EXPERIMENT-1 & 2

Name: Aartee chimate
UID:2018140012
Branch:IT
Sub: CSS

AIM: To implement Substitution, ROT13, Transposition, Double Transposition, Vernam cipher, and Diffie Hellman in python.

CODE:

```
import string



def SubCypher():

    # A list containing all characters
    all_letters= string.ascii_letters


    """
    create a dictionary to store the substitution
    for the given alphabet in the plain text
    based on the key
    """


    dict1 = {}

    plain_txt= input("Enter Plain text to be encrypted:  ")
    key = int(input("Enter no. of position shifts:  "))

    for i in range(len(all_letters)):
        dict1[all_letters[i]] = all_letters[(i+key)%len(all_letters)]



    cipher_txt=[]

    # loop to generate ciphertext

    for char in plain_txt:
        if char in all_letters:
```

```
            temp = dict1[char]
            cipher_txt.append(temp)
        else:
            temp =char
            cipher_txt.append(temp)

    cipher_txt= "".join(cipher_txt)
    print("Encrypted Message:  ",cipher_txt)


    #create a dictionary to store the substitution
    #for the given alphabet in the cipher
    #text based on the key


    dict2 = {}
    for i in range(len(all_letters)):
        dict2[all_letters[i]] = all_letters[(i-key)%(len(all_letters))]

    # loop to recover plain text
    decrypt_txt = []

    for char in cipher_txt:
        if char in all_letters:
            temp = dict2[char]
            decrypt_txt.append(temp)
        else:
            temp = char
            decrypt_txt.append(temp)

    decrypt_txt = "".join(decrypt_txt)
    print("Decrypted Message:  ", decrypt_txt)


def ROT13():


    # Dictionary to lookup the index of alphabets
    dict1 = {'A' : 1, 'B' : 2, 'C' : 3, 'D' : 4, 'E' : 5,
            'F' : 6, 'G' : 7, 'H' : 8, 'I' : 9, 'J' : 10,
            'K' : 11, 'L' : 12, 'M' : 13, 'N' : 14, 'O' : 15,
            'P' : 16, 'Q' : 17, 'R' : 18, 'S' : 19, 'T' : 20,
            'U' : 21, 'V' : 22, 'W' : 23, 'X' : 24, 'Y' : 25, 'Z' : 26}
```

```
# Dictionary to lookup alphabets
# corresponding to the index after shift
dict2 = {0 : 'Z', 1 : 'A', 2 : 'B', 3 : 'C', 4 : 'D', 5 : 'E',
        6 : 'F', 7 : 'G', 8 : 'H', 9 : 'I', 10 : 'J',
        11 : 'K', 12 : 'L', 13 : 'M', 14 : 'N', 15 : 'O',
        16 : 'P', 17 : 'Q', 18 : 'R', 19 : 'S', 20 : 'T',
        21 : 'U', 22 : 'V', 23 : 'W', 24 : 'X', 25 : 'Y'}

# Function to encrypt the string
# according to the shift provided
def encrypt(message, shift):
    cipher = ''
    for letter in message:
        # checking for space
        if(letter != ' '):
            # looks up the dictionary and
            # adds the shift to the index
            num = ( dict1[letter] + shift ) % 26
            # looks up the second dictionary for
            # the shifted alphabets and adds them
            cipher += dict2[num]
        else:
            # adds space
            cipher += ' '

    return cipher

# Function to decrypt the string
# according to the shift provided
def decrypt(message, shift):
    decipher = ''
    for letter in message:
        # checks for space
        if(letter != ' '):
            # looks up the dictionary and
            # subtracts the shift to the index
            num = ( dict1[letter] - shift + 26) % 26
            # looks up the second dictionary for the
            # shifted alphabets and adds them
            decipher += dict2[num]
        else:
            # adds space
            decipher += ' '
```

```
        return decipher

    # function to run the program
    def main():
        # use 'upper()' function to convert any lowercase characters to uppercase
        message = input("Enter plain text to be encrypted:  ")
        shift = 13
        result = encrypt(message.upper(), shift)

        print("Encrypted Cypher:  ", result)

        message = result
        shift = 13
        result = decrypt(message.upper(), shift)
        print ("Decrypted Cypher:  ",result)

    # Executes the main function
    if __name__ == '__main__':
        main()



def TranspoCypher():

    import math

    def encryptMessage(msg):
        cipher = ""

        # track key indices
        k_indx = 0

        msg_len = float(len(msg))
        msg_lst = list(msg)
        key_lst = sorted(list(key))

        # calculate column of the matrix
        col = len(key)

        # calculate maximum row of the matrix
        row = int(math.ceil(msg_len / col))

        # add the padding character '_' in empty
        # the empty cell of the matix
```

```
    fill_null = int((row * col) - msg_len)
    msg_lst.extend('_' * fill_null)

    # create Matrix and insert message and
    # padding characters row-wise
    matrix = [msg_lst[i: i + col]
              for i in range(0, len(msg_lst), col)]

    # read matrix column-wise using key
    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 decryptMessage(cipher):
    msg = ""

    # track key indices
    k_indx = 0

    # track msg indices
    msg_indx = 0
    msg_len = float(len(cipher))
    msg_lst = list(cipher)

    # calculate column of the matrix
    col = len(key)

    # calculate maximum row of the matrix
    row = int(math.ceil(msg_len / col))

    # convert key into list and sort
    # alphabetically so we can access
    # each character by its alphabetical position.
    key_lst = sorted(list(key))

    # create an empty matrix to
    # store deciphered message
    dec_cipher = []
    for _ in range(row):
        dec_cipher += [[None] * col]
```

```
        # Arrange the matrix column wise according
        # to permutation order by adding into new matrix
        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

        # convert decrypted msg matrix into a string
        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


    msg = input("Enter Plain text to be Encrypted:  ")
    key = input("Enter Key:  ")
    cipher = encryptMessage(msg)
    print("Encrypted Message: {}".
               format(cipher))

    print("Decryped Message: {}".
          format(decryptMessage(cipher)))

def DoubTranspoCypher():


    import math




    def encryptMessage(msg, key):
        cipher = ""
```

