NAME: THAKKAR MALAY TUSHARBHAI
ENROLLMENT NO.:20012011169
DEPARTMENT: B.Tech. (CE)
Batch: CEIT-B_6AB4
SUBJECT: 2CEIT6PE3: CRYPTOGRAPHY AND NETWORK SECURITY
COLLEGE: U. V. PATEL COLLEGE OF ENGINEERING

Practical-1

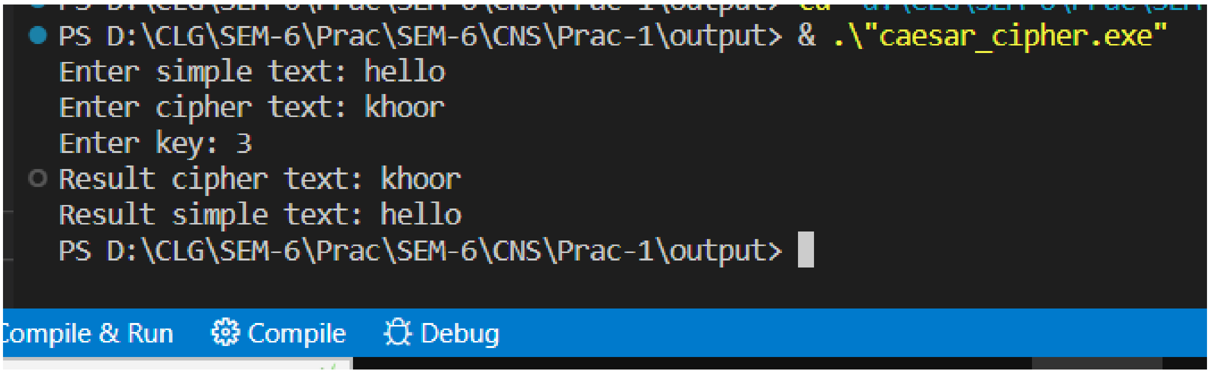
1. **Write a program to perform encryption and decryption using Caesar cipher algorithm. Encryption procedure: C=E(P)=(P+K) mod 26**
   Decryption Procedure: P=D(C)=(C-K) mod 26

- **CODE:**

```
#include <iostream>;
using namespace std;
string cipher_encryption(string text,int key)
{
    string result = "";
    for (int i = 0; i < text.length(); i++) {
        if (isupper(text[i]))
            result += char(int(text[i] + key - 65) % 26 + 65);
        else
            result += char(int(text[i] + key - 97) % 26 + 97);
    }
    return result;
}
string cipher_decryption(string text,int key)
{
   char ch;
      for(int i = 0; text[i] != '\0'; ++i) {
         ch = text[i];
         //decrypt for lowercase letter
         if(ch >= 'a' && ch <= 'z') {
            ch = ch - key;
            if(ch < 'a'){
               ch = ch + 'z' - 'a' + 1;
            }
            text[i] = ch;
         }
         //decrypt for uppercase letter
         else if(ch >= 'A' && ch <= 'Z') {
            ch = ch - key;
            if(ch < 'A') {
               ch = ch + 'Z' - 'A' + 1;
            }
```

```
            text[i] = ch;
          }
        }
        return text;
}
int main(){
    string simpletext="";
    string result_simpletext="";
    string ciphertext="";
    string result_ciphertext="";
    int key=0;
    cout<<"Enter simple text: ";
    cin>>simpletext;
     cout<<"Enter cipher text: ";
    cin>>ciphertext;
    cout<<"Enter key: ";
    cin>>key;
    result_ciphertext=cipher_encryption(simpletext,key);
    result_simpletext=cipher_decryption(ciphertext,key);
    cout<<"Result cipher text: "<<result_ciphertext<<endl;
    cout<<"Result simple text: "<<result_simpletext<<endl;
}
```

- **OUTPUT:**



2. Write a program to perform encryption and decryption using Modified Caesar cipher algorithm.

- **CODE:**

```
def encryption(planeText):
    encryptData = ""
    for i in range(len(planeText)):
        char = planeText[i]
        if char.isupper():
            if i % 2 == 0:
```

```
                encryptData += chr((ord(char) + 1 - 65) % 26 + 65)
            else:
                encryptData += chr((ord(char) - 1 - 65) % 26 + 65)
        elif char.islower():
            if i % 2 == 0:
                encryptData += chr((ord(char) + 1 - 97) % 26 + 97)
            else:
                encryptData += chr((ord(char) - 1 - 97) % 26 + 97)
        else:
            encryptData += char
    return encryptData

def decryption(cipherText):
    decryptData = ""
    for i in range(len(cipherText)):
        char = cipherText[i]
        if char.isupper():
            if i % 2 == 0:
                decryptData += chr((ord(char) - 1 - 65) % 26 + 65)
            else:
                decryptData += chr((ord(char) + 1 - 65) % 26 + 65)
        elif char.islower():
            if i % 2 == 0:
                decryptData += chr((ord(char) - 1 - 97) % 26 + 97)
            else:
                decryptData += chr((ord(char) + 1 - 97) % 26 + 97)
        else:
            decryptData += char
    return decryptData
planeText = input("Plain text: ")
ct = encryption(planeText)
print("Cipher text:", ct)
print()
cipherText = input("Encrypted text: ")
dt = decryption(cipherText)
print("Plain text:", dt)
```

OUTPUT:

```
PS D:\CLG\SEM-6\Prac\SEM-6\CNS> python -u "d:\CLG\SEM-6\Prac\SEM-6\CNS\Prac-1\demo.py"
Plain text: hello
Cipher text: idmkp

Encrypted text: idmkp
Plain text: hello
PS D:\CLG\SEM-6\Prac\SEM-6\CNS>
```

Practical-2

AIM: Write a program to find plain text messages and key information corresponds to following cipher text messages using brute-force technique on Caesar cipher.

- **Code:**

```
#finding plain text messages and key information of
#cipher text messages using brute-force technique on Caesar cipher

def decryption(text, key):
    decrypted_text = ""
    for char in text:
        if char.isupper():
            decrypted_text += chr((ord(char) - key - 65) % 26 + 65)
        else:
            decrypted_text += chr((ord(char) - key - 97) % 26 + 97)
    return decrypted_text


encrypted_text = input("Enter encrypted text : ")
for key in range(26):
    decrypted_text = decryption(encrypted_text, key)
    print("Key : " + str(key))
    print("Decrypted text : " + decrypted_text + "\n")
```

- **Output:**

1. PmttwEmtkwumBwCDXKM

```
Key : 7
Decrypted text : IfmmpXfmdpnfUpVWQDF

Key : 8
Decrypted text : HelloWelcomeToUVPCE

Key : 9
Decrypted text : GdkknVdkbnldSnTUOBD
```

2. Qefpfpzxbpbozfmeboxidlofqej

```
Key : 22
Decrypted text : Uijtjtdbftfsdjqifsbmhpsjuin

Key : 23
Decrypted text : Thisiscaesercipheralgorithm

Key : 24
Decrypted text : Sghrhrbzdrdqbhogdqzkfnqhsgl
```

3. TrvjviTzgyvizjNvrbRcxfizkyd

```
Key : 16
Decrypted text : DbftfsDjqifsjtXfblBmhpsjuin

Key : 17
Decrypted text : CaeserCipherisWeakAlgorithm

Key : 18
Decrypted text : BzdrdqBhogdqhrVdzjZkfnqhsgl
```

4. LbhNerFzneggbNggnpxPnrfrePvcure

```
Key : 12
Decrypted text : ZpvBsfTnbsuupBuubdlDbftfsDjqifs

Key : 13
Decrypted text : YouAreSmarttoAttackCaeserCipher

Key : 14
Decrypted text : XntZqdRlzqssnZsszbjBzdrdqBhogdq
```

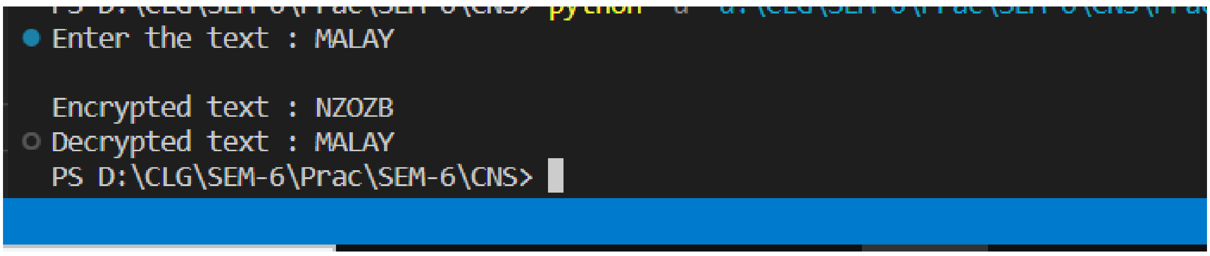
Practical-3

AIM: - Write a program to perform encryption and decryption using Mono-alphabetic Cipher Technique.

➢ **CODE:**

```
#encryption and decryption using Mono-alphabetic Cipher Technique
mono_alpha = {
    'A':'Z','B':'Y','C':'X','D':'W','E':'V','F':'U','G':'T','H':'S','I':'R','J':'Q','K':'P','L':'O','M':'N','N':'M','O':'L','P':'K','Q':'J','R':'I','S':'H','T':'G','U':'F','V':'E','W':'D','X':'C','Y':'B','Z':'A'
}
def encryption(text):
    encrypted_text = ""
    for char in text:
        encrypted_text += mono_alpha[char]
    return encrypted_text

def decryption(text):
    decrypted_text = ""
    for char in text:
        decrypted_text += list(mono_alpha.keys())[list(mono_alpha.values()).index(char)]
    return decrypted_text

text = input("Enter the text : ")
encrypted_text = encryption(text)
print("\nEncrypted text : " + encrypted_text)

decrypted_text = decryption(encrypted_text)
print("Decrypted text : " + decrypted_text)
```

➢ **OUTPUT:**


```
Enter the text : MALAY

Encrypted text : NZOZB
Decrypted text : MALAY
PS D:\CLG\SEM-6\Prac\SEM-6\CNS>
```

Practical-4

AIM: - Write a program to perform encryption and decryption using Polyalphabetic Cipher (Vigenere Cipher) Technique.

➢ CODE:

```
def encryption(plain_text, key_word):
    j = 0
    encrypted_text = ""
    for p in plain_text:
        cipher_value = (ord(p)-65 + ord(key_word[j])-65) % 26
        cipher_text = chr(cipher_value + 65)
        encrypted_text += cipher_text
        if(j == len(key_word)-1):
            j = 0
        else:
            j += 1
    return encrypted_text
def decryption(encrypted_text, key_word):
    j = 0
    decrypted_text = ""
    for p in encrypted_text:
        plain_text_value = (ord(p)-65 - ord(key_word[j])-65) % 26
        plain_text = chr(plain_text_value + 65)
        decrypted_text += plain_text
        if(j == len(key_word)-1):
            j = 0
        else:
            j += 1
    return decrypted_text
plain_text =input("Enter Plaintext: ")
key_word = input("Enter Keyword: ")
encrypted_text = encryption(plain_text, key_word)
print("\nEncrypted text : " + encrypted_text)

decrypted_text = decryption(encrypted_text, key_word)
print("Decrypted text : " + decrypted_text)
```

➢ OUTPUT:

```
Enter Plaintext: WENEEDMOREMONEY
Enter Keyword: MEC

Encrypted text : IIPQIFYSTQQQZIA
Decrypted text : WENEEDMOREMONEY
PS D:\CLG\SEM-6\Prac\SEM-6\CNS>
```

Practical-5

AIM: - Write a program to perform encryption and decryption using Play-fair Cipher Technique.

CODE:

```
key = input("Enter key: ")
key = key.replace(" ", "")
key = key.upper()

def matrix(x, y, initial):
    return [[initial for i in range(x)] for j in range(y)]
result = list()
for i in key:  # storing key
    if i not in result:
        if i == "J":
            result.append("I")
        else:
            result.append(i)
flag = 0
for i in range(65, 91):  # storing other character
    if chr(i) not in result:
        if i == 73 and chr(74) not in result:
            result.append("I")
            flag = 1
    elif flag == 0 and i == 73 or i == 74:
        pass
    else:
        result.append(chr(i))
k = 0
my_matrix = matrix(5, 5, 0)  # initialize matrix
for i in range(0, 5):  # making matrix
    for j in range(0, 5):
        my_matrix[i][j] = result[k]
        k += 1
print("\nPLAY-FAIR KEYWORD MATRIX:")
for i in range(0, 5):
    for j in range(0, 5):
        print(my_matrix[i][j], end=" ")
    print()
def locindex(c):  # get location of each character
    loc = list()
    if c == "J":
        c = "I"
    for i, j in enumerate(my_matrix):
```

```
        for k, l in enumerate(j):
            if c == l:
                loc.append(i)
                loc.append(k)
                return loc

def encrypt():  # Encryption
    msg = str(input("Enter the plaintext: "))
    msg = msg.upper()
    msg = msg.replace(" ", "")
    i = 0
    for s in range(0, len(msg) + 1, 2):
        if s < len(msg) - 1:
            if msg[s] == msg[s + 1]:
                msg = msg[: s + 1] + "X" + msg[s + 1:]
    if len(msg) % 2 != 0:
        msg = msg[:] + "X"
    print("\nCipher text:", end=" ")
    while i < len(msg):
        loc = list()
        loc = locindex(msg[i])
        loc1 = list()
        loc1 = locindex(msg[i + 1])
        if loc[1] == loc1[1]:
            print(
                "{}{}".format(
                    my_matrix[(loc[0] + 1) % 5][loc[1]],
                    my_matrix[(loc1[0] + 1) % 5][loc1[1]],
                ),
                end=" ",
            )
        elif loc[0] == loc1[0]:
            print(
                "{}{}".format(
                    my_matrix[loc[0]][(loc[1] + 1) % 5],
                    my_matrix[loc1[0]][(loc1[1] + 1) % 5],
                ),
                end=" ",
            )
        else:
            print(
                "{}{}".format(my_matrix[loc[0]][loc1[1]],
                              my_matrix[loc1[0]][loc[1]]),
                end=" ",
            )
            i = i + 2
    print()
```

```
def decrypt():  # decryption
    msg = str(input("Enter Cipher text: "))
    msg = msg.upper()
    msg = msg.replace(" ", "")
    decrypted_msg = ""
    i = 0
    while i < len(msg):
        loc = list()
        loc = locindex(msg[i])
        loc1 = list()
        loc1 = locindex(msg[i + 1])
        if loc[1] == loc1[1]:
            decrypted_msg += "{}{}".format(my_matrix[(loc[0] - 1) %
                                           5][loc[1]], my_matrix[(loc1[0] - 1) % 5][loc1[1]]) + " "
        elif loc[0] == loc1[0]:
            decrypted_msg += "{}{}".format(my_matrix[loc[0]][(loc[1] - 1) %
                                           5], my_matrix[loc1[0]][(loc1[1] - 1) % 5]) + " "
        else:
            decrypted_msg += "{}{}".format(my_matrix[loc[0]]
                                           [loc1[1]], my_matrix[loc1[0]][loc[1]]) + " "
            i = i + 2
        print("\nPlain text:", decrypted_msg)
        decrypted_msg = decrypted_msg.replace('X', '').replace(" ", "")
        print("\nPlain text after romoving all X and spces:", decrypted_msg)

while 1:
    choice = int(input("\nChoose one: \n1. Encryption \n2. Decryption \n3. Exit\n\n"))
    if choice == 1:
        encrypt()
    elif choice == 2:
        decrypt()
    elif choice == 3:
        break
    else:
        print("Please, choose correct choice.")
```

OUTPUT:

```
E:\Sem-6\CNS\Practical-5>py 5.py
Enter key: FHSDIKN

PLAY-FAIR KEYWORD MATRIX:
F H S D I
K N A B C
E G L M O
P Q R T U
1. Encryption
2. Decryption
3. Exit

2
Enter Cipher text: CHTLLR

Plain text: NI RM AL

Plain text after romoving all X and spces: NIRMAL

Choose one:
1. Encryption
2. Decryption
3. Exit

3

E:\Sem-6\CNS\Practical-5>
```

Practical-6

AIM: Write a program to perform encryption and decryption using Rail-Fence Cipher Technique.

- **CODE:**

```
import re
def cipher_encryption():
    msg = input("Enter message: ")
    rails = int(input("Enter number of rails: "))
    msg = msg.replace(" ", "")
    railMatrix = []

    for i in range(rails):
        railMatrix.append([])
    for row in range(rails):
        for column in range(len(msg)):
            railMatrix[row].append('.')

    row = 0
    check = 0
    for i in range(len(msg)):
        if check == 0:
            railMatrix[row][i] = msg[i]
            row += 1
            if row == rails:
                check = 1
                row -= 1

        elif check == 1:
            row -= 1
            railMatrix[row][i] = msg[i]
            if row == 0:
                check = 0
                row = 1
    encryp_text = ""
    for i in range(rails):
        for j in range(len(msg)):
            encryp_text += railMatrix[i][j]

    encryp_text = re.sub(r"\.", " ", encryp_text)
    print("Encrypted Text: {}".format(encryp_text))
    print()

def cipher_decryption():
    msg = input("Enter message: ")
    rails = int(input("Enter number of rails: "))
```

```
msg = msg.replace(" ", "")
railMatrix = []
for i in range(rails):
    railMatrix.append([])
for row in range(rails):
    for column in range(len(msg)):
        railMatrix[row].append('.')
row = 0
check = 0
for i in range(len(msg)):
    if check == 0:
        railMatrix[row][i] = msg[i]
        row += 1
        if row == rails:
            check = 1
            row -= 1
    elif check == 1:
        row -= 1
        railMatrix[row][i] = msg[i]
        if row == 0:
            check = 0
            row = 1
ordr = 0
for i in range(rails):
    for j in range(len(msg)):
        temp = railMatrix[i][j]
        if re.search("\\.", temp):
            continue
        else:
            railMatrix[i][j] = msg[ordr]
            ordr += 1

for i in railMatrix:
    for column in i:
        print(column, end="")
    print("\n")

check = 0
row = 0
decryp_text = ""
for i in range(len(msg)):
    if check == 0:
        decryp_text += railMatrix[row][i]
        row += 1
        if row == rails:
            check = 1
            row -= 1
```

```
        elif check == 1:
            row -= 1
            decryp_text += railMatrix[row][i]
            if row == 0:
                check = 0
                row = 1
    decryp_text = re.sub(r"\.", " ", decryp_text)
    print("Decrypted Text: {}".format(decryp_text))
    print()
def main():
    while 1:
        choice = int(
            input("1. Encryption\n2. Decryption]\n3. Exit\nChoose(1,2,3): "))
        if choice == 1:
            print("\n---Encryption---\n")
            cipher_encryption()
        elif choice == 2:
            print("\n---Decryption---\n")
            cipher_decryption()
        elif choice == 3:
            break
        else:
            print("\nInvalid Choice.\n")
if __name__ == "__main__":
    main()
```

➢ OUTPUT:

```
1. Encryption
2. Decryption]
3. Exit
Choose(1,2,3): 1

---Encryption---

Enter message: malay
Enter number of rails: 2
Encrypted Text: m l y a a

1. Encryption
2. Decryption]
3. Exit
Choose(1,2,3): 2

---Decryption---

Enter message: mlyaa
Enter number of rails: 2
m.l.y

.a.a.

Decrypted Text: malay
Run Testcases  ⊗ 0 ⚠ 0
```

❖

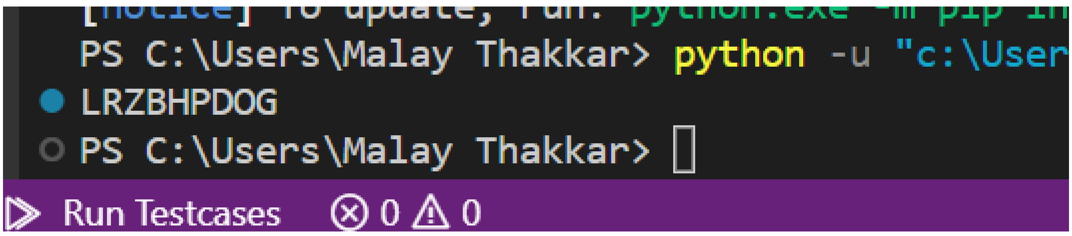
Practical-7

AIM: Write a program to perform encryption and decryption using Hill Cipher algorithm

- **Encryption (CODE)**

```
import numpy as np
import string
import random

# Define variables
dimension = 3 # Your N
key = np.matrix([[6, 24, 1], [13, 16, 10], [20, 17, 15]]) # Your key
message = 'HILLCRYPT' # Your message
# Generate the alphabet
alphabet = string.ascii_uppercase
# Encrypted message
encryptedMessage = ""

# Group message in vectors and generate crypted message
for index, i in enumerate(message):
    values = []
    # Make bloc of N values
    if index % dimension == 0:
        for j in range(0, dimension):
            if(index + j < len(message)):
                values.append([alphabet.index(message[index + j])])
            else:
                values.append([random.randint(0,25)])
        # Generate vectors and work with them
        vector = np.matrix(values)
        vector = key * vector
        vector %= 26
        for j in range(0, dimension):
            encryptedMessage += alphabet[vector.item(j)]

# Show the result
print(encryptedMessage)
```

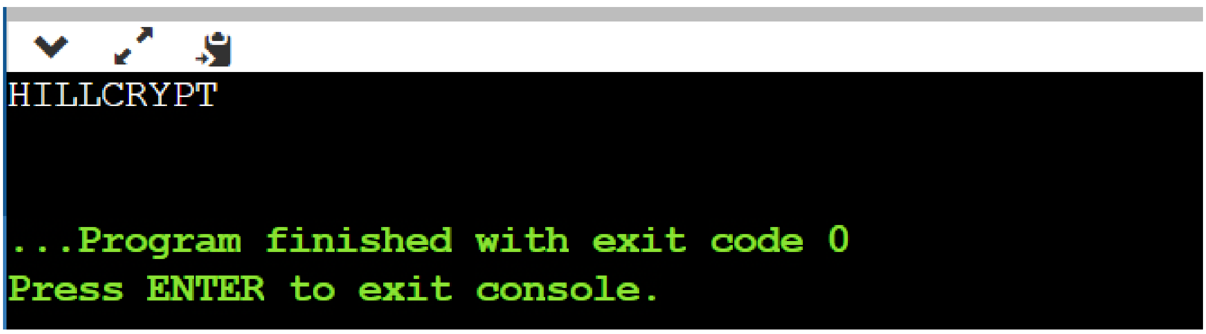
- **OUTPUT:**



➢ Encryption (CODE):

```
 # Decryption:
import numpy as np
from sympy import Matrix
import string

# Define variables
dimension = 3 # Your N
key = np.matrix([[6, 24, 1], [13, 16, 10], [20, 17, 15]]) # Your key
message = 'LRZBHPDOG'                 # You message

alphabet = string.ascii_uppercase       # Generate the alphabet

# Encrypted message
decryptedMessage = ""

# Get the decrypt key
key = Matrix(key)
key = key.inv_mod(26)
key = key.tolist()

# Group message in vectors and generate crypted message
for index, i in enumerate(message):
    values = []
    # Create the N blocs
    if index % dimension == 0:
        for j in range(0, dimension):
            values.append([alphabet.index(message[index + j])])
        # Create the vectors and work with them
        vector = np.matrix(values)
        vector = key * vector
        vector %= 26
        for j in range(0, dimension):
            decryptedMessage += alphabet[vector.item(j)]

print(decryptedMessage)    # Show the result
```

➢ OUTPUT:



Practical-8

- **AIM: Implement columnar transposition cipher encryption and decryption**
- **CODE:**
- **Encryption:**

```
import math

def row(s,key):

    # to remove repeated alphabets in key
    temp=[]
    for i in key:
        if i not in temp:
            temp.append(i)
    k=""
    for i in temp:
        k+=i
    print("The key used for encryption is: ",k)

    # ceil is used to adjust the count of
    # rows according to length of message
    b=math.ceil(len(s)/len(k))

    # if b is less than length of key, then it will not form square matrix when
    # length of meessage not equal to rowsize*columnsize of square matrix
    if(b<len(k)):
        b=b+(len(k)-b)
    # if b is greater than length of key, then it will not from a
    # square matrix, but if less then length of key, we have to add padding

    arr=[['_' for i in range(len(k))]
         for j in range(b)]
    i=0
    j=0
    # arranging the message into matrix
    for h in range(len(s)):
        arr[i][j]=s[h]
        j+=1
        if(j>len(k)-1):
            j=0
            i+=1
    print("The message matrix is: ")
    for i in arr:
        print(i)
```

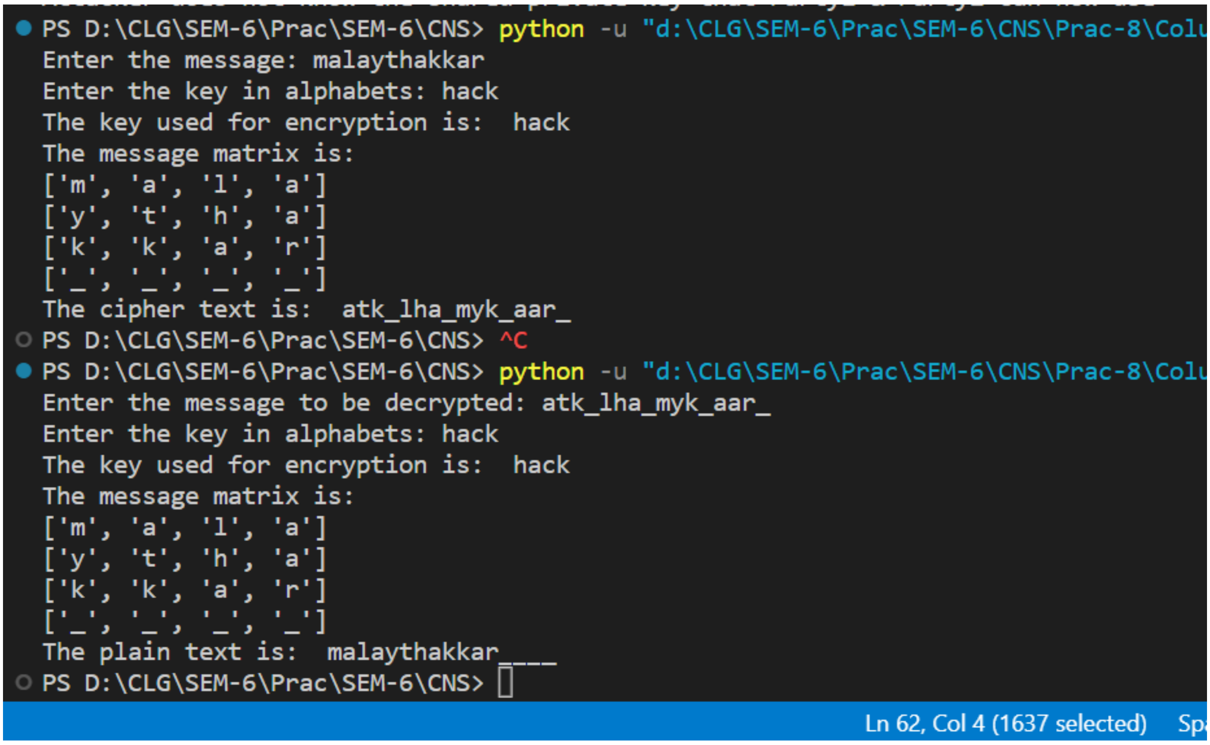
```
    cipher_text=""
    # To get indices as the key numbers instead of alphabets in the key, according
    # to algorithm, for appending the elementsof matrix formed earlier, column wise.
    kk=sorted(k)

    for i in kk:
        # gives the column index
        h=k.index(i)
        for j in range(len(arr)):
            cipher_text+=arr[j][h]
    print("The cipher text is: ",cipher_text)

msg=input("Enter the message: ")
key=input("Enter the key in alphabets: ")
row(msg,key)
```

- **Decryption:**

```
import math

def row(s,key):

    # to remove repeated alphabets in key
    temp=[]
    for i in key:
        if i not in temp:
            temp.append(i)
    k=""
    for i in temp:
        k+=i
    print("The key used for encryption is: ",k)

    arr=[['' for i in range(len(k))]
         for j in range(int(len(s)/len(k)))]

    # To get indices as the key numbers instead of alphabets in the key, according
    # to algorithm, for appending the elementsof matrix formed earlier, column wise.
    kk=sorted(k)

    d=0
    # arranging the cipher message into matrix
    # to get the same matrix as in encryption
    for i in kk:
        h=k.index(i)
        for j in range(len(k)):
            arr[j][h]=s[d]
            d+=1
```

```
    print("The message matrix is: ")
    for i in arr:
        print(i)

    # the plain text
    plain_text=""
    for i in arr:
        for j in i:
            plain_text+=j
    print("The plain text is: ",plain_text)


msg=input("Enter the message to be decrypted: ")
key=input("Enter the key in alphabets: ")
row(msg,key)
```

➢ **OUTPUT:**



```
PS D:\CLG\SEM-6\Prac\SEM-6\CNS> python -u "d:\CLG\SEM-6\Prac\SEM-6\CNS\Prac-8\Colu
Enter the message: malaythakkar
Enter the key in alphabets: hack
The key used for encryption is:  hack
The message matrix is:
['m', 'a', 'l', 'a']
['y', 't', 'h', 'a']
['k', 'k', 'a', 'r']
['_', '_', '_', '_']
The cipher text is:  atk_lha_myk_aar_
PS D:\CLG\SEM-6\Prac\SEM-6\CNS> ^C
PS D:\CLG\SEM-6\Prac\SEM-6\CNS> python -u "d:\CLG\SEM-6\Prac\SEM-6\CNS\Prac-8\Colu
Enter the message to be decrypted: atk_lha_myk_aar_
Enter the key in alphabets: hack
The key used for encryption is:  hack
The message matrix is:
['m', 'a', 'l', 'a']
['y', 't', 'h', 'a']
['k', 'k', 'a', 'r']
['_', '_', '_', '_']
The plain text is:  malaythakkar____
PS D:\CLG\SEM-6\Prac\SEM-6\CNS>
Ln 62, Col 4 (1637 selected)    Sp
```

Practical-9

AIM: Implementation of diffie-hellman ford.

- ## CODE:

```
 print ("Both parties agree to a single prime")
prime=int(input("Enter the prime number to be considered: "))

# Primitive root to be used use
print ("Both must agree with single primitive root to use")
root=int(input("Enter the primitive root: "))

# Party1 chooses a secret number
alicesecret=int(input("Enter a secret number for Party1: "))

# Party2 chooses a secret number (bs)
bobsecret=int(input("Enter a secret number for Party2: "))
print("\n")

# Party1 public key A=(root^alicesecret)*mod(prime)

print ("Party1's  public key -> A = root^alicesecre*mod(prime))")
alicepublic=(root**alicesecret)%prime
print ("Party1 public key is: ",alicepublic, "\n")


# Party2 public key B=(root^bobsecret)*mod(prime)
print ("Party2's public key -> B = root^bobsecret*)mod(prime))")
bobpublic=(root**bobsecret)%prime
print ("Party2 public key is", bobpublic, "\n")

# Party1 and Party2 exchange their public keys
# Eve(attacker) nows both parties public keys

# Party1 now calculates the shared key K:
# K = B^(alicesecret)*mod(prime)
print ("Party1 calculates the shared key as K=B^alicesecret*(mod(prime))")
alicekey=(bobpublic**alicesecret)%prime
print ("Party1 calculates the shared key and results: ",alicekey, "\n")

# Party2 calculates the shared key K:
# K = A^(bobsecret)*mod(prime)
print ("Party2 calculates the shared key as K = A^bobsecret*(mod(prime))")
bobkey =(alicepublic**bobsecret)%prime
print ("Party2 calculates the shared key and gets", bobkey, "\n")
```

#Both Alice and Bob now share a key which Eve cannot calculate
print ("Attacker does not know the shared private key that Party1 & Party2 can now use")

➢ OUTPUT:

```
PS D:\CLG\SEM-6\Prac\SEM-6\CNS> python -u "d:\CLG\SEM-6\Prac\SEM-6\CNS\Prac-9\Prac9.py
Both parties agree to a single prime
Enter the prime number to be considered: 7
Both must agree with single primitive root to use
Enter the primitive root: 5
Enter a secret number for Party1: 3
Enter a secret number for Party2: 2


Party1's  public key -> A = root^alicesecre*mod(prime))
Party1 public key is:  6

Party2's public key -> B = root^bobsecret*)mod(prime))
Party2 public key is 4

Party1 calculates the shared key as K=B^alicesecret*(mod(prime))
Party1 calculates the shared key and results:  1

Party2 calculates the shared key as K = A^bobsecret*(mod(prime))
Party2 calculates the shared key and gets 1

Attacker does not know the shared private key that Party1 & Party2 can now use
PS D:\CLG\SEM-6\Prac\SEM-6\CNS> 
Ln 46, Col 1 (1623 selected)   Spaces: 4
```

Practical-10

AIM: Implications of AES Algorithms.

- **CODE:**

```
 import math
import string

main=string.ascii_lowercase
# Pseudo code for GCD is
'''
function gcd(a, b)
    if b = 0
        return a
    else
        return gcd(b, a mod b)
'''

# Naive method finding the multiplicative inverse of two numbers
def multiplicative_inverse(a, m):
    a=a%m;
    for x in range(1,m) :
        if((a*x)%m==1) :
            return x
    return 1

# Fast modular exponentiation function (can be used when
# something is very large for calculation modular)
'''
def get_mod_expo(base,exponent,modulus):
    result=1
    while exponent:
        d=exponent%2
        exponent=exponent//2
        if d:
            result=result*base%modulus
        base=base*base%modulus
    return result
'''
# Function to generate a public and private key pair
def generate_keypair(p, q):
    n=p*q
    print("Value of n: ",n)

    # Phi is the Euler's totient of n
    phi = (p-1)*(q-1)
    print("Value of phi(n): ", phi)
```

```
    # Choose an integer e such that e and phi(n) are co-prime
    # e = random.randrange(1, phi) for random pick
    print("Enter e such that is co-prime to ", phi,": ")
    e=int(input())

    # Using Euclid's Algorithm to verify that e and phi(n) are co-prime
    # THe built in function gcd helps with the same
    g=math.gcd(e,phi)
    while(g!=1):
        print("The number you entered is not co-prime")
        e=int(input())
        g=math.gcd(e,phi)

    print("Value of exponent(e) entered is: ", e)

    # To generate the private key
    d = multiplicative_inverse(e, phi)
    # We can use Extended Euclidean Algorithm because
    # we know that e and phi are coprimes

    # Public key is (e, n) and private key is (d, n)
    return (e,n),(d,n)

# Function to Encrypt the message
def encrypt(public_key, to_encrypt):
    key, n = public_key

    # we can also use fast modular exponentiation here
    cipher=pow(to_encrypt,key)%n
    return cipher

# Function to Decrypt the message
def decrypt(private_key, to_decrypt):
    key, n = private_key

    # we can also use fast modular exponentiation here
    decrypted=pow(to_decrypt,key)%n
    return decrypted

# Main Program
# primes of 8 bits in length in binary
p=int(input("Enter prime p: "))
q=int(input("Enter prime q (!=p): "))

# to make sure that p not equal to q while generating randomly
while(p==q):
    p=int(input("Enter prime p: "))
```

```
    q=int(input("Enter prime q (!=p): "))
print("Prime number p: ",p)
print("Prime number q: ",q)
print("Generating Public/Private key-pairs!")
public, private = generate_keypair(p, q)
print("Your public key is (e,n) ", public)
print("Your private key is (d,n) ", private)

message = input("Enter the message: ")
# converting into lower case and removing spaces
message=message.replace(" ","")
message=message.lower()
arr=[]
cipher_text=[]
for i in message:
    if i in main:
        arr.append(main.index(i))
for i in arr:
    cipher_text.append(encrypt(public,i))

print("Encrypted message (Cipher Text): ",cipher_text)

plain=[]
for i in cipher_text:
    plain.append(decrypt(private,i))
plain_text=''
for i in plain:
    plain_text=plain_text+main[i]
print("Plain text array: ",plain)
print("Decrypted message (Plