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.

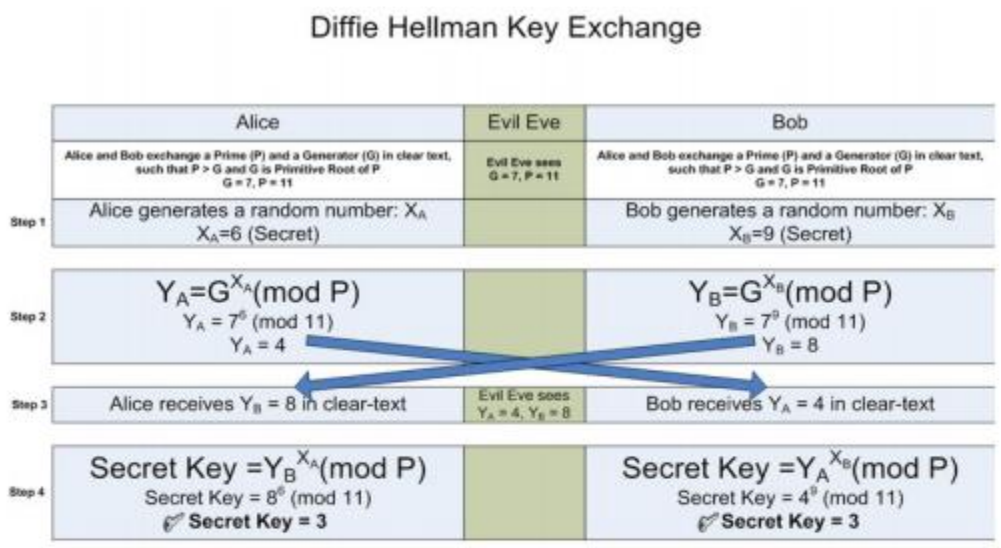
In a columnar transposition, the message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order. Both the width of the rows and the permutation of the columns are usually defined by a keyword. For example, the word ZEBRAS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