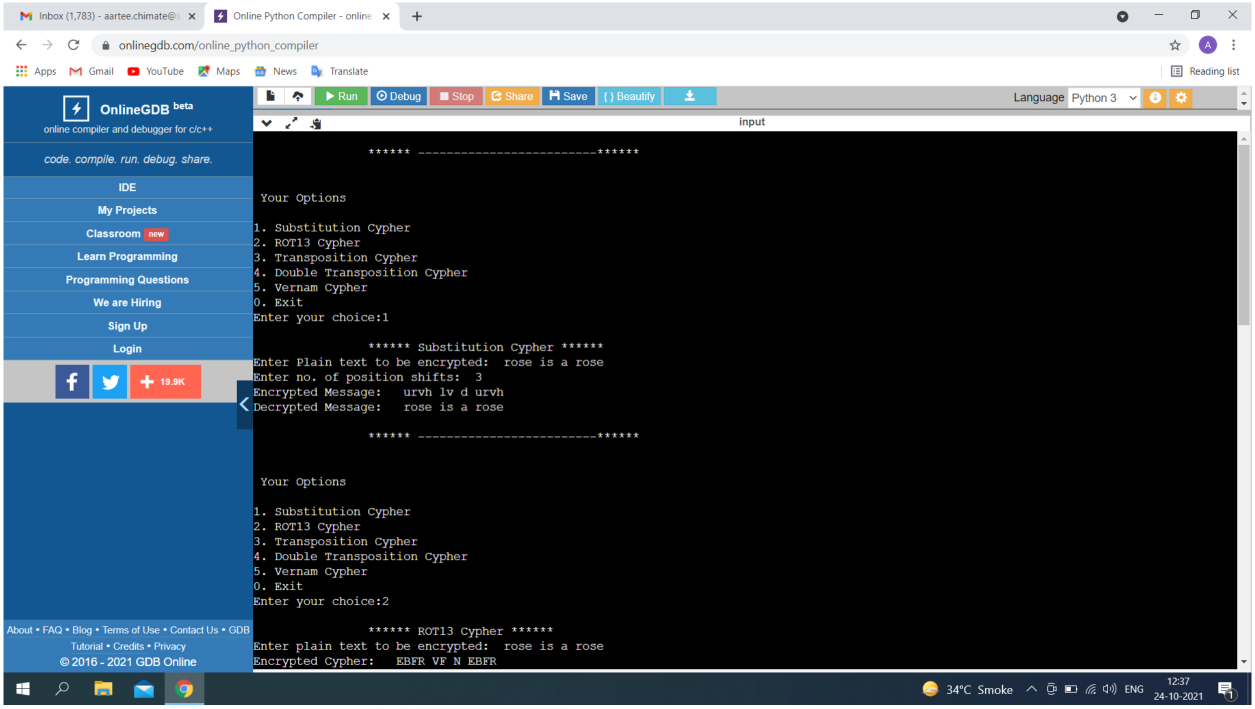
```
    # track key indices
    k_indx = 0

    msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))

    # calculate column of the matrix
    col = len(key)

    # calculate maximum row of the matrix
    row = int(math.ceil(msg_len / col))

    # add the padding character '_' in
    # the empty cell of the matix
    fill_null = int((row * col) - msg_len)
    msg_lst.extend('_' * fill_null)

    # create Matrix and insert message and
    # padding characters row-wise
    matrix = [msg_lst[i: i + col]
              for i in range(0, len(msg_lst), col)]

    # read matrix column-wise using key
    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 decryptMessage(cipher, key):
    msg = ""

    # track key indices
    k_indx = 0

    # track msg indices
    msg_indx = 0
```

