NAME: BHARGAVI S. POYEKAR
TE COMPS
BATCH-C
UID: 2018130040

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) 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.

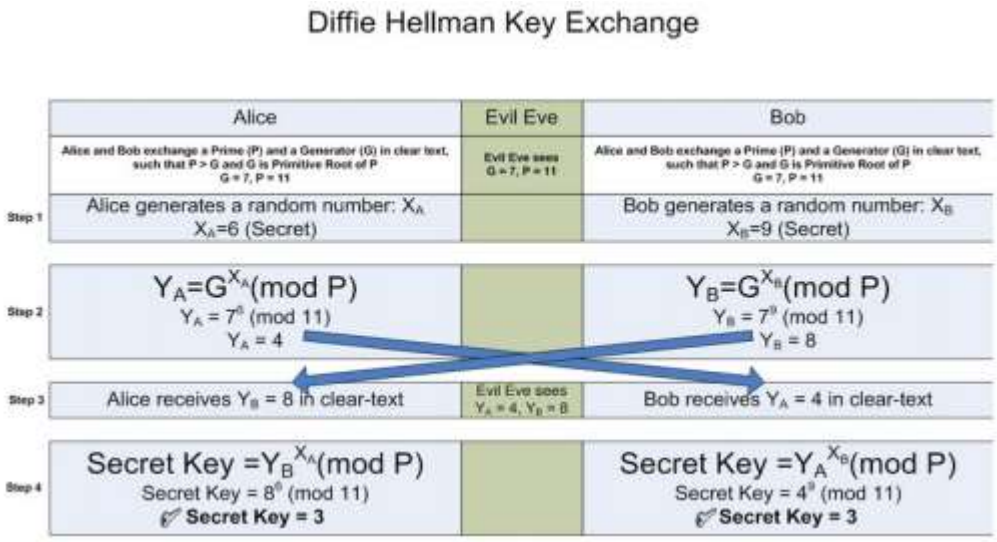
In a columnar transposition, the message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order. Both the width of the rows and the permutation of the columns are usually defined by a keyword. For example, the word ZEBRAS 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.

Double Transposition:
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.

Vernam cipher:
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.

Diffie –Hellman Key exchange algorithm:
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).



