Vishal Salvi
2019230069
Batch C

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.

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.

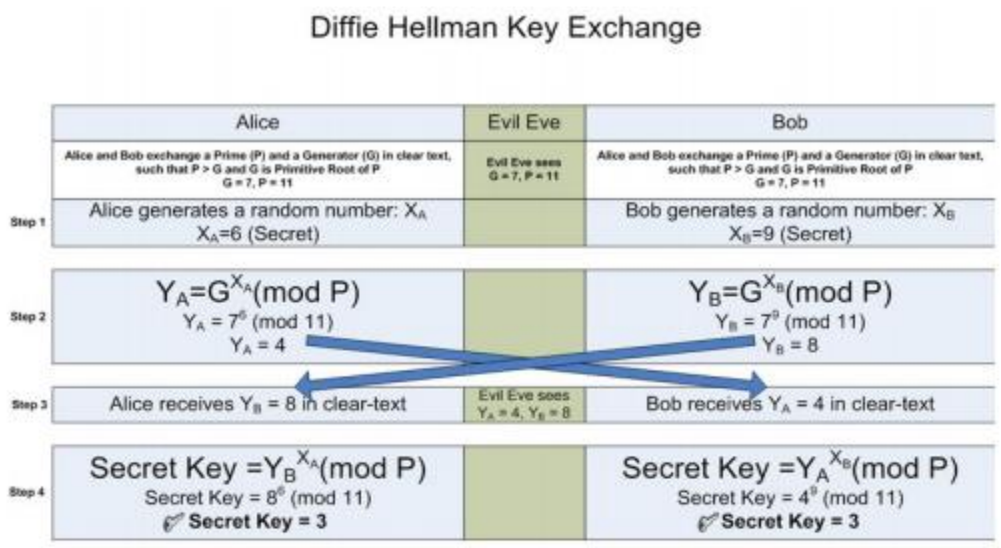
In a columnar transposition, the message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order. Both the width of the rows and the permutation of the columns are usually defined by a keyword. For example, the word ZEBRAS 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	$Y_A=G^{X_A}(\text{mod } P)$ $Y_A = 7^6$ (mod 11) $Y_A = 4$		$Y_B=G^{X_B}(\text{mod } P)$ $Y_B = 7^9$ (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 $=Y_B^{X_A}(\text{mod } P)$ Secret Key = 8^6 (mod 11) **Secret Key = 3**		Secret Key $=Y_A^{X_B}(\text{mod } P)$ Secret Key = 4^9 (mod 11) **Secret Key = 3**