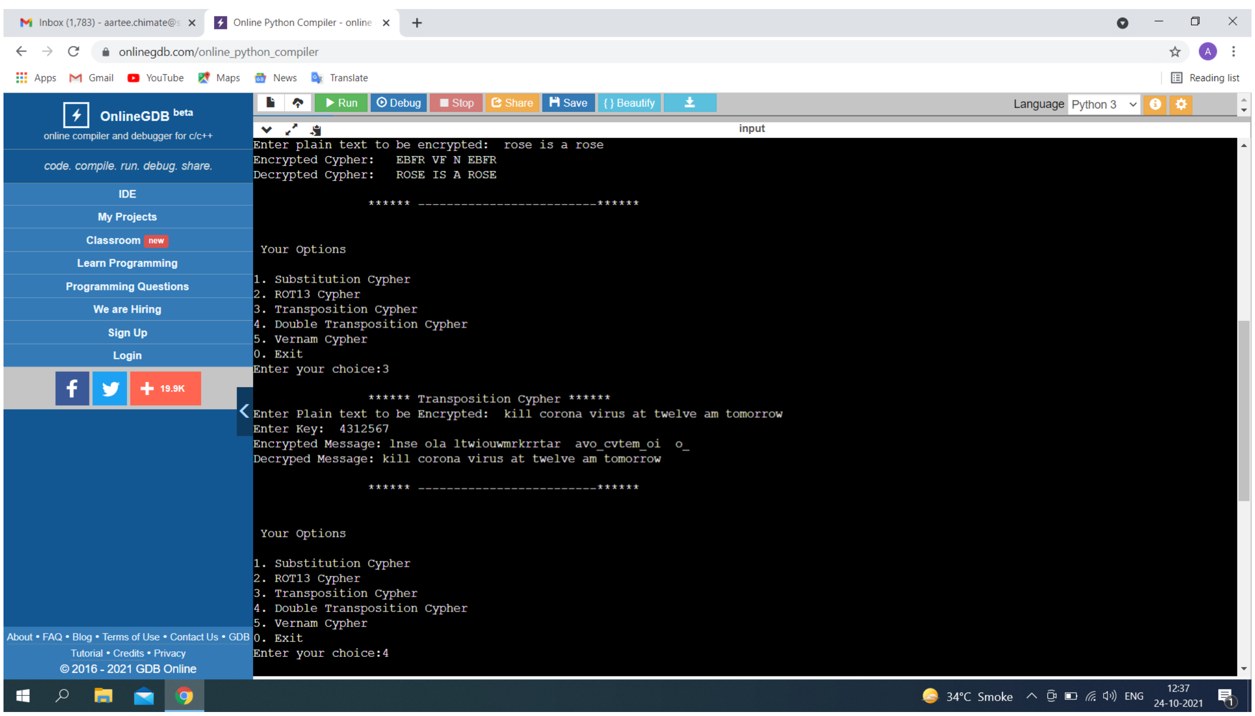
```
msg_len = float(len(cipher))
msg_lst = list(cipher)

# calculate column of the matrix
col = len(key)

# calculate maximum row of the matrix
row = int(math.ceil(msg_len / col))

# convert key into list and sort
# alphabetically so we can access
# each character by its alphabetical position.
key_lst = sorted(list(key))

# create an empty matrix to
# store deciphered message
dec_cipher = []
for _ in range(row):
    dec_cipher += [[None] * col]

# Arrange the matrix column wise according
# to permutation order by adding into new matrix

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

# convert decrypted msg matrix into a string
try:
    msg = ''.join(sum(dec_cipher, []))

except TypeError:
    raise TypeError("This program cannot",
                    "handle repeating words.")



"""
if null_count > 0:
```