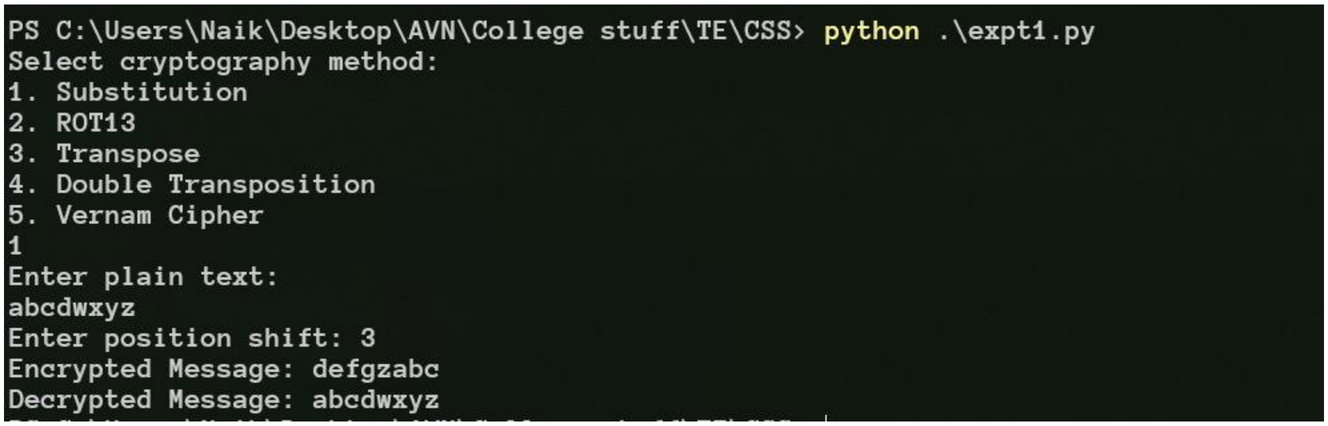
CODE:

```
import random
import numpy as np

print('Select cryptography method:\n1. Substitution\n2. ROT13\n3
. Transpose\n4. Double Transposition\n5. Vernam Cipher')
method = int(input())

print('Enter plain text: ')
text = str(input())


if method == 1:
   shift = int(input(“Enter position shift: ”))
   encrypted = ""
   decrypted = ""
   for i in text:
      if ord(i) >= 65 and ord(i) <= 90:
         encrypted += chr((ord(i) + shift - 65) % 26 + 65)
      elif ord(i) >= 97 and ord(i) <= 122:
         encrypted += chr((ord(i) + shift - 97) % 26 + 97)
   for i in encrypted:
      if ord(i) >= 65 and ord(i) <= 90:
         decrypted += chr((ord(i) - shift - 65) % 26 + 65)
      elif ord(i) >= 97 and ord(i) <= 122:
         decrypted += chr((ord(i) - shift - 97) % 26 + 97)

   print("Encrypted Message: " + encrypted)
   print("Decrypted Message: " + decrypted)

elif method == 2:
   shift = 13
   encrypted = ""
   decrypted = ""
   for i in text:
      if ord(i) >= 65 and ord(i) <= 90:
         encrypted += chr((ord(i) + shift - 65) % 26 + 65)
      elif ord(i) >= 97 and ord(i) <= 122:
         encrypted += chr((ord(i) + shift - 97) % 26 + 97)
   for i in encrypted:
        if ord(i) >= 65 and ord(i) <= 90:
            decrypted += chr((ord(i) - shift - 65) % 26 + 65)
        elif ord(i) >= 97 and ord(i) <= 122:
            decrypted += chr((ord(i) - shift - 97) % 26 + 97)

    print("Encrypted Message: " + encrypted)
    print("Decrypted Message: " + decrypted)

elif method == 3:
    encrypted = ""
    decrypted = ""
    key = input("Enter key: ")
    length_key = len(key)
    order = ""

    for i in range(1, length_key+1):
        order += str(i)
    str1 = key + order + text
    if len(text)%length_key != 0:
        for i in range(length_key - len(text)%length_key):
            str1 += chr(random.randint(97, 122))

    grid = np.array(list(str1)).reshape(len(str1)//length_key, le
ngth_key)
    # Encryption
    grid_sort = grid[:, grid[0].argsort()]
    for j in range(length_key):
        for i in range(2, len(grid_sort)):
            encrypted += grid_sort[i][j]
        encrypted += " "

    # Decryption
    gresort = grid_sort[:, grid_sort[1].argsort()]
    for i in range(2, len(gresort)):
        for j in range(length_key):
            decrypted+=gresort[i][j]

    print("Encrypted Message: " + encrypted)
    print("Decrypted Message: " + decrypted[:len(text)])
```

```
elif method == 4:
    encrypted = ""
    decrypted = ""
    key = input("Enter key: ")
    length_key = len(key)
    order = ""

    for i in range(1, length_key+1):
        order += str(i)

    str1 = key + order + text

    if len(text)%length_key != 0:
        for i in range(length_key - len(text)%length_key):
            str1 += chr(random.randint(97, 122))

    grid = np.array(list(str1)).reshape(len(str1)//length_key, le
ngth_key)


    # First Transpose
    grid = grid[:, grid[0].argsort()]
    for j in range(length_key):
        for i in range(2, len(grid)):
            encrypted += grid[i][j]

    str1 = key + order + encrypted
    grid2 = np.array(list(str1)).reshape(len(str1)//length_key, l
ength_key)
    grid2 = grid2[:, grid2[0].argsort()]
    encrypted2 = ""

    # Second Transpose
    for j in range(length_key):
        for i in range(2, len(grid2)):
            encrypted2 += grid2[i][j]
        encrypted2 += " "


    # Decryption 1
    grid_reverse = grid2[:, grid2[1].argsort()]
    for i in range(2, len(grid_reverse)):
        for j in range(length_key):
            decrypted += grid_reverse[i][j]

    str1 = ""
    for i in range(2):
        for j in range(length_key):
            str1 += grid2[i][j]

    grid = np.array(list(decrypted)).reshape(length_key, len(decr
ypted)//length_key)
    grid = grid.T
    for i in range(len(grid)):
        for j in range(length_key):
            str1 += grid[i][j]

    grid = np.array(list(str1)).reshape(len(str1)//length_key, le
ngth_key)
    grid_reverse2 = grid[:, grid[1].argsort()]

    # Decryption 2
    final_decrypted = ""
    for i in range(2, len(grid_reverse2)):
        for j in range(length_key):
            final_decrypted+=grid_reverse2[i][j]

    print("Encrypted Message: " + encrypted2)
    print("Decrypted Message: " + final_decrypted[:len(text)])

elif method == 5:
    key = ""
    while len(key) != len(text):
        key = input("Enter key same length as text: ")

    encrypted = ""
    decrypted = ""

    for i in range(len(text)):
        encrypted += chr(ord(text[i]) ^ ord(key[i]))
    for i in range(len(text)):
        decrypted += chr(ord(encrypted[i]) ^ ord(key[i]))

    print("Encrypted Message: " + encrypted)
    print("Decrypted Message: " + decrypted)
```

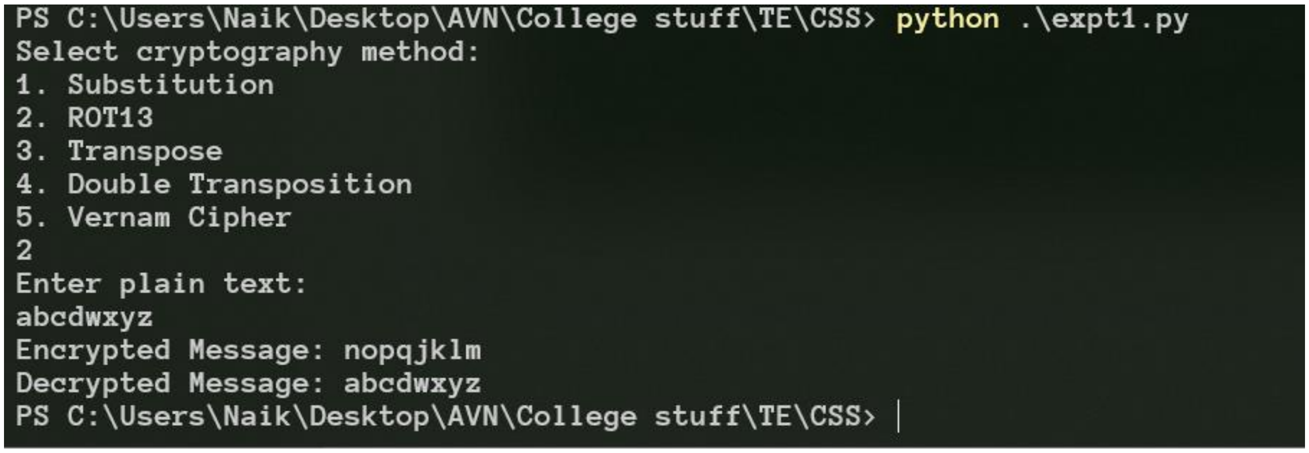
OUTPUT:

Substitution



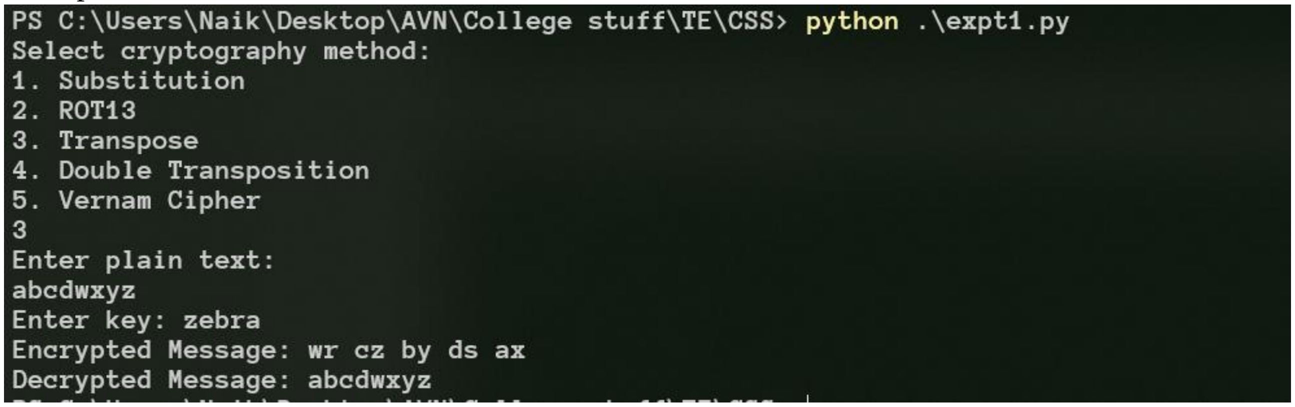
```
PS C:\Users\Naik\Desktop\AVN\College stuff\TE\CSS> python .\expt1.py
Select cryptography method:
1. Substitution
2. ROT13
3. Transpose
4. Double Transposition
5. Vernam Cipher
1
Enter plain text:
abcdwxyz
Enter position shift: 3
Encrypted Message: defgzabc
Decrypted Message: abcdwxyz
```

ROT 13



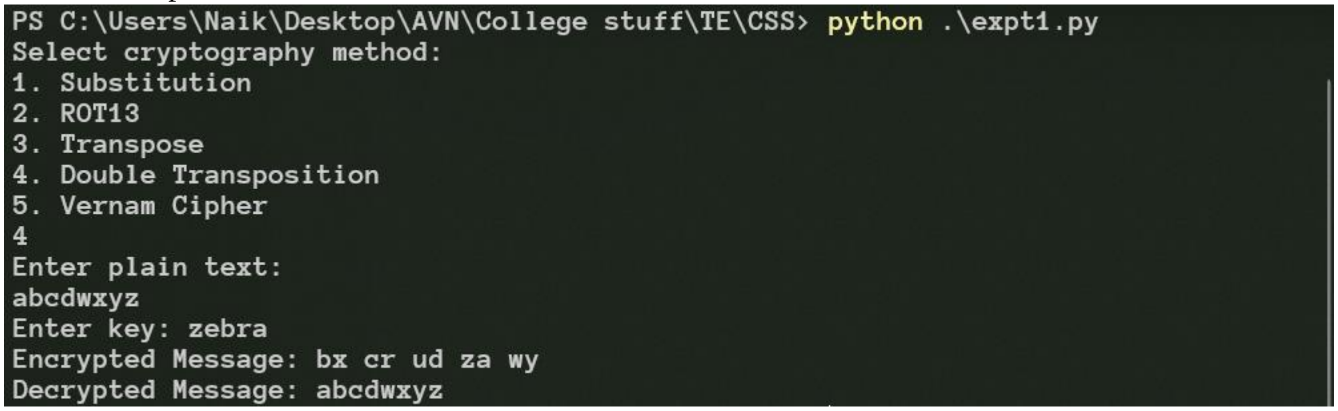
```
PS C:\Users\Naik\Desktop\AVN\College stuff\TE\CSS> python .\expt1.py
Select cryptography method:
1. Substitution
2. ROT13
3. Transpose
4. Double Transposition
5. Vernam Cipher
2
Enter plain text:
abcdwxyz
Encrypted Message: nopqjklm
Decrypted Message: abcdwxyz
PS C:\Users\Naik\Desktop\AVN\College stuff\TE\CSS> |
```

Transposition



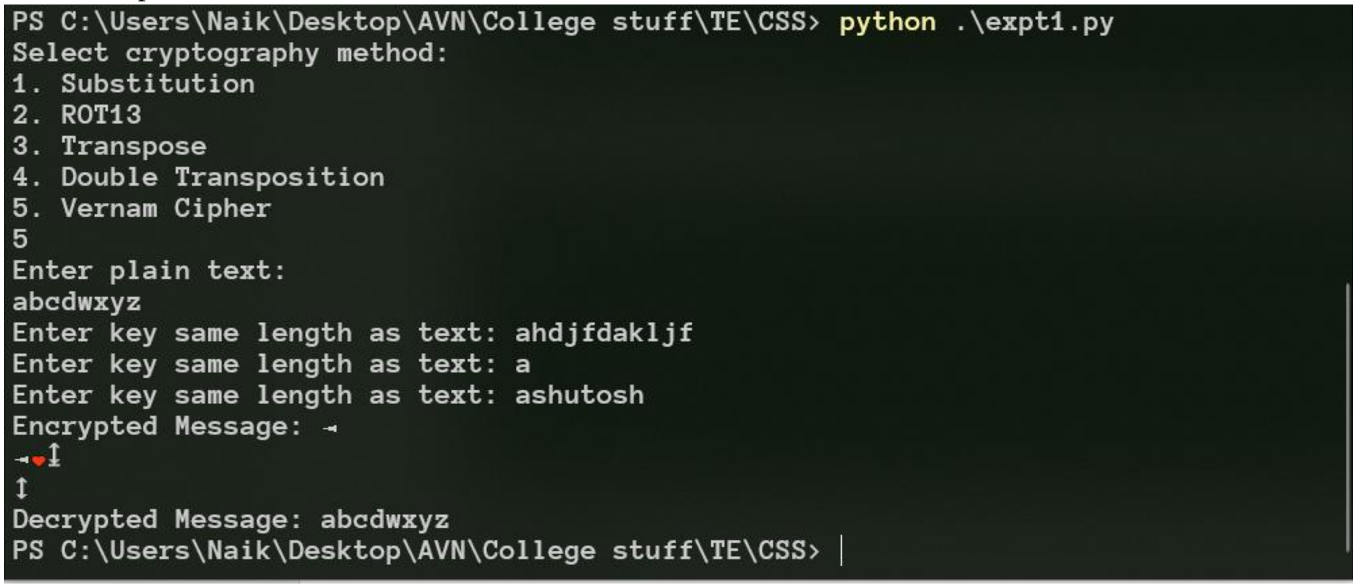
```
PS C:\Users\Naik\Desktop\AVN\College stuff\TE\CSS> python .\expt1.py
Select cryptography method:
1. Substitution
2. ROT13
3. Transpose
4. Double Transposition
5. Vernam Cipher
3
Enter plain text:
abcdwxyz
Enter key: zebra
Encrypted Message: wr cz by ds ax
Decrypted Message: abcdwxyz
```

Double transposition



```
PS C:\Users\Naik\Desktop\AVN\College stuff\TE\CSS> python .\expt1.py
Select cryptography method:
1. Substitution
2. ROT13
3. Transpose
4. Double Transposition
5. Vernam Cipher
4
Enter plain text:
abcdwxyz
Enter key: zebra
Encrypted Message: bx cr ud za wy
Decrypted Message: abcdwxyz
```

Vernam cipher



```
PS C:\Users\Naik\Desktop\AVN\College stuff\TE\CSS> python .\expt1.py
Select cryptography method:
1. Substitution
2. ROT13
3. Transpose
4. Double Transposition
5. Vernam Cipher
5
Enter plain text:
abcdwxyz
Enter key same length as text: ahdjfdakljf
Enter key same length as text: a
Enter key same length as text: ashutosh
Encrypted Message: ⇥
⇥♥↨
↕
Decrypted Message: abcdwxyz
PS C:\Users\Naik\Desktop\AVN\College stuff\TE\CSS> |
```

OBSERVATIONS:

- Caeser cipher is a type of substitution cipher where the characters are shifted.
- It can be broken easily using cryptography where certain patterns/structures are observed in the cipher text.
- ROT-13 is essentially a special type of Caeser cipher wherein the characters are shifted by a position of 13.
- It is the least secure type of cipher since it can be easily decrypted by shifting the characters back by 13.
- Vernam cipher uses the same key for both encrypting and decrypting. The key has to be the same length as the plain text.
- After XOR with the key, the resultant characters may end up being outside the range of ASCII alphabets resulting in generation of random symbols.