Code:

```
print('\n1.Substitution')
print('2.ROT 13 ')
print('3.Transposition')
print('4.Double transposition')
print('5.Vernam')
print('6.Exit\n')
choice = input('Your choice: ')

import string
import random
def Text(encrypted, decrypted):
    print('Encrypted message:', encrypted)
    print('Decrypted message:', decrypted)


def substitution_cipher(input_text, shift):
    lowercase = string.ascii_lowercase
    uppercase = string.ascii_uppercase

    #Encryption
    substitution_dic = {} # Create Dictionary

    for i in range(len(lowercase)):
        substitution_dic[lowercase[i]] = lowercase[(i + shift) % len(lowercase)]

    for i in range(len(uppercase)):
        substitution_dic[uppercase[i]] = uppercase[(i + shift) % len(uppercase)]

    encrypt_text = []

    for j in input_text:
        if j in lowercase or j in uppercase:
            temp = substitution_dic[j]
            encrypt_text.append(temp) #append selected letter to cipher_text
        else:
            encrypt_text.append(j)

    encrypt_text = "".join(encrypt_text)

    #Decryption

    substitution_dic2 = {}  #Dictionary for decryption (x-n)mod26

    for i in range(len(lowercase)):
        substitution_dic2[lowercase[i]] = lowercase[(i - shift) % len(lowercase)]
    for i in range(len(uppercase)):
        substitution_dic2[uppercase[i]] = uppercase[(i - shift) % len(uppercase)]

    decrypt_text = []

    for char in encrypt_text:
        if char in lowercase or char in uppercase:
            decrypt_text.append(substitution_dic2[char])
        else:
            decrypt_text.append(char)

    decrypt_text = "".join(decrypt_text)
    Text(encrypt_text, decrypt_text)


def ROT(input_text, shift):
    lowercase = string.ascii_lowercase
    uppercase = string.ascii_uppercase

    #Encryption
    substitution_dic = {} # Create Dictionary

    for i in range(len(lowercase)):
        substitution_dic[lowercase[i]] = lowercase[(i + shift) % len(lowercase)]

    for i in range(len(uppercase)):
        substitution_dic[uppercase[i]] = uppercase[(i + shift) % len(uppercase)]

    encrypt_text = []

    for j in input_text:
        if j in lowercase or j in uppercase:
            temp = substitution_dic[j]
            encrypt_text.append(temp) #append selected letter to cipher_text
        else:
            encrypt_text.append(j)

    encrypt_text = "".join(encrypt_text)

    #Decryption

    substitution_dic2 = {}  #Dictionary for decryption (x-n)mod26

    for i in range(len(lowercase)):
        substitution_dic2[lowercase[i]] = lowercase[(i - shift) % len(lowercase)]

    for i in range(len(uppercase)):
        substitution_dic2[uppercase[i]] = uppercase[(i - shift) % len(uppercase)]
    decrypt_text = []

    for char in encrypt_text:
        if char in lowercase or char in uppercase:
            decrypt_text.append(substitution_dic2[char])
        else:
            decrypt_text.append(char)

    decrypt_text = "".join(decrypt_text)
    Text(encrypt_text, decrypt_text)

def transpose_cipher(input_text):
    cipher_msg = ''
    key = []

    key = random.sample(range(4), 4)    #random key
    msg_len = len(input_text)
    col = len(key)
    row = int(msg_len / col)

    if msg_len % col:
        row += 1
    msg_list = list(input_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)]  # Range increment by col
    for i in range(col):    #matrix column wise
        curr_idx = key[i]
        cipher_msg += ''.join([row[curr_idx]for row in matrix]) #Add all the letters in the current
column

    # Decryption
    decrypt_msg = ''  # Dec message
    cipher_idx = 0  # ciphered message index
    c_list = list(cipher_msg)  # message into a list


    dec_matrix = []     #deciphered message
    for i in range(row):
        dec_matrix += [[None] * col]  # none value store in deciphered matrix

    # arrange the message column wise
    for i in range(col):
        curr_idx = key[i]
        for j in range(row):
            dec_matrix[j][curr_idx] = c_list[cipher_idx]
            cipher_idx += 1

    # Sum() and join all the elements to form a string
    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

    Text(cipher_msg, decrypt_msg)

def double_transpose_cipher(input_text):
    cipher_msg1 = ''
    cipher_msg2 = ''
    key = []

    key = random.sample(range(4), 4)    #Random key
    msg_len = len(input_text)
    col = len(key)
    row = int(msg_len / col)

    if msg_len % col:
        row += 1
    msg_list1 = list(input_text)  # Convert message into list

    # Encryption
    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)]  # Range increment by col

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

    #Again Encrypt
    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])

# Decryption
decrypt_msg1 = ''
decrypt_msg2 = ''

cipher_idx1 = 0  # ciphered message index
c_list1 = list(cipher_msg2)  # message into a list

dec_matrix1 = []    #deciphered message
for i in range(row):
    dec_matrix1 += [[None] * col]  # none value store in deciphered matrix

# arrange the message column wise
for i in range(col):
    curr_idx = key[i]
    for j in range(row):
        dec_matrix1[j][curr_idx] = c_list1[cipher_idx1]
        cipher_idx1 += 1

# Sum() and join all the elements to form a string
decrypt_msg1 = ''.join(sum(dec_matrix1, []))

#Again decrypt
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
Text(cipher_msg2, decrypt_msg2)
```