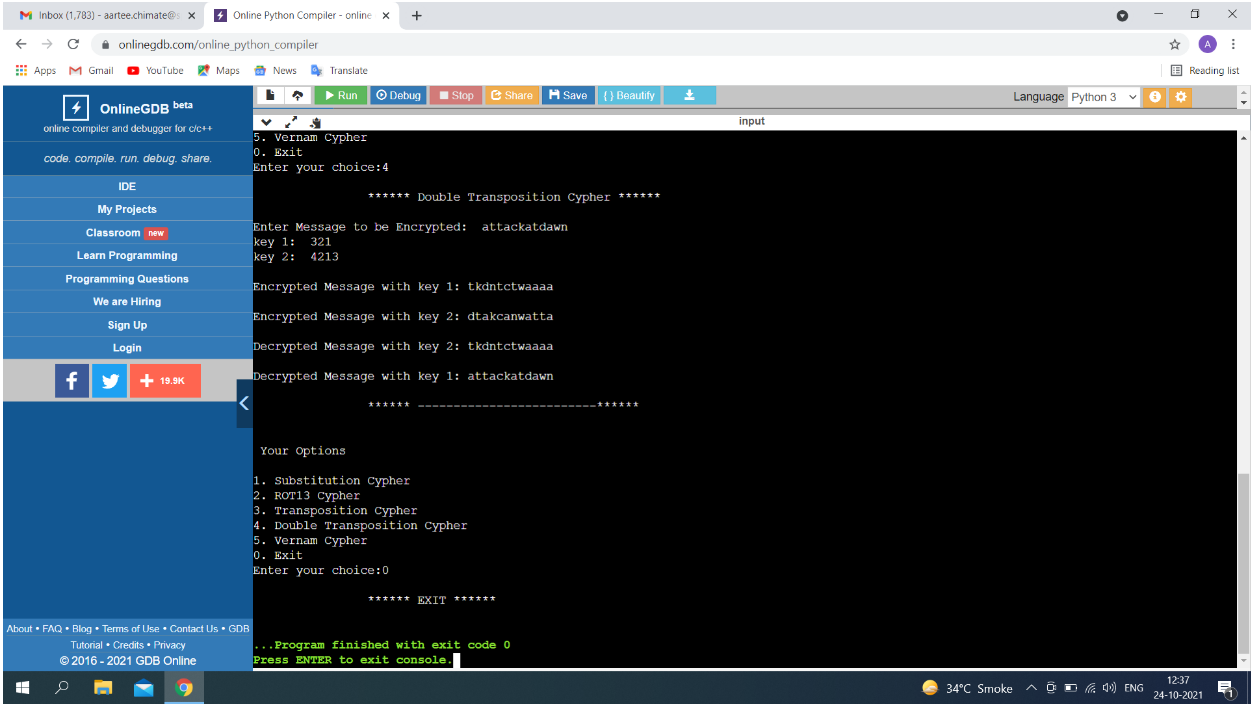
```
        return msg[: -null_count]
    """
    return msg


# Driver Code
msg = input("\nEnter Message to be Encrypted:  ")
key1 = input("key 1:  ")
key2 = input("key 2:  ")


cipher = encryptMessage(msg, key1)
print("\nEncrypted Message with key 1: {}".format(cipher))


cipher = encryptMessage(cipher, key2)
print("\nEncrypted Message with key 2: {}".format(cipher))

print("\nDecrypted Message with key 2: {}".format(decryptMessage(cipher, key2)))


decryCypherMess = decryptMessage(cipher, key2)
print("\nDecrypted Message with key 1: {}".format(
    decryptMessage(decryCypherMess, key1)))


def VernamCypher():
    def VernamCipherFunction(text, key):
        result = "";
        ptr = 0;
        for char in text:
            result = result + chr(ord(char) ^ ord(key[ptr]));
            ptr = ptr + 1;
            if ptr == len(key):
                ptr = 0;
        return result




    input_text = input("\nEnter Text To Encrypt:\t");
    encryption_key = input("Input key:  ");
    encryption = VernamCipherFunction(input_text, encryption_key);
    print("\nEncrypted Vernam Cipher Text:\t" + encryption);
```

```
    decryption = VernamCipherFunction(encryption, encryption_key);
    print("\nDecrypted Vernam Cipher Text:\t" + decryption);


def main():
    flag=1;
    while flag==1:
        print("\n\t\t****** -------------------------******\t\t");
        print("\n\n Your Options\n");
        print("1. Substitution Cypher");
        print("2. ROT13 Cypher");
        print("3. Transposition Cypher");
        print("4. Double Transposition Cypher");
        print("5. Vernam Cypher");
        print("0. Exit");
        c=int(input("Enter your choice:"));

        if c==1:
            print("\n\t\t****** Substitution Cypher ******\t\t");
            SubCypher();

        elif c==2:
            print("\n\t\t****** ROT13 Cypher ******\t\t");
            ROT13();

        elif c==3:
            print("\n\t\t****** Transposition Cypher ******\t\t");
            TranspoCypher();

        elif c==4:
            print("\n\t\t****** Double Transposition Cypher ******\t\t");
            DoubTranspoCypher();

        elif c==5:
            print("\n\t\t****** Vernam Cypher ******\t\t");
            VernamCypher();
        elif c==0:
            print("\n\t\t****** EXIT ******\t\t");
            flag=0;

if __name__ == "__main__":
    main()
```

