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

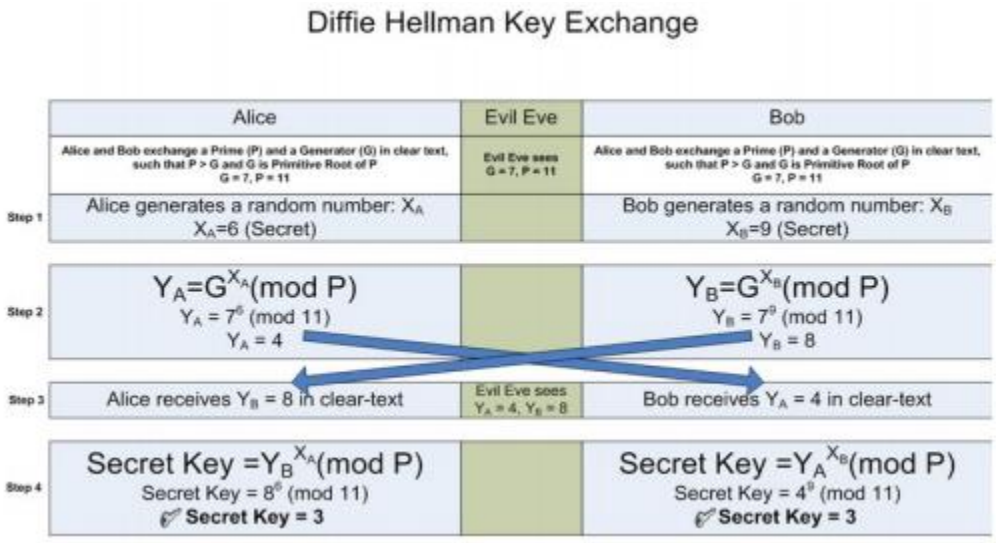
example, the word ZEBRAS is of length 6 (so the rows are of length 6), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be "6 3 2 4 1 5".

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.

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:

```
import math
import random
import string
import numpy as np
from sympy import isprime, is_primitive_root

def transposition_cipher(message, keyword, flag):
    '''Set flag = True for encryption and False for decryption '''
    if flag == True:
        matrix = []
        temp = []
        j = 0

        for letter in message:
            if j < len(keyword):
                if (ord(letter) >= 65 and ord(letter) <= 90 ) or (ord(letter) >=
97 and ord(letter) <= 122):
                    temp.append(letter)
                    j += 1
            else:
                matrix.append(temp)
                temp = [letter]
                j = 1
        else:
            for i in range(len(keyword) - len(temp)):
                temp += random.choice(string.ascii_letters)
            matrix.append(temp)

        matrix.insert(0, list(keyword))

        matrix = list(zip(*sorted(zip(*matrix))))
        matrix = matrix[1:]

        encrypted_message = ""
        count = 0

        for i in range(len(keyword)):
            for j in range(len(matrix)):
                if count < 5:
                    encrypted_message += matrix[j][i]
                    count += 1
                else:
                    encrypted_message += " " + matrix[j][i]
                    count = 1
```

```
        return encrypted_message
    else:
        encrypted_message = "".join(message.split()) # Removing spaces from the m
essage
        encrypted_message_list = list(encrypted_message)

        matrix = []
        temp = []
        j = 0
        column_length = len(encrypted_message) // len(keyword)

        for letter in encrypted_message_list:
            if j < column_length:
                temp.append(letter)
                j += 1
            else:
                # print(temp)
                matrix.append(temp)
                temp = [letter]
                j = 1
        else:
            matrix.append(temp)

        numpy_matrix = np.array(matrix)
        numpy_matrix = np.transpose(numpy_matrix)

        sorted_key = list(keyword)
        sorted_key.sort()

        numpy_matrix = np.vstack((sorted_key, numpy_matrix))
        numpy_matrix = list(zip(*sorted(zip(*numpy_matrix), key = lambda i: list(
key).index(i[0]))))
        numpy_matrix = numpy_matrix[1:]

        decrypted_message = []
        for sublist in numpy_matrix:
            decrypted_message += sublist

        decrypted_message = ''.join(decrypted_message)

        return decrypted_message

def caeser_cipher(message, shift, flag):
    '''Set flag = True for encryption and False for decryption '''
    output = ""
```

```
    if flag == True:
        for letter in message:
            if ord(letter) >= 65 and ord(letter) <= 90:
                encrypted_letter = chr(((ord(letter) - 65 + shift) % 26) + 65)
                output += encrypted_letter
            elif ord(letter) >= 97 and ord(letter) <= 122:
                encrypted_letter = chr(((ord(letter) - 97 + shift) % 26) + 97)
                output += encrypted_letter
    else:
        for letter in message:
            if ord(letter) >= 65 and ord(letter) <= 90:
                decrypted_letter = chr(((ord(letter) - 65 - shift) % 26) + 65)
                output += decrypted_letter
            elif ord(letter) >= 97 and ord(letter) <= 122:
                decrypted_letter = chr(((ord(letter) - 97 - shift) % 26) + 97)
                output += decrypted_letter

    return output

def vernam_cipher(message, keyword):
    output = ""

    for letter1, letter2 in zip(message, keyword):
        xored_character = chr(ord(letter1) ^ ord(letter2))
        output += xored_character

    return output

def diffie_hellman(X_A, X_B, P, G):
    Y_A = pow(G, X_A, P) # Generated by Alice
    Y_B = pow(G, X_B, P) # Generated by Bob

    K_A = pow(Y_B, X_A, P) # Shared Key Obtained by Alice
    K_B = pow(Y_A, X_B, P) # Shared Key Obtained by Bob

    return K_A, K_B

if __name__ == "__main__":
    while True:
        print("Select an encryption algorithm: ")
        print("a. Substitution Cipher")
        print("b. ROT 13")
        print("c. Columnar Transposition Cipher")
        print("d. Double Columnar Transposition Cipher")
        print("e. Vernam Cipher")
        print("f. Diffie Hellman Key Exchange")
```

```
        print("g. Exit")

        choice = input()
        if choice == "a" or choice == "A":
            plain_text = input("\nEnter the plain text: ")
            shift = int(input("Enter the shift: "))
            encrypted_message = caeser_cipher(plain_text, shift, True)
            decrypted_message = caeser_cipher(encrypted_message, shift, False)

            print("Encrypted Message: " + encrypted_message)
            print("Decrypted Message: " + decrypted_message + "\n")

        elif choice == "b" or choice == "B":
            plain_text = input("\nEnter the plain text: ")
            shift = 13
            encrypted_message = caeser_cipher(plain_text, shift, True)
            decrypted_message = caeser_cipher(encrypted_message, shift, False)

            print("Encrypted Message: " + encrypted_message)
            print("Decrypted Message: " + decrypted_message + "\n")

        elif choice == "c" or choice == "C":
            plain_text = list(input("\nEnter the plain text: "))
            key = input("Enter a keyword: ")

            encrypted_message = transposition_cipher(plain_text, key, True)
            decrypted_message = transposition_cipher(encrypted_message, key, Fals
e)

            print("Encrypted message: " + encrypted_message)
            print("Decrypted message: " + decrypted_message + "\n")

        elif choice == "d" or choice == "D":
            plain_text = list(input("\nEnter the plain text: "))
            key = input("Enter a keyword: ")

            encrypted_message = transposition_cipher(plain_text, key, True)
            encrypted_message = transposition_cipher(encrypted_message, key, True
)
            decrypted_message = transposition_cipher(encrypted_message, key, Fals
e)
            decrypted_message = transposition_cipher(decrypted_message, key, Fals
e)

            print("Encrypted message: " + encrypted_message)
            print("Decrypted message: " + decrypted_message + "\n")
```

Traditional Crypto Methods and Key exchange/PV

```
        elif choice == "e" or choice == "e":

            plain_text = input("\nEnter the plain text: ")
            key = input("Enter a keyword of same length: ")

            while len(plain_text) != len(key):
                print("Plain text and Keyword are not of same length. Please give
 the input again.")
                plain_text = input("\nEnter the plain text: ")
                key = input("Enter a keyword of same length: ")

            encrypted_message = vernam_cipher(plain_text, key)
            decrypted_message = vernam_cipher(encrypted_message, key)

            print("Encrypted Message: " + encrypted_message)
            print("Decrypted Message: " + decrypted_message + "\n")

        elif choice == "f" or choice == "F":
            X_A = int(input("\nEnter a random secret number for Alice: "))
            X_B = int(input("Enter a random secret number for Bob: "))

            while True:
                P = int(input("Enter a large prime number, P: "))
                G = int(input("Enter a generator, G, such that G is a primitive r
oot mod P: "))

                if not isprime(P):
                    print("P should be a prime number")
                elif P <= G:
                    print("P should be greater than G")
                elif not is_primitive_root(G, P):
                    print("G is not a primitive root mod P")
                else:
                    break

            K_A, K_B = diffie_hellman(X_A, X_B, P, G)

            print("Key generated by Alice: " + str(K_A))
            print("Key generated by Bob: " + str(K_B) + "\n")

        elif choice == "g" or choice == "G":
            break
        else:
            print("\nEnter valid choice.\n")
```

Output:

```
C:\Users\Rohit Pai\Desktop\CSS\CSS Lab\Experiment 1>python Ciphers.py
Select an encryption algorithm:
a. Substitution Cipher
b. ROT 13
c. Columnar Transposition Cipher
d. Double Columnar Transposition Cipher
e. Vernam Cipher
f. Diffie Hellman Key Exchange
g. Exit
```

Fig. 1. – Choices offered to the user

```
C:\Users\Rohit Pai\Desktop\CSS\CSS Lab\Experiment 1>python Ciphers.py
Select an encryption algorithm:
a. Substitution Cipher
b. ROT 13
c. Columnar Transposition Cipher
d. Double Columnar Transposition Cipher
e. Vernam Cipher
f. Diffie Hellman Key Exchange
g. Exit
fg

Enter valid choice.
```

Fig. 2. – Invalid Choice

```
C:\Users\Rohit Pai\Desktop\CSS\CSS Lab\Experiment 1>python Ciphers.py
Select an encryption algorithm:
a. Substitution Cipher
b. ROT 13
c. Columnar Transposition Cipher
d. Double Columnar Transposition Cipher
e. Vernam Cipher
f. Diffie Hellman Key Exchange
g. Exit
a

Enter the plain text: WE ARE DISCOVERED. FLEE AT ONCE.
Enter the shift: 10
Encrypted Message: GOKBONSCMYFOBONPVOOKDYXMO
Decrypted Message: WEAREDISCOVEREDFLEEATONCE
```

Fig. 3. – Substitution Cipher

```
C:\Users\Rohit Pai\Desktop\CSS\CSS Lab\Experiment 1>python Ciphers.py
Select an encryption algorithm:
a. Substitution Cipher
b. ROT 13
c. Columnar Transposition Cipher
d. Double Columnar Transposition Cipher
e. Vernam Cipher
f. Diffie Hellman Key Exchange
g. Exit
b

Enter the plain text: WE ARE DISCOVERED. FLEE AT ONCE.
Encrypted Message: JRNERQVFPBIRERQSYRRNGBAPR
Decrypted Message: WEAREDISCOVEREDFLEEATONCE
```

Fig. 4. – ROT13 Cipher

```
C:\Users\Rohit Pai\Desktop\CSS\CSS Lab\Experiment 1>python Ciphers.py
Select an encryption algorithm:
a. Substitution Cipher
b. ROT 13
c. Columnar Transposition Cipher
d. Double Columnar Transposition Cipher
e. Vernam Cipher
f. Diffie Hellman Key Exchange
g. Exit
c

Enter the plain text: WE ARE DISCOVERED. FLEE AT ONCE.
Enter a keyword: ZEBRAS
Encrypted message: EVLNn ACDTH ESEAQ ROFOF DEECB WIREE
Decrypted message: WEAREDISCOVEREDFLEEATONCEQHFnB
```

Fig. 5. – Transposition Cipher

```
C:\Users\Rohit Pai\Desktop\CSS\CSS Lab\Experiment 1>python Ciphers.py
Select an encryption algorithm:
a. Substitution Cipher
b. ROT 13
c. Columnar Transposition Cipher
d. Double Columnar Transposition Cipher
e. Vernam Cipher
f. Diffie Hellman Key Exchange
g. Exit
d

Enter the plain text: WE ARE DISCOVERED. FLEE AT ONCE.
Enter a keyword: ZEBRAS
Encrypted message: VEOEE LTODI VDAWW NWRER ASFCE ECEOK
Decrypted message: WEAREDISCOVEREDFLEEATONCEOWWVK
```

Fig. 6. – Double Transposition Cipher

```
C:\Users\Rohit Pai\Desktop\CSS\CSS Lab\Experiment 1>python Ciphers.py
Select an encryption algorithm:
a. Substitution Cipher
b. ROT 13
c. Columnar Transposition Cipher
d. Double Columnar Transposition Cipher
e. Vernam Cipher
f. Diffie Hellman Key Exchange
g. Exit
e

Enter the plain text: Rohit
Enter a keyword of same length: Hauib
Encrypted Message: →♫↔▬
Decrypted Message: Rohit
```

Fig. 7. – Vernam Cipher

```
C:\Users\Rohit Pai\Desktop\CSS\CSS Lab\Experiment 1>python Ciphers.py
Select an encryption algorithm:
a. Substitution Cipher
b. ROT 13
c. Columnar Transposition Cipher
d. Double Columnar Transposition Cipher
e. Vernam Cipher
f. Diffie Hellman Key Exchange
g. Exit
f

Enter a random secret number for Alice: 6
Enter a random secret number for Bob: 15
Enter a large prime number, P: 23
Enter a generator, G, such that G is a primitive root mod P: 5
Key generated by Alice: 2
Key generated by Bob: 2
```

Fig. 8 – Diffie Hellman Key Exchange

Observations:

1) Encryption algorithms like columnar transposition are easier to encrypt and decrypt by hand than using software. (This is not the case for very large plain text and key where the effort is worth the time saved).

2) Caesar cipher and ROT-13 are simple but weak forms of encryption. They can be easily broken by brute force.

3) Single transposition cipher is a better encryption method than substitution cipher but can be broken by guessing the column length and looking for possible anagrams. Double transposition cipher was used to overcome this drawback and is secure as long as the keys are not repeated.

4) Surprisingly, Vernam Cipher or One-Time Pad, one of the simplest methods of encryption, is perfectly secure as long as the key is truly random.

5) Diffie Hellman Key Exchange is a fascinating algorithm which uses the fact that the discrete logarithm problem does not have an efficient solution to generate shared keys between 2 parties using an insecure channel.

Conclusion:

I learnt the basics of cryptography and the traditional cryptographic algorithms. I also learnt the implementation of these algorithms in Python.