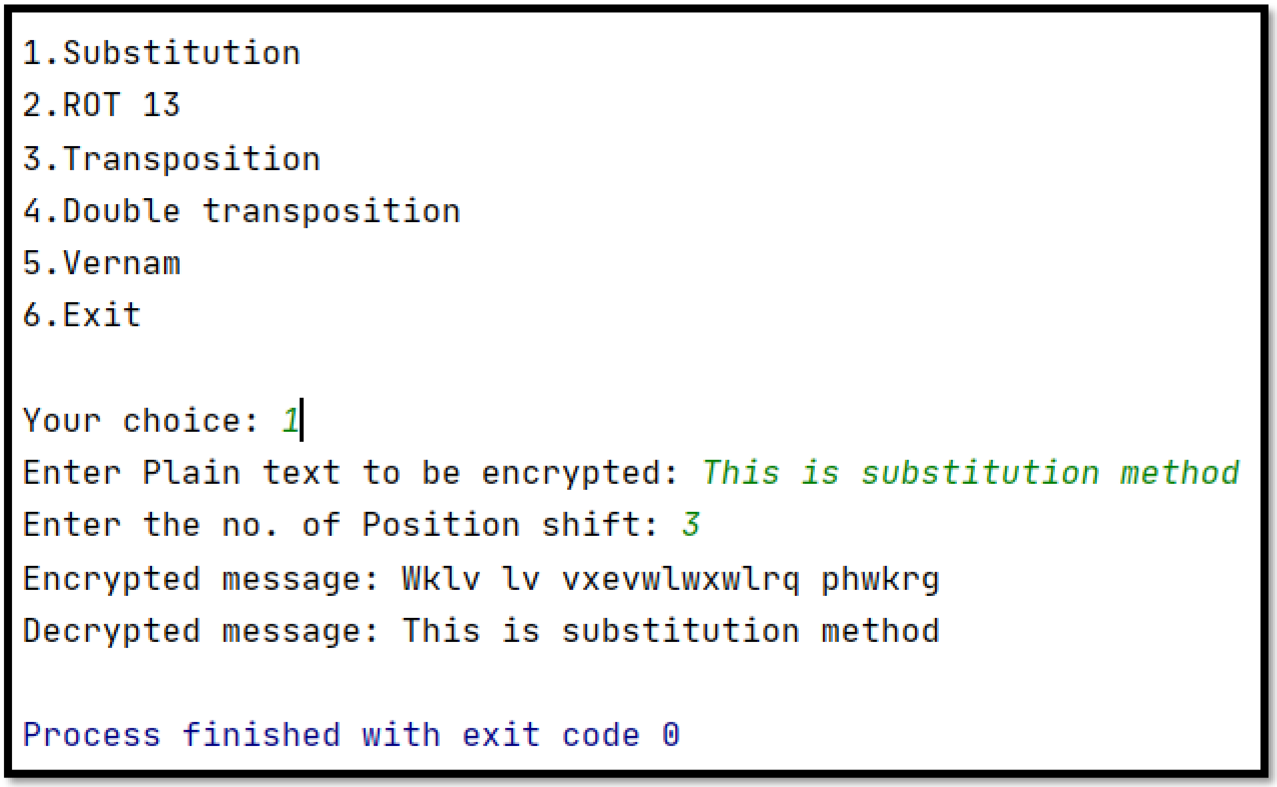
```
def vernam_cipher(input_text, key):
    letters = string.ascii_lowercase
    msg_len = len(input_text)
    input_msg_list = list(input_text) #Convert into list
    key_list = list(key)
    # No corresponding to the letters in the input msg and key
    msg_letter_no = []
    key_letter_no= []

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

    # Encryption
    cipher_list = []

    # Iterate through the lists
    for i in range(len(msg_letter_no)):    # XOR no from list
        sum = msg_letter_no[i] + key_letter_no[i]
        sum = sum % 26
        char = letters[sum]
        cipher_list.append(char)

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

    # Decryption
    dec_list = []

    # Iterate through the lists
    for i in range(len(cipher_list)): #XOR no from list
        sum = letters.index(cipher_list[i]) - key_letter_no[i]
        sum = sum % 26
        char = letters[sum]
        dec_list.append(char)

    decrypt_msg = ''.join(dec_list)
    Text(cipher_text, decrypt_msg)

if choice != '6':
    input_text = input('Enter Plain text to be encrypted: ')
    if choice == '1':
        shift = int(input('Enter the no. of Position shift: '))
        substitution_cipher(input_text, shift)
    elif choice == '2':
        ROT(input_text, 13)
    elif choice == '3':
        transpose_cipher(input_text)
```

```
elif choice == '4':
    double_transpose_cipher(input_text)
elif choice == '5':
    key = input('Input key: ')
    vernam_cipher(input_text, key)
```