OUTPUT:


****** ----------------------------******
Your Options
1. Substitution Cypher
2. ROT13 Cypher
3. Transposition Cypher
4. Double Transposition Cypher
5. Vernam Cypher
0. Exit
Enter your choice:1
****** Substitution Cypher ******
Enter Plain text to be encrypted: rose is a rose
Enter no. of position shifts: 3
Encrypted Message: urvh lv d urvh
Decrypted Message: rose is a rose
****** ----------------------------******
Your Options
1. Substitution Cypher
2. ROT13 Cypher
3. Transposition Cypher
4. Double Transposition Cypher
5. Vernam Cypher
0. Exit
Enter your choice:2
****** ROT13 Cypher ******
Enter plain text to be encrypted: rose is a rose
Encrypted Cypher: EBFR VF N EBFR


Enter plain text to be encrypted: rose is a rose
Encrypted Cypher: EBFR VF N EBFR
Decrypted Cypher: ROSE IS A ROSE
****** ----------------------------******
Your Options
1. Substitution Cypher
2. ROT13 Cypher
3. Transposition Cypher
4. Double Transposition Cypher
5. Vernam Cypher
0. Exit
Enter your choice:3
****** Transposition Cypher ******
Enter Plain text to be Encrypted: kill corona virus at twelve am tomorrow
Enter Key: 4312567
Encrypted Message: lnse ola ltwiouwmrkrrtar avo_cvtem_oi o_
Decryped Message: kill corona virus at twelve am tomorrow
****** ----------------------------******
Your Options
1. Substitution Cypher
2. ROT13 Cypher
3. Transposition Cypher
4. Double Transposition Cypher
5. Vernam Cypher
0. Exit
Enter your choice:4



Conclusion:

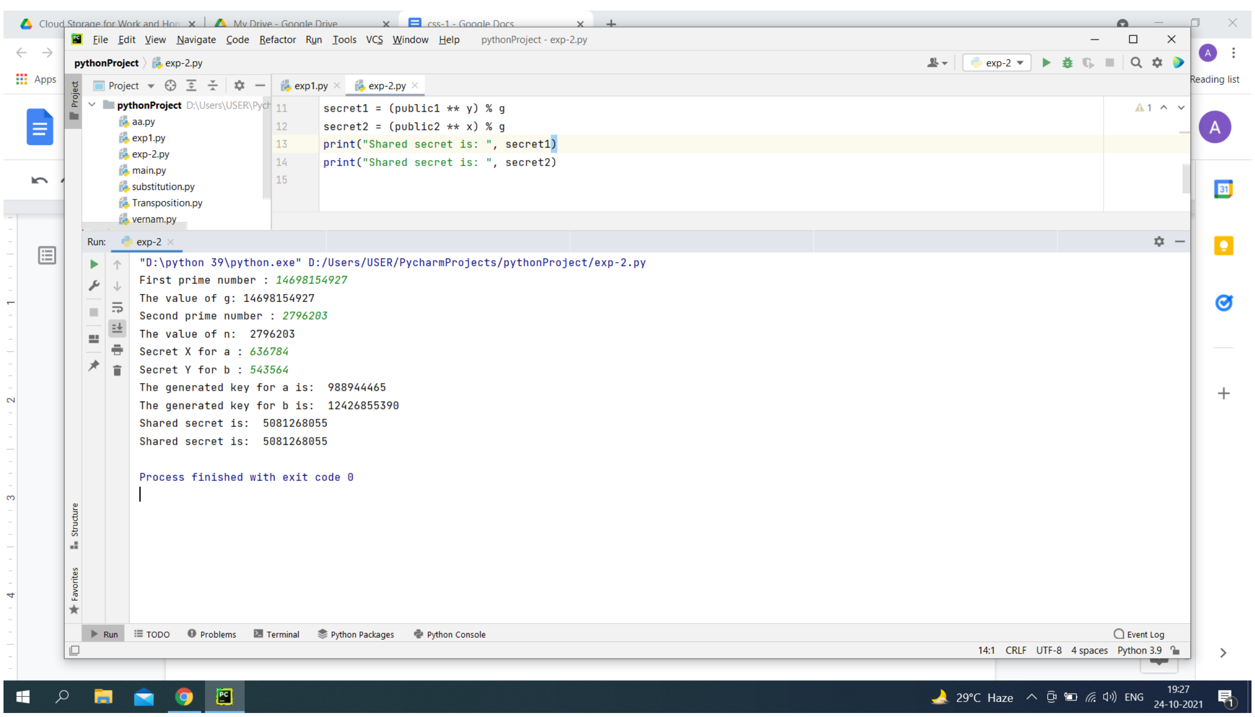
1. In the substitution algorithm and ROT 13 algorithm , I had to make sure that while shifting the ASCII values of the characters, the final ordinal value of every character does not exceed the total number of available characters. I solved this problem by utilizing the modulo operation and applied it while encrypting as well as decrypting a given string.
2. A transposition cipher does not substitute one symbol for another, instead it changes the location of the symbol. A symbol in the first position of the plaintext may appear in the tenth position of the ciphertext. A symbol in the eight position in the plaintext may appear in the first position of the ciphertext. Basically, a transposition cipher reorders (transpose) the symbols.
3. Double transposition designates the letters in the original plaintext message by the numbers designating their position. First the plaintext is written into an array of a given size and then permutation of rows and columns is done according to the specified permutations.
4. The key is exactly the same as the length of the message which is encrypted for vernam cipher.. The key is made up of random symbols. The key is used one time only and never used again for any other message to be encrypted.The vernam cipher is an unbreakable symmetric encryption technique. The key is unbreakable because it is as long as the given message.

Diffie Hellman:

CODE:

```
g = int(input("First prime number : "))
print("The value of g:", g)
n = int(input("Second prime number : "))
print("The value of n: ",n)
x = int(input("Secret X for a : "))
y = int(input("Secret Y for b : "))
public1 = (n ** x) % g
public2 = (n ** y) % g
print("The generated key for a is: ", public1)
print("The generated key for b is: ", public2)
secret1 = (public1 ** y) % g
secret2 = (public2 ** x) % g
print("Shared secret is: ", secret1)
print("Shared secret is: ", secret2)
```

OUTPUT:



Conclusion:

1. Diffie Hellman algorithm generating a symmetric key on a public network. This algorithm makes use of the primitive root of any prime numbers that are given by the user.This algorithm enables two parties communicating over a public channel to establish a mutual secret without it being transmitted over the internet.I found that as long as it is implemented alongside an appropriate authentication method and the numbers have been selected properly, it is not considered vulnerable to attack.