Experiment 1: Traditional Crypto Methods and Key Exchange

* 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:

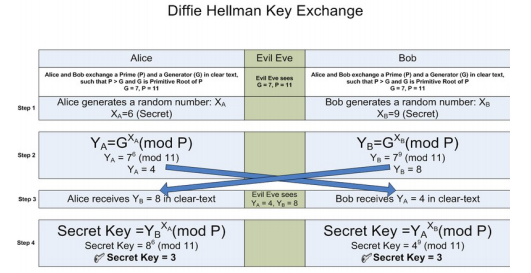
A single columnar transposition could be attacked by guessing possible column lengths, writing the message out in its columns (but in the wrong order, as the key is not yet known), and then looking for possible anagrams. Thus to make it stronger, a double transposition was often used. This is simply 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).



* 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
   1. Substitution
      1. Your choice
      2. Enter Plain text to be encrypted
      3. Enter the no. of Position shift
      4. Encrypted Message
      5. Decrypted Message
   2. ROT 13
      1. Your choice
      2. Enter Plain text to be encrypted
      3. Encrypted Message
      4. Decrypted Message
   3. Transpose
      1. Your choice
      2. Enter Plain text to be encrypted
      3. Encrypted Message
      4. Decrypted Message
   4. Double Transposition
      1. Your choice
      2. Enter Plain text to be encrypted
      3. Encrypted Message
      4. Decrypted Message
   5. Vernam Cipher
      1. Your choice
      2. Enter Plain text to be encrypted
      3. Input Key
      4. Encrypted Message
      5. Decrypted Message

CODE:

import math

def encrypt(msg, shift):

encrypted\_msg = []

for char in msg:

if char.isupper():

encrypted\_msg.append(chr((ord(char)+shift-65)%26+65))

elif char.islower():

encrypted\_msg.append(chr((ord(char)+shift-97)%26+97))

encrypted\_msg = "".join(encrypted\_msg)

return encrypted\_msg

def decrypt(msg, shift):

decrypted\_msg = []

for char in msg:

if char.isupper():

decrypted\_msg.append(chr((ord(char)-shift-65)%26+65))

elif char.islower():

decrypted\_msg.append(chr((ord(char)-shift-97)%26+97))

decrypted\_msg = "".join(decrypted\_msg)

return decrypted\_msg

def transpose\_encrypt(msg, key):

cipher = ""

k\_index = 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\_index])

cipher += ''.join([row[curr\_idx]

for row in matrix])

k\_index += 1

return cipher

def transpose\_decrypt(cipher, key):

msg = ""

k\_index = 0

msg\_index = 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\_index])

for j in range(row):

dec\_cipher[j][curr\_idx] = msg\_lst[msg\_index]

msg\_index += 1

k\_index += 1

try:

msg = ''.join(sum(dec\_cipher, []))

except TypeError:

raise TypeError("This program cannot",

"handle repeating words.")

return msg

def VC(text, key):

output = ""

ptr = 0

for char in text:

output = output + chr(((ord(char) - 65) ^ (ord(key[ptr]) - 65)) + 65)

ptr = ptr + 1

if ptr == len(key):

ptr = 0

return output

def DiffieHellman(g, n, a, b):

# when person A sends secret key to the insecure public channel

x = pow(g, a, n)

# when person A sends secret key to the insecure public channel

y = pow(g, b, n)

# person A receives secret key y

ka = pow(y, a, n)

# person B receives secret key x

kb = pow(x, b, n)

return ka, kb

n = int(input("Select the Cryptography Method\n1. Substitution \n2. ROT13\n3. Transpose\n4. Double Transpose\n5. Vernam Cipher\n6. Diffie Hellman\n=>"))

if n == 1:

msg = input("Message to be encrypted: ")

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

encrypted\_msg = encrypt(msg, shift)

print("Encrypted message is :", encrypted\_msg)

decrypted\_msg = decrypt(encrypted\_msg, shift)

print("Decrypted msg : ", decrypted\_msg)

elif n == 2:

msg = input("Message to be encrypted: ")

shift = 13

encrypted\_msg = encrypt(msg, shift)

print("Encrypted message is :", encrypted\_msg)

decrypted\_msg = decrypt(encrypted\_msg, shift)

print("Decrypted message is :", decrypted\_msg)

elif n == 3:

msg = input("Message to be encrypted: ")

key = input("Enter the key : ")

encrypted\_msg = transpose\_encrypt(msg, key)

print("Encrypted Message is:", encrypted\_msg)

decrypted\_msg = transpose\_decrypt(encrypted\_msg, key)

null\_count = decrypted\_msg.count('\_')

if null\_count > 0:

print("Decrypted Message is:", decrypted\_msg[:-null\_count])

elif n == 4:

plain\_text = input("Message to be encrypted: ")

input\_key = input("Enter the input key : ")

input\_key2 = input("Enter another input key for second transposition : ")

encrypted\_msg = transpose\_encrypt(plain\_text, input\_key)

double\_encrypted\_msg = transpose\_encrypt(encrypted\_msg, input\_key2)

print("Encrypted Message after double transposition :", double\_encrypted\_msg)

decrypted\_msg = transpose\_decrypt(double\_encrypted\_msg, input\_key2)

double\_decrypted\_msg = transpose\_decrypt(decrypted\_msg, input\_key)

null\_count = double\_decrypted\_msg.count('\_')

if null\_count > 0:

print("Decrypted Message after double transposition :", double\_decrypted\_msg[:-null\_count])

elif n == 5:

msg = input("Message to be encrypted: ")

key = input("Enter the input key : ")

if len(msg) != len(key):

print("length of input key and plain text should be same")

else:

plain\_text = msg.upper()

input\_key = key.upper()

cipher\_text = VC(plain\_text, input\_key)

print("Encrypted message :", cipher\_text)

decipher\_text = VC(cipher\_text, input\_key)

print("Decrypted message :", decipher\_text)

elif n == 6:

g = int(input("Enter the Prime number g: "))

n = int(input("Enter the Second Prime number of n: "))

a = int(input("Private key of person A: "))

b = int(input("Private key of person B: "))

ka, kb = DiffieHellman(g, n, a, b)

print("K1:", ka)

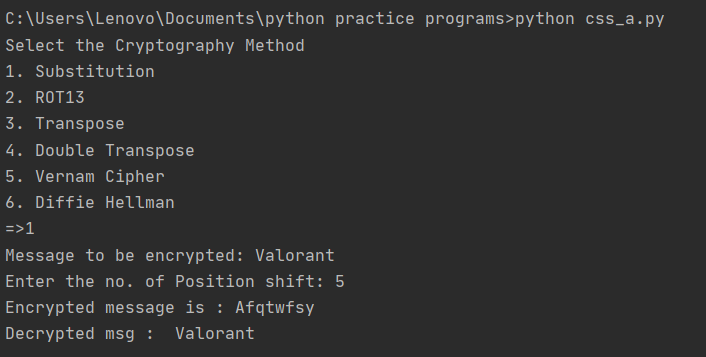
print("K2:", kb)

else:

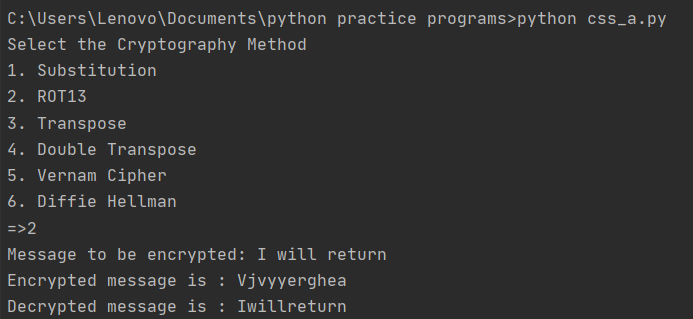
print("Choose from given options")

OUTPUT:

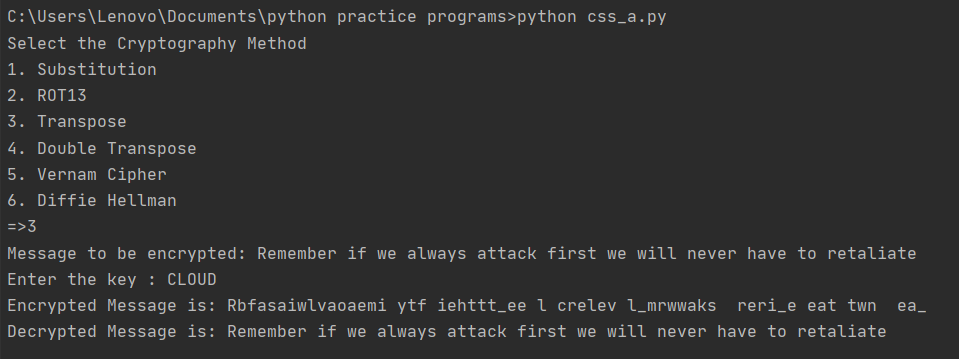
1. Substitution



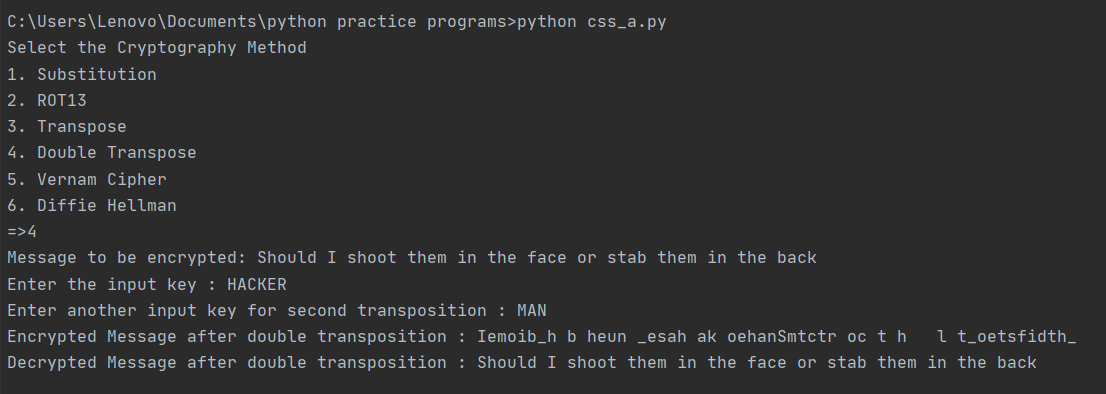
1. ROT 13



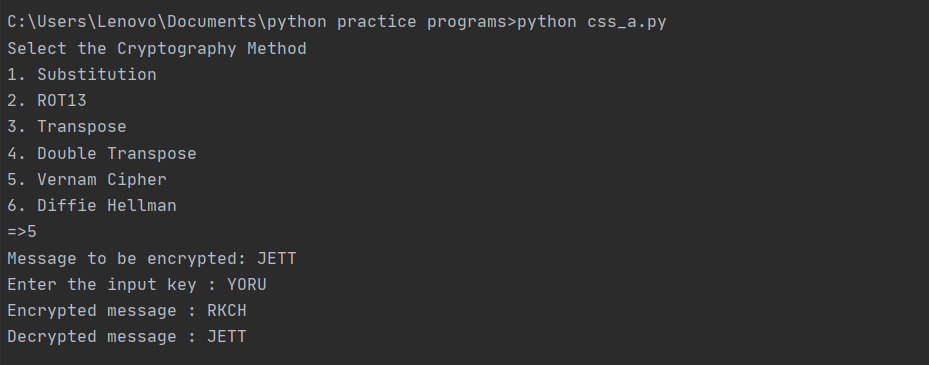
1. Transpose



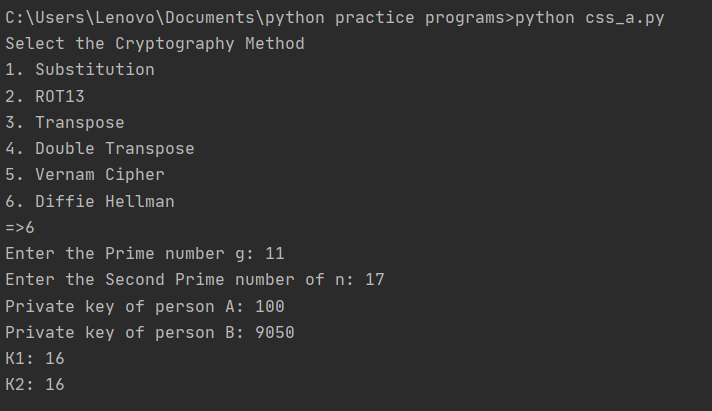
1. Double Transposition



1. Vernam Cipher



1. Diffie –Hellman



OBSERVATIONS:

1. In Substitution method the text is shifted and it can be cracked by observing some pattern in the encrypted code
2. ROT13 has fixed shifting of characters with the shift of 13 characters even it can be cracked as it is done for Substitution method.
3. In Transpose method the characters positions are interchanged in a particular manner with the help of key also known as Columnar Transposition.
4. If the length of key/ message is known the cipher can be decrypted. For more high level protection we use double transpose, as the encrypted message is again encrypted with the key given and is more secured.
5. In Vernam Cipher since XOR function is used between plain text & key and cipher & key, we don’t need different functions for encrypting and decrypting.
6. In Diffie Hellman, considering Alice(a) and Bob(b) as our example both has their own private key and in the insecure public communication channel there is one key G and other number N which is a very large number around 4000 bits so that when private key of Alice’s and Bob’s private key are out there public can access only (ag) and (bg) which is very difficult to separate and when these keys are exchanged both can have (abg) which is same and no one get the original message straight out form the insecure public communication channel as they will need private keys of Alice or Bob but they are not shared and the message is safe and secured.