3 LAB TASKS

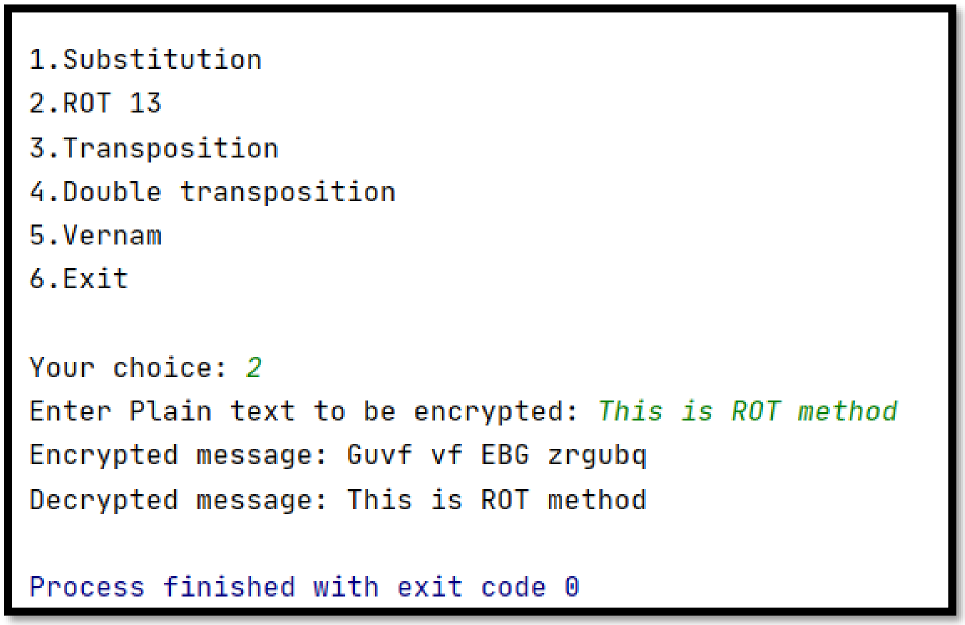
Write a single program which fits all algorithms. YOU should generate output in following manner:

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



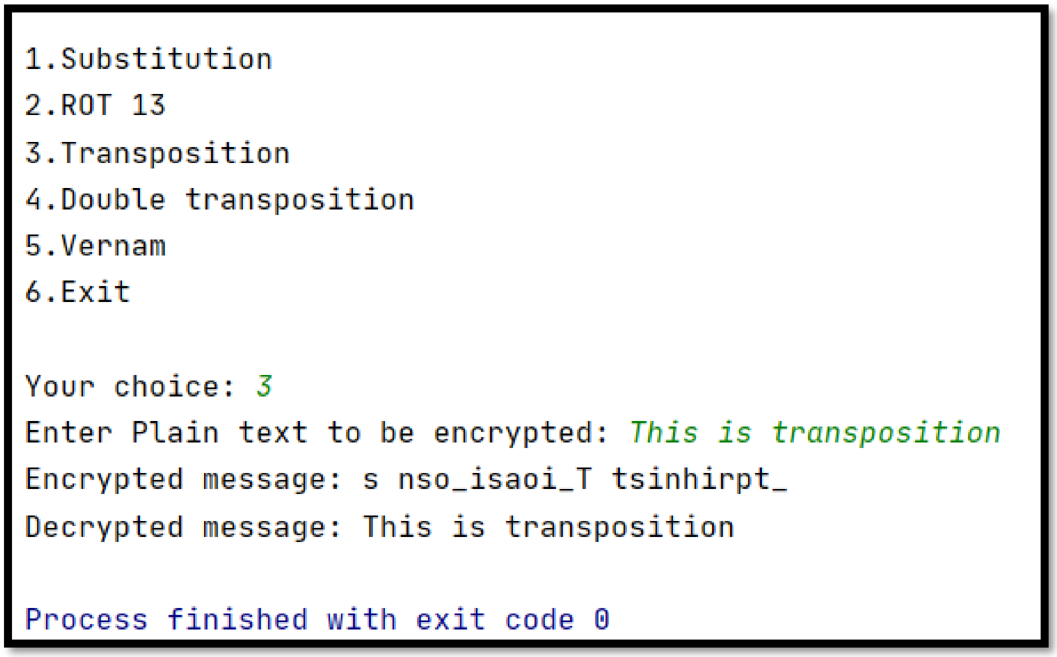
```
1.Substitution
2.ROT 13
3.Transposition
4.Double transposition
5.Vernam
6.Exit

Your choice: 1
Enter Plain text to be encrypted: This is substitution method
Enter the no. of Position shift: 3
Encrypted message: Wklv lv vxevwlwxwlrq phwkrg
Decrypted message: This is substitution method

Process finished with exit code 0
```

b. ROT 13
   i. Your choice
   ii. Enter Plain text to be encrypted
   iii. Encrypted Message
   iv. Decrypted Message