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:

```
# input key for substitution method

def input_key():

    key = int(input("Enter the no. of position shift: "))

    return key


# input of plain text

def input_plain():

    plain = input("Enter Plain text to be encrypted:\n")

    return plain


# common function for substitution and ROT 13

def caesar(plain, key):

    plain_list = list(plain)

    encrypt = []
```

```
decrypt = []

# Encryption
for i in plain_list:
    if ord(i) >= 65 and ord(i) <= 90:
        # if character is between A-Z
        encrypt.append(chr(((ord(i)-65+key) % 26)+65))
    elif ord(i) >= 97 and ord(i) <= 122:
        # if character is between a-z
        encrypt.append(chr(((ord(i)-97+key) % 26)+97))
    else:
        # if character is special
        encrypt.append(i)
enc = ''.join(encrypt)
print("Encrypted Message is: "+enc)

# Decryption
for i in encrypt:
    if ord(i) >= 65 and ord(i) <= 90:
        decrypt.append(chr(((ord(i)-65-key) % 26)+65))
    elif ord(i) >= 97 and ord(i) <= 122:
        decrypt.append(chr(((ord(i)-97-key) % 26)+97))
    else:
        decrypt.append(i)
dec = ''.join(decrypt)
```

```
    print("Decrypted Message is: "+dec)


# Transposition
def transpose(plain):
    key = input("Enter cipher key: ")
    l_key = list(key)  # list of key characters
    s_key = sorted(l_key)  # sorted key
    plain_list = list(plain)  # list of plain cipher text characters

    # Encryption
    rem = len(plain) % len(key)
    emp = len(key)-rem   # Finding empty characters at the end in matrix
    for i in range(emp):
        plain_list.append('_')  # replacing empty space at the end with _

    # Convert list into matrix
    matrix = [[] for j in range(len(key))]
    encrypt = []
    for i in range(len(matrix)):
        for j in range(i, len(plain_list), len(key)):
            matrix[i].append(plain_list[j])

    # Rearranging matrix according to the key
    for i in range(len(key)):
        encrypt.append(matrix[l_key.index(s_key[i])])
```

```
# Converting matrix to list
for i in range(len(key)):
    encrypt[i] = ''.join(encrypt[i])

# Converting list to string
enc = ''.join(encrypt)
print("Encrypted Message is: " + enc)

# Decryption
matrix = [[] for j in range(len(key))]
decrypt = []

# Converting list of encrypted message to matrix
for i in range(len(matrix)):
    for j in range(i, len(encrypt), len(key)):
        matrix[i].append(encrypt[j])

# Rearranging matrix back to original form
for i in range(len(key)):
    decrypt.append(matrix[s_key.index(l_key[i])])

# Reconverting encrypted matrix to original message
dec_list = []
for i in range(len(enc)//len(key)):
    for j in range(len(key)):
        dec_list.append(decrypt[j][0][i])
```

```
    # List to string
    dec = ''.join(dec_list)

    # Removing extra _ at the end, added while encrypting
    dec = dec.replace('_', '')
    print("Decrypted Message is: " + dec)

# Double transposition
def double_transpose(plain):
    key = input("Enter cipher key: ")
    l_key = list(key)
    s_key = sorted(l_key)
    plain_list = list(plain)

    # Encryption
    # Using transposition method first time
    rem = len(plain) % len(key)
    emp = len(key)-rem
    for i in range(emp):
        plain_list.append('_')

    matrix = [[] for j in range(len(key))]
    encrypt = []
    for i in range(len(matrix)):
        for j in range(i, len(plain_list), len(key)):
            matrix[i].append(plain_list[j])
```

```
    for i in range(len(key)):
        encrypt.append(matrix[l_key.index(s_key[i])])

    for i in range(len(key)):
        encrypt[i] = ''.join(encrypt[i])
    enc = ''.join(encrypt)
    # First encryption done
    enc_list = list(enc)

    # Again using transposition
    matrix = [[] for j in range(len(key))]
    encrypt = []
    for i in range(len(matrix)):
        for j in range(i, len(enc_list), len(key)):
            matrix[i].append(enc_list[j])

    for i in range(len(key)):
        encrypt.append(matrix[l_key.index(s_key[i])])

    for i in range(len(key)):
        encrypt[i] = ''.join(encrypt[i])

    enc = ''.join(encrypt)
    # Message is encrypted
    print("Encrypted Message is: " + enc)
```

```
# Decryption
matrix = [[] for j in range(len(key))]
decrypt = []
for i in range(len(matrix)):
    for j in range(i, len(encrypt), len(key)):
        matrix[i].append(encrypt[j])

for i in range(len(key)):
    decrypt.append(matrix[s_key.index(l_key[i])])

dec_list = []
for i in range(len(enc)//len(key)):
    for j in range(len(key)):
        dec_list.append(decrypt[j][0][i])
dec = ''.join(dec_list)

# After first decryption intermediate message is recovered.

# Converting the list to a similar format in the transposition method
ldec = []
q = len(dec)//len(key)
while dec_list:
    ldec.append(dec_list[:q])
    dec_list = dec_list[q:]
```

```
    str_dec = []
    for i in ldec:
        str_dec.append(''.join(i))

    # Again decrypting to get the original message
    matrix = [[] for j in range(len(key))]
    doub_decrypt = []
    for i in range(len(matrix)):
        for j in range(i, len(str_dec), len(key)):
            matrix[i].append(str_dec[j])

    for i in range(len(key)):
        doub_decrypt.append(matrix[s_key.index(l_key[i])])

    dec_list = []
    for i in range(len(dec)//len(key)):
        for j in range(len(key)):
            dec_list.append(doub_decrypt[j][0][i])
    dec = ''.join(dec_list)
    dec = dec.replace('_', '')
    print("Decrypted Message is: " + dec)

# Vernam Cipher
def vernam(plain):
    while(True):
        key = input("Input Key:\n")
```

```
        if(len(key) == len(plain)):
            break
        else:
            print("Invalid key!!! Length of key and plain text shoud be equal")
    encrypt = []
    decrypt = []

    # Encryption
    for i in range(len(plain)):
        encrypt.append(chr(((ord(plain[i])-65 ^ ord(key[i])-65) % 26)+65))
    enc = ''.join(encrypt)
    print("Encrypted Message is: "+enc)
    #Decryption
    for i in range(len(encrypt)):
        decrypt.append(chr(((ord(encrypt[i])-65 ^ ord(key[i])-65) % 26)+65))
    dec = ''.join(decrypt)
    print("Decrypted Message is: "+dec)

# Switch Case for Cryptography methods
class Switcher(object):
    def indirect(self, i):
        # to call the required method of cryptography
        method_name = 'choice_'+str(i)
        method = getattr(self, method_name, lambda: 'Invalid option')
        return method()
```

```
    def choice_1(self):
        print("Your Choice: Substitution method")
        plain = input_plain()
        key = input_key()
        caesar(plain, key)

    def choice_2(self):
        print("Your Choice: ROT 13")
        plain = input_plain()
        caesar(plain, 13)  # passing 13 as key to caesar method

    def choice_3(self):
        print("Your Choice: Transposition Cipher")
        plain = input_plain()
        transpose(plain)

    def choice_4(self):
        print("Your Choice: Double Transposition Cipher")
        plain = input_plain()
        double_transpose(plain)

    def choice_5(self):
        print("Your Choice: Vernam Cipher")
        plain = input_plain()
        vernam(plain)
```

```
s = Switcher()  # Creating object of Switcher Class
ch = int(input("Select the cryptography method:\n(1)Substitution.\n(2)ROT
    13.\n(3)Transposition.\n(4)Double Transposition.\n(5)Vernam Cipher.\n"))
s.indirect(ch)
```

OUTPUT:

1. Substitution Method:

```
C:\Users\bharg\Desktop\python>python crypto.py
Select the cryptography method:
(1)Substitution.
(2)ROT 13.
(3)Transposition.
(4)Double Transposition.
(5)Vernam Cipher.
1
Your Choice: Substitution method
Enter Plain text to be encrypted:
I am Bhargavi
Enter the no. of position shift: 10
Encrypted Message is: S kw Lrkbqkfs
Decrypted Message is: I am Bhargavi
```

2. ROT 13:

```
C:\Users\bharg\Desktop\python>python crypto.py
Select the cryptography method:
(1)Substitution.
(2)ROT 13.
(3)Transposition.
(4)Double Transposition.
(5)Vernam Cipher.
2
Your Choice: ROT 13
Enter Plain text to be encrypted:
Hello How are You?
Encrypted Message is: Uryyb Ubj ner Lbh?
Decrypted Message is: Hello How are You?
```

3. Transposition Cipher:

```
C:\Users\bharg\Desktop\python>python crypto.py
Select the cryptography method:
(1)Substitution.
(2)ROT 13.
(3)Transposition.
(4)Double Transposition.
(5)Vernam Cipher.
3
Your Choice: Transposition Cipher
Enter Plain text to be encrypted:
Meet me tonight, I am Waiting
Enter cipher key: Black
Encrypted Message is: Mmn,mte gIWntth ag otai_eei  i
Decrypted Message is: Meet me tonight, I am Waiting
```

4. Double Transposition:

```
C:\Users\bharg\Desktop\python>python crypto.py
Select the cryptography method:
(1)Substitution.
(2)ROT 13.
(3)Transposition.
(4)Double Transposition.
(5)Vernam Cipher.
4
Your Choice: Double Transposition Cipher
Enter Plain text to be encrypted:
Css is interesting
Enter cipher key: subject
Encrypted Message is: n s_Cnegsse_srtiiit_
Decrypted Message is: Css is interesting
```

5. Vernam Cipher:

```
C:\Users\bharg\Desktop\python>python crypto.py
Select the cryptography method:
(1)Substitution.
(2)ROT 13.
(3)Transposition.
(4)Double Transposition.
(5)Vernam Cipher.
5
Your Choice: Vernam Cipher
Enter Plain text to be encrypted:
BHARGAVI
Input Key:
CREATIVE
Encrypted Message is: DWERVIAM
Decrypted Message is: BHARGAVI
```

Observations/ Interesting Facts:

1. ROT 13 is similar to substitution method, only difference is that the position shift is fixed in ROT 13 that is 13.
2. Using ROT cipher encryption twice gives us the original message, so decryption can be performed by just performing the encryption once more. (because there are 26 letters and twice of 13 is 26)
3. Double transposition was used during World War I and II and it was also considered as one of the most complicated ciphers because the double transposition makes it difficult to guess the original message without the key.
4. In Verman Cipher, same code of encryption is applied in decryption, in encryption the XOR is performed between key and original plain text and in decryption XOR is performed between key and encrypted message.

CONCLUSION:

1. I implemented all the required methods of cryptography in python.
2. I used a common function with different inputs for substitution cipher and ROT 13 as they are similar.
3. I performed columnar transposition in both transposition and double transposition which makes it difficult to decipher without a key.