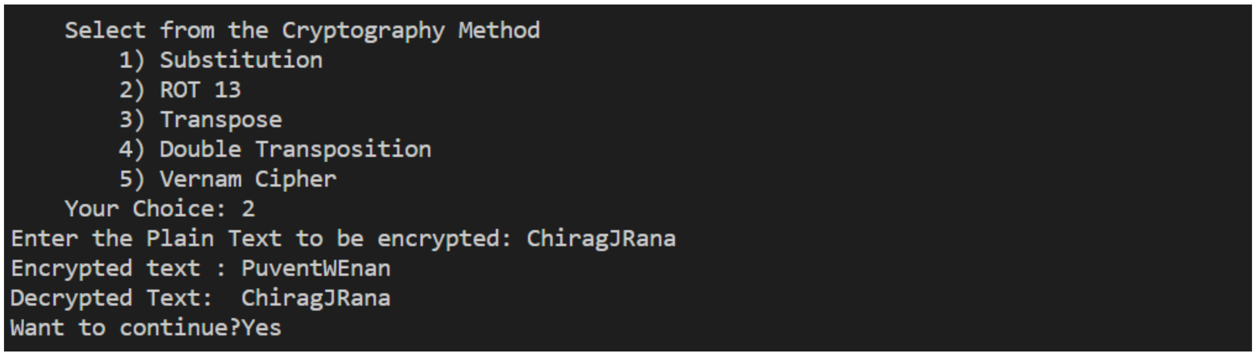
```
import string
import random
import math

def substitution(text, shift = 0):

   if not shift: # Check whether it is ROT13 or Substitution
      shift = int(input('Enter the number of position shifts: '))

   # Get the lower and upper case letters
   lowercase_letters=string.ascii_lowercase
   uppercase_letters=string.ascii_uppercase
   ll = len(lowercase_letters)
   ul = len(uppercase_letters)

   ########## ENCRYPTION  ##########

   #Create a dictionary to store all letters with their substitution using formula (x+n)mod26
   substitutionDict1 = {}

   for i in range(ll) :
      substitutionDict1[lowercase_letters[i]]= lowercase_letters[(i+shift)%ll]

   for i in range(ul) :
      substitutionDict1[uppercase_letters[i]]= uppercase_letters[(i+shift)%ul]

   cipher_text = ''
   for c in text:
      cipher_text += substitutionDict1[c] if c in lowercase_letters or c in uppercase_letters
else c

   print('Encrypted text :',cipher_text)

   ######### DECRYPTION ########

   # A dictionary for decrypting each letter (x-n)mod26
   substitutionDict2 = {}

   for i in range(ll) :
      substitutionDict2[lowercase_letters[i]]= lowercase_letters[(i-shift)%ll]

   for i in range(ul) :
      substitutionDict2[uppercase_letters[i]]= uppercase_letters[(i-shift)%ul]

   decrypt_text  = ''

   for char in cipher_text :
      decrypt_text   += substitutionDict2[char] if char in lowercase_letters or char in
uppercase_letters else char

   print('Decrypted Text: ', decrypt_text)


def rot13(text):
   substitution(text,shift = 13)

def Transpose(text):
   cipher_msg=''
   key=[]

   # Randomly generate a key
   key=random.sample(range(4),4)

   msg_len=len(text)
   col=len(key)
   row= int(math.ceil(msg_len/col))

   msg_list=list(text)   #Convert message into list

   ############# ENCRYPTION ###########

   fill_null = int((row * col) - msg_len)
   msg_list.extend(['-']*fill_null)
   # Put the _ character into empty cells at the end of the message.

   matrix=[ msg_list[i: i+col] for i in range(0,len(msg_list),col)]

   # Iterate matrix column wise
   for i in range(col):
      curr_idx=key[i]
      cipher_msg+=''.join([row[curr_idx] for row in matrix])
      # Add all the letters in the current column represented by curr_idx

   print('Encrypted text : ',cipher_msg)
   ############# DECRYPTION ###########
   decrypt_msg=''  #Decrypted message
   cipher_idx=0    #Keep track of ciphered message index
   c_list=list(cipher_msg) #Convert cipher message into a list

   # Matrix to store deciphered message
   dec_matrix = []
   for i in range(row):
      dec_matrix+=[[None]*col]    #Fill all cells of matrix with None

   # Arrange the message columnwise into the decipher matrix according to the key
   for i in range(col):
      curr_idx=key[i]

      for j in range(row):
          dec_matrix[j][curr_idx]=c_list[cipher_idx]
          #Take the letter from the cipher message and put it into correct position
          cipher_idx+=1

   decrypt_msg = ''.join(sum(dec_matrix, []))

   null_count = decrypt_msg.count('-')
   # Count of the _ character

   if null_count > 0:
      decrypt_msg= decrypt_msg[: -null_count] #Remove the _ character from the message

   print('Decrypted Text: ',decrypt_msg)

def Double_Transpose(text):
   cipher_msg1=''
   cipher_msg2=''
   key=[]

   key=random.sample(range(4),4)
   # Randomly generate a key

   msg_len=len(text)

   col=len(key)
   row=int(math.ceil(msg_len/col))

   ############ ENCRYPTION ##########

   msg_list1=list(text)   #Convert message into list
```

```
fill_null = int((row * col) - msg_len)
msg_list1.extend('-' * fill_null)
# Put the _ character into empty cells at the end of the message.

matrix1=[ msg_list1[i: i+col] for i in range(0,len(msg_list1),col) ]

# Iterate matrix column wise
for i in range(col):
    curr_idx=key[i]
    cipher_msg1+=''.join([row[curr_idx]
                for row in matrix1])
    # Add all the letters in the current column represented by curr_idx

#### Encrypt again
msg_list2=list(cipher_msg1)

matrix2=[ msg_list2[i: i+col]
        for i in range(0,len(msg_list2),col)]

for i in range(col):
    curr_idx=key[i]
    cipher_msg2+=''.join([row[curr_idx] for row in matrix2])

print('Encrypted text 1 : ',cipher_msg1)
print('Encrypted text 2: ',cipher_msg2)

############# DECRYPTION ###########

decrypt_msg1=''  #Decrypted message
decrypt_msg2=''  #Decrypted message

cipher_idx1=0    #Keep track of ciphered message index
c_list1=list(cipher_msg2) #Convert cipher message into a list

# Matrix to store deciphered message
dec_matrix1 = []
for i in range(row):
    dec_matrix1+=[[None]*col]    #Fill all cells of matrix with None

# Arrange the message columnwise into the decipher matrix according to the key
for i in range(col):
    curr_idx=key[i]
    for j in range(row):
            dec_matrix1[j][curr_idx]=c_list1[cipher_idx1]
            #Take the letter from the cipher message and put it into correct position
            cipher_idx1+=1

    # Convert decipher matrix into decrypted message
    # Sum() combines all the rwos in the matrix into a single array
    # And then join all the elements to form a string
    decrypt_msg1 = ''.join(sum(dec_matrix1, []))

    ###### decrypt again

    cipher_idx2=0       #Keep track of the first decrypted message
    c_list2=list(decrypt_msg1)

    # Second matrix
    dec_matrix2=[]
    for i in range(row):
        dec_matrix2+=[[None]*col]

    for i in range(col) :
        curr_idx=key[i]
        for j in range(row):
            dec_matrix2[j][curr_idx]=c_list2[cipher_idx2]
            cipher_idx2+=1

    decrypt_msg2=''.join(sum(dec_matrix2,[]))

    null_count = decrypt_msg2.count('-')
    # Count of the _ character

    if null_count > 0:
        decrypt_msg2= decrypt_msg2[: -null_count] #Remove the _ character from the message

    print('Decrypted Text 1: ',decrypt_msg1)
    print('Decrypted Text 2: ',decrypt_msg2)

def Vernam_Cipher(text):
    letters=string.ascii_lowercase
    key = input('Enter the key: ')
    msg_len=len(text)
    key_len = len(key)
    if key_len != msg_len:
        print("Enter the key of the same length")
        print("Try again")
        return

    # Convert strings into lists
    input_msg_list=list(text)
    key_list=list(key)
    # Numbers corresponding to the letters in the input msg and key
    msg_letter_numbers=[]
    key_letter_numbers=[]

    # Add numbers to list
    for i in range(msg_len):
        msg_letter_numbers.append(letters.index(input_msg_list[i]))
        key_letter_numbers.append(letters.index(key_list[i]))



    #########  ENCRYPTION #########

    cipher_list=[]

    # Iterate through the lists
    for i in range(len(msg_letter_numbers)) :
        # XOR the numbers from both lists
        sumval=msg_letter_numbers[i]+key_letter_numbers[i]
        sumval=sumval%26
        char=letters[sumval] # Get the character corresponding to the numbers
        cipher_list.append(char)

    cipher_text=''.join(cipher_list)    # The cipher text


    print('Encrypted text : ',cipher_text)
    #########  DECRYPTION #########

    dec_list=[]

    # Iterate through the lists
    for i in range(len(cipher_list)) :
        # XOR the numbers from both lists
        sumval=letters.index(cipher_list[i])-key_letter_numbers[i]
        sumval=sumval%26
        char=letters[sumval] # Get the character corresponding to the numbers
        dec_list.append(char)
    decrypt_msg=''.join(dec_list)
    print('Decrypted Text: ',decrypt_msg)



def main():

    option = int(input('''
    Select from the Cryptography Method
        1) Substitution
        2) ROT 13
        3) Transpose
        4) Double Transposition
        5) Vernam Cipher
    '''))

    options = {
        1: substitution,
        2: rot13,
        3: Transpose,
        4: Double_Transpose,
        5: Vernam_Cipher,
    }

    funct = options.get(option,lambda: 'Invalid')
    text= input('Enter the Plain Text to be encrypted: ')

    return funct(text)




if __name__ == '__main__':

    while True:
        main()
        condition = input('Want to continue?')
        if condition == 'No':
            break
```

