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 : SHREYAS PATEL

ROLL NO : 42

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".

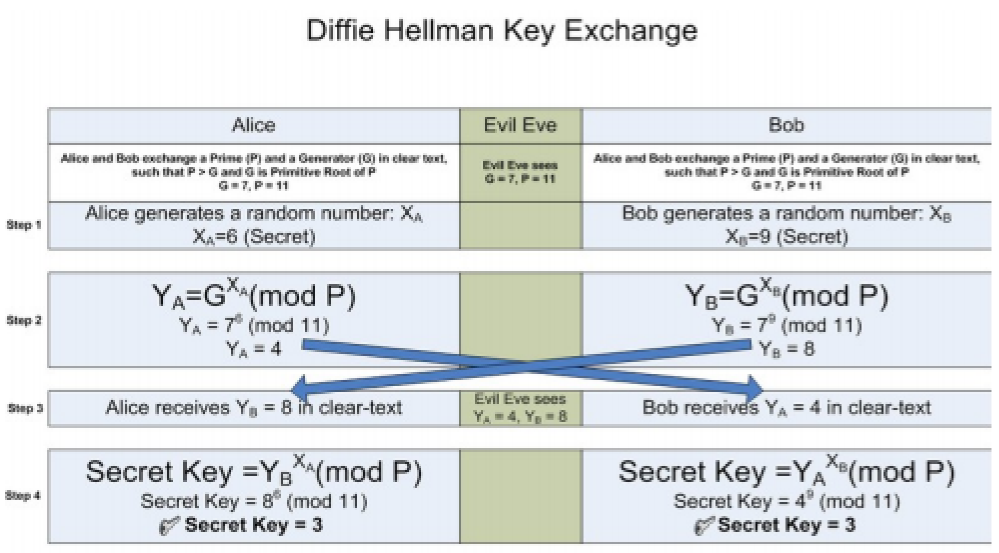
In a regular columnar transposition cipher, any spare spaces are filled with nulls; in an irregular columnar transposition cipher, the spaces are left blank. Finally, the message is read off in columns, in the order specified by the keyword.

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).


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
$Y_A=G^{X_A}$(mod P) $Y_A = 7^6$ (mod 11) $Y_A = 4$
$Y_B=G^{X_B}$(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 =$Y_B^{X_A}$(mod P) Secret Key = 8^6 (mod 11) Secret Key = 3
Secret Key =$Y_A^{X_B}$(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:

CODE :

```
import math
def double_transpose(string):
    key = str(raw_input("Enter cipher key: "))
    l_key = list(key)
    s_key = sorted(l_key)
    string_list = list(string)

    rem = len(string) % len(key)
    emp = len(key)-rem
    for i in range(emp):
        string_list.append('_')

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

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

enc_list = list(enc)

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)

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)
```

```
    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)
def encrypt3(msg):
    key = "HACK"
    cipher = ""
    k_indx = 0
    msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))
    col = len(key)
    row = int(math.ceil(msg_len / col))
    fill_null = int((row * col) - msg_len)
    msg_lst.extend('_' * fill_null)
    matrix = [msg_lst[i: i + col]
```

```
             for i in range(0, len(msg_lst), col)]
    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])
        cipher += ''.join([row[curr_idx]
                          for row in matrix])
        k_indx += 1
    return cipher

def decrypt3(cipher):
    key = "HACK"
    msg = ""
    k_indx = 0
    msg_indx = 0
    msg_len = float(len(cipher))
    msg_lst = list(cipher)
    col = len(key)
    row = int(math.ceil(msg_len / col))
    key_lst = sorted(list(key))
    dec_cipher = []
    for _ in range(row):
        dec_cipher += [[None] * col]
    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])

        for j in range(row):
            dec_cipher[j][curr_idx] = msg_lst[msg_indx]
            msg_indx += 1
        k_indx += 1
    try:
        msg = ''.join(sum(dec_cipher, []))
    except TypeError:
        raise TypeError("This program cannot",
                        "handle repeating words.")

    null_count = msg.count('_')

    if null_count > 0:
        return msg[: -null_count]
    return msg
def encrypt5(string, key):
      i = 0
      string1 = []
```

```
        while i < len(string):
            k = ord(string[i]) + ord(key[i]) - 97
            if k > 122:
                    k = k - 26
            string1.append(chr(k))
            i = i + 1
        str1 = ""
        for j in string1:
            str1 = str1 + j
        print("The encrypted message is :" + str1)
        return string1
def decrypt5(string3, key):
        i = 0
        string2 = []
        while i < len(string3):
            k = ord(string3[i]) - ord(key[i]) + 97
            if k < 97:
                    k = k + 26
            string2.append(chr(k))
            i = i + 1
        str2 = ""
        for j in string2:
            str2 = str2 + j
        print("The decrypted message is :" + str2)
def diffie_hiemann(g,n,x,y):
        a = pow(n,x,g)
        b = pow(n,y,g)
        temp = a
        a = b
        b = temp
        k1 = pow(a,x,g)
        k2 = pow(b,y,g)
        print("The keys are " + str(k1) + " and " + str(k2))
def encrypt2(string):
        i = 0
        string1 = []
        while i < len(string):
            k = ord(string[i]) + 13
            if k > 122:
                    k = k - 26
            string1.append(chr(k))
            i = i + 1
```

```
    str1 = ""
    for j in string1:
        str1 = str1 + j
    print("The encrypted message is :" + str1)
    return string1
def decrypt2(string3):
    i = 0
    string2 = []
    while i < len(string3):
        k = ord(string3[i]) - 13
        if k < 97:
                k = k + 26
        string2.append(chr(k))
        i = i + 1
    str2 = ""
    for j in string2:
        str2 = str2 + j
    print("The decrypted message is :" + str2)
def encrypt1(string, n):
    i = 0
    string1 = []
    while i < len(string):
        k = ord(string[i]) + n
        if k > 122:
                k = k - 26
        string1.append(chr(k))
        i = i + 1
    str1 = ""
    for j in string1:
        str1 = str1 + j
    print("The encrypted message is :" + str1)
    return string1
def decrypt1(string3, n):
    i = 0
    string2 = []
    while i < len(string3):
        k = ord(string3[i]) - n
        if k < 97:
                k = k + 26
        string2.append(chr(k))
        i = i + 1
    str2 = ""
```

```
        for j in string2:
            str2 = str2 + j
        print("The decrypted message is :" + str2)
while 1 == 1:
        print("Enter your choice of method to be used:")
        a = int(raw_input())
        if a == 1:
            print("Enter the string: ")
            string = list(raw_input())
            print("Substitution Method:")
            print("Enter the no.of positions to be shifted: ")
            n = int(raw_input())
            string3 = encrypt1(string, n)
            decrypt1(string3, n)
        if a == 2:
            print("Enter the string: ")
            string = list(raw_input())
            print("ROT 13 Method:")
            string3 = encrypt2(string)
            decrypt2(string3)
        if a == 3:
            print("Enter the string: ")
            string = str(raw_input())
            cipher = encrypt3(string)
            print("Encrypted Message: {}".
                 format(cipher))
            print("Decryped Message: {}".
         format(decrypt3(cipher)))
        if a == 4:
            print("Double Transposition Method:")
            print("Enter the string")
            string = str(raw_input())
            double_transpose(string)
        if a == 5:
            print("Enter the string: ")
            string = list(raw_input())
            print("Vernam Cipher Method:")
            print("Enter the key:")
            key = list(raw_input())
            string3 = encrypt5(string, key)
            decrypt5(string3, key)
        if a == 6:
            print("Diffie-Hiemann Method:")
            print("Enter g and n:")
            g = int(raw_input())
            n = int(raw_input())
            print("Enter x and y:")
            x = int(raw_input())
            y = int(raw_input())
            diffie_hiemann(g,n,x,y)
        if a == 7:
            break
```

OUTPUT :

```
Enter your choice of method to be used:
1
Enter the string:
substitution
Substitution Method:
Enter the no.of positions to be shifted:
6
The encrypted message is :yahyzozazout
The decrypted message is :substitution
Enter your choice of method to be used:
2
Enter the string:
rotation
ROT 13 Method:
The encrypted message is :ebgngvba
The decrypted message is :rotation
Enter your choice of method to be used:
3
Enter the string:
hello shreyas
Encrypted Message: e e_lsy_horslha_
Decryped Message: hello shreyas
Enter your choice of method to be used:
5
Enter the string:
vernamcipher
Vernam Cipher Method:
Enter the key:
gdhsjdbhfbej
```

```
The encrypted message is :bhyfjpdpuiia
The decrypted message is :vernamcipher
Enter your choice of method to be used:
6
Diffie-Hiemann Method:
Enter g and n:
23
45
Enter x and y:
3
8
The keys are 1 and 1
Enter your choice of method to be used:
4
Double Transposition Method:
Enter the string
shreyas
Enter cipher key: hjgretu
Encrypted Message is: s_r_y__e_a_sh_
Decrypted Message is: shreyas
Enter your choice of method to be used:
7
```

OBSERVATIONS :

1. Substitution Cipher takes the string, asks the user how many places to be shifted and then encrypts the string accordingly and decrypts it in opposite manner.
2. ROT13 is same as Substitution method, just the number of places is fixed – 13
3. Transposition Cipher divides the string into rows and columns and encrypts it according to the preference of the key provided by the user. The letter which comes first, the letters in that column will be written first. Decryption is exactly the opposite
4. Double transposition is almost the same as transposition but done two times
5. Vernam Cipher takes the string and Xors its letters with the ones in the key which should be of equal length as that of the string. The resultant string is the encrypted one. It is then decrypted by doing the exact opposite

CONCLUSION : Through this experiment I came to know the use of these algorithms in cryptography