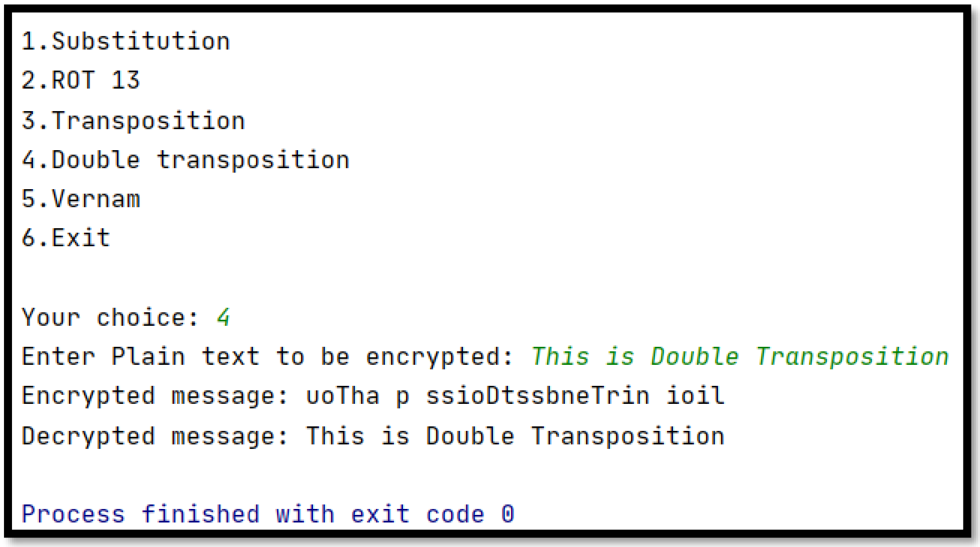
```
1.Substitution
2.ROT 13
3.Transposition
4.Double transposition
5.Vernam
6.Exit

Your choice: 2
Enter Plain text to be encrypted: This is ROT method
Encrypted message: Guvf vf EBG zrgubq
Decrypted message: This is ROT method

Process finished with exit code 0
```

c. Transpose
   i. Your choice
   ii. Enter Plain text to be encrypted
   iii. Encrypted Message
   iv. Decrypted Message



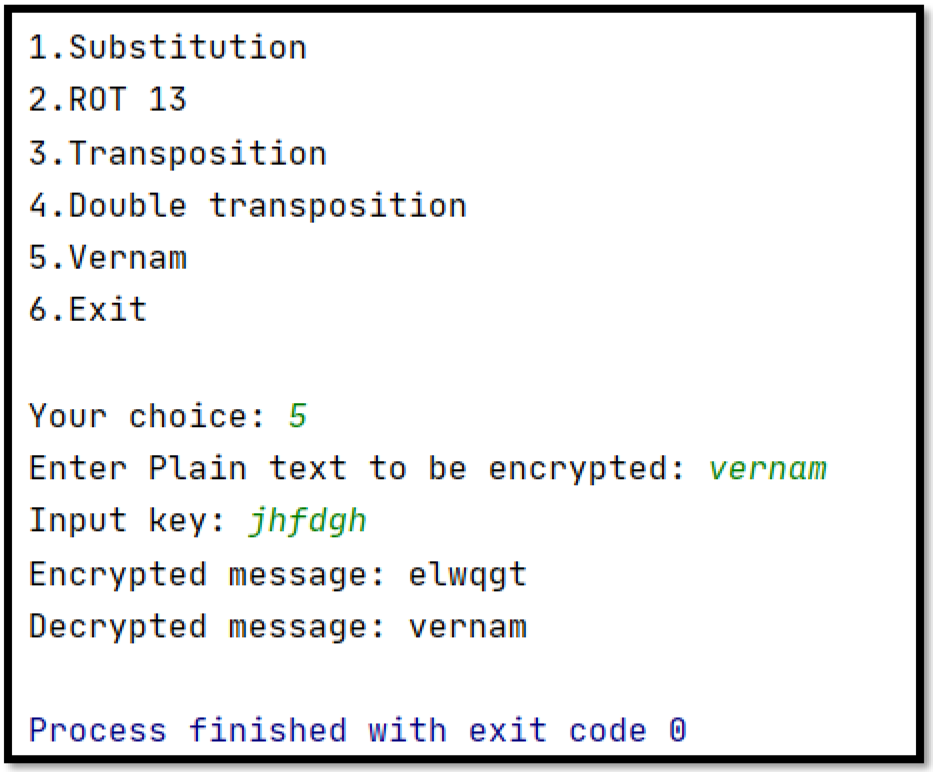
```
1.Substitution
2.ROT 13
3.Transposition
4.Double transposition
5.Vernam
6.Exit

Your choice: 3
Enter Plain text to be encrypted: This is transposition
Encrypted message: s nso_isaoi_T tsinhirpt_
Decrypted message: This is transposition

Process finished with exit code 0
```

d. Double Transposition
   i. Your choice
   ii. Enter Plain text to be encrypted
   iii. Encrypted Message
   iv. Decrypted Message



```
1.Substitution
2.ROT 13
3.Transposition
4.Double transposition
5.Vernam
6.Exit

Your choice: 4
Enter Plain text to be encrypted: This is Double Transposition
Encrypted message: uoTha p ssioDtssbneTrin ioil
Decrypted message: This is Double Transposition

Process finished with exit code 0
```

e. Vernam Cipher
   i. Your choice
   ii. Enter Plain text to be encrypted
   iii. Input Key
   iv. Encrypted Message
   v. Decrypted Message



```
1.Substitution
2.ROT 13
3.Transposition
4.Double transposition
5.Vernam
6.Exit

Your choice: 5
Enter Plain text to be encrypted: vernam
Input key: jhfdgh
Encrypted message: elwqgt
Decrypted message: vernam

Process finished with exit code 0
```

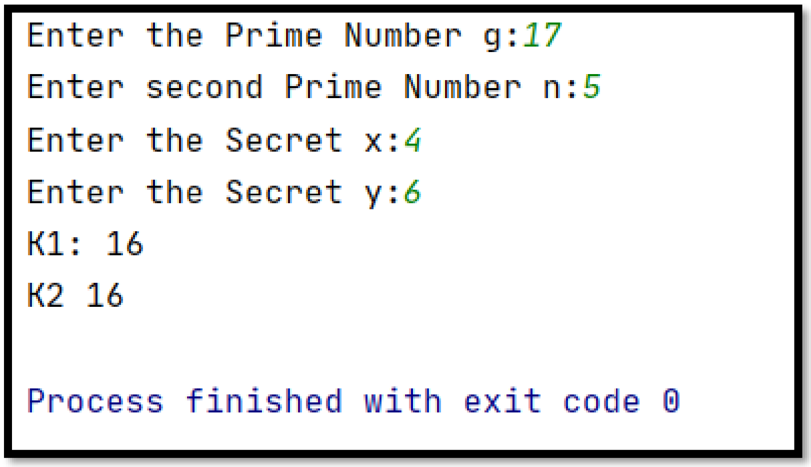
f. Diffie Hellman

   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:

Code:

```
#1st prime number
g=int(input("Enter the Prime Number g:"))
#2nd prime number
n=int(input("Enter second Prime Number n:"))
#Enter 1st Secret Key
x=int(input("Enter the Secret x:"))
#Enter 2nd Secret Key
y=int(input("Enter the Secret y:"))
#k1=(n^x mod(g))
k1=(n**x)%g
#k2=(n^y mod(g))
k2=(n**y)%g
#Exchange of values take place
#k1_key=(k2^x mod(g))
k1_key=(k2**x)%g
#k2_key=(k2^x mod(g))
k2_key=(k1**y)%g
print("K1:",k1_key)
print("K2",k2_key)
```

Output:



```
Enter the Prime Number g:17
Enter second Prime Number n:5
Enter the Secret x:4
Enter the Secret y:6
K1:  16
K2 16

Process finished with exit code 0
```

Observation:

1) Substitution Cipher Technique is a traditional cipher text technique which is used to encrypt a plain text into cipher text. In this technique, each character is substituted with other character/number or other symbol. This techniques changes identity of a character but not the position of it.
2) The ROT13 cipher is not very secure as it is just a special case of the Caeser cipher. The Caeser cipher can be broken by either frequency analysis or by just trying out all 25 keys whereas the ROT13 cipher can be broken by just shifting the letters 13 places. Therefore, it has no practical use.
3) Transposition Cipher Technique is also a traditional cipher text technique which is used to encrypt a plain text into cipher text. In this technique, each character position is changed to different position.
4) Double transportation can make the job of the cryptanalyst difficult. It designates the letters in the original plaintext message by the numbers designating their position.
5) The Vernam cipher is, a perfect cipher. Instead of a single key, each plaintext character is encrypted using its own key.
6) Diffie Hellman (DH) key exchange algorithm is a method for securely exchanging cryptographic keys over a public communications channel.