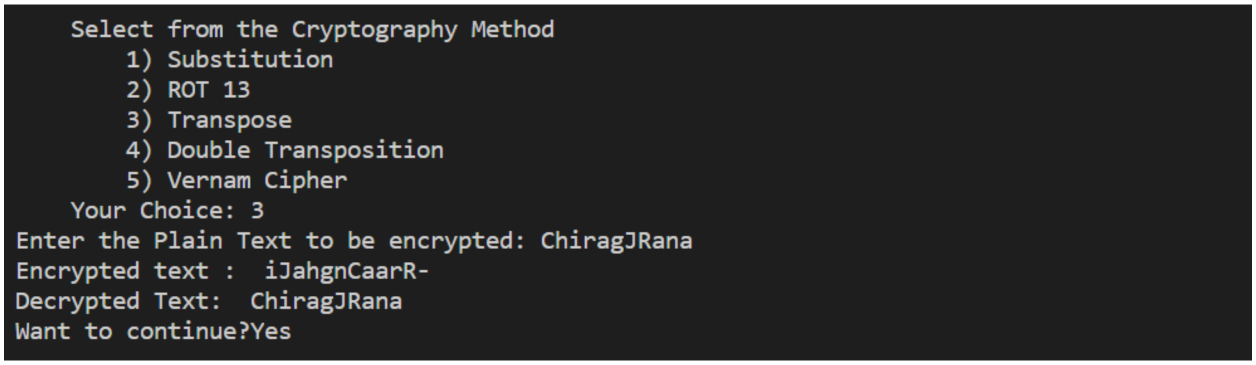
Output:
For Substition:


```
D:\SPIT academics\TE COMPS\Sem 6\Cryptography and System Sec\Lab\DCCN_Test\CSS>py expt1.py

    Select from the Cryptography Method
        1) Substitution
        2) ROT 13
        3) Transpose
        4) Double Transposition
        5) Vernam Cipher
    Your Choice: 1
Enter the Plain Text to be encrypted: ChiragJRana
Enter the number of position shifts: 56
Encrypted text : GlmvekNVere
Decrypted Text:  ChiragJRana
Want to continue?Yes
```

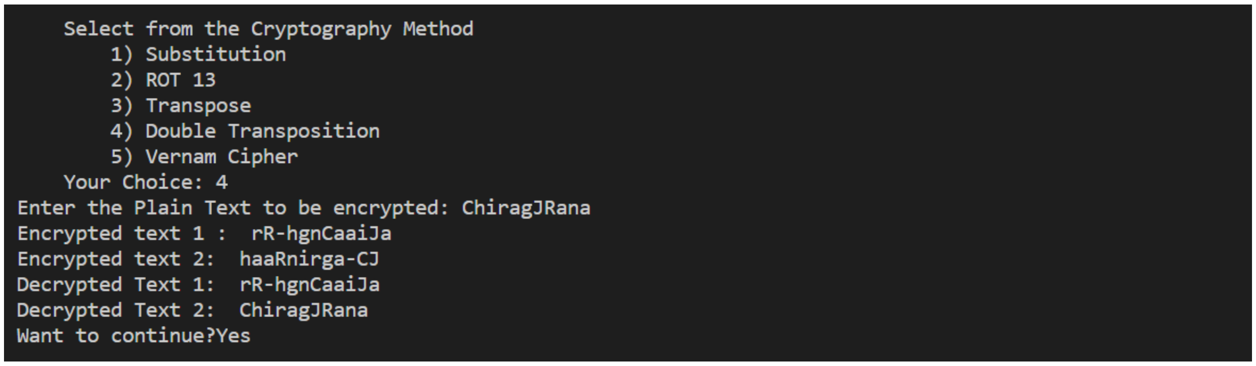
For ROT13 :


```
    Select from the Cryptography Method
        1) Substitution
        2) ROT 13
        3) Transpose
        4) Double Transposition
        5) Vernam Cipher
    Your Choice: 2
Enter the Plain Text to be encrypted: ChiragJRana
Encrypted text : PuventWEnan
Decrypted Text:  ChiragJRana
Want to continue?Yes
```

For Transpose:

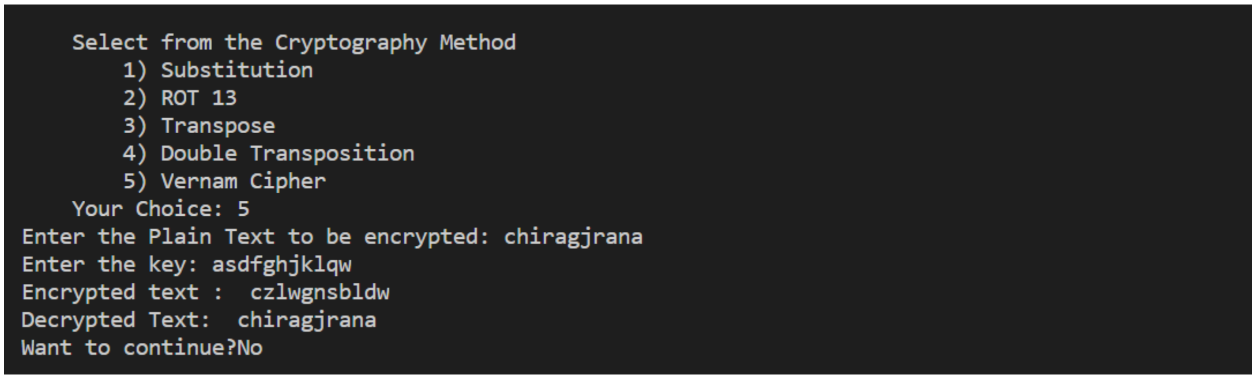

```
    Select from the Cryptography Method
        1) Substitution
        2) ROT 13
        3) Transpose
        4) Double Transposition
        5) Vernam Cipher
    Your Choice: 3
Enter the Plain Text to be encrypted: ChiragJRana
Encrypted text :  iJahgnCaarR-
Decrypted Text:  ChiragJRana
Want to continue?Yes
```

For Double Transpose:



```
    Select from the Cryptography Method
        1) Substitution
        2) ROT 13
        3) Transpose
        4) Double Transposition
        5) Vernam Cipher
    Your Choice: 4
Enter the Plain Text to be encrypted: ChiragJRana
Encrypted text 1 :  rR-hgnCaaiJa
Encrypted text 2:  haaRnirga-CJ
Decrypted Text 1:  rR-hgnCaaiJa
Decrypted Text 2:  ChiragJRana
Want to continue?Yes
```

For Vernam Cipher:



```
    Select from the Cryptography Method
        1) Substitution
        2) ROT 13
        3) Transpose
        4) Double Transposition
        5) Vernam Cipher
    Your Choice: 5
Enter the Plain Text to be encrypted: chiragjrana
Enter the key: asdfghjklqw
Encrypted text :  czlwgnsbldw
Decrypted Text:  chiragjrana
Want to continue?No
```

Observations:

1. In Substitution Method the characters are shifted by a value and it become easily broken by detecting the patterns in the cipher text.
2. ROT-13 is same as Substitution cipher method with fixed shift value making it more easy to determine the original text.
3. Columnar Transposition method uses a randomized order of columns to for encrypting so if the number of columns are more than it would accordingly take more timeto crack as the key length increases accordingly.
4. Transposition method adds additional text for creating the matrix properly of proper length which could further improvise the encryption process.
5. Double Transposition method uses the same Transposition method twice to encrypt it and hence requires 2 keys.
6. Vernam Cipher uses key of the same length as the Plain text to XOR the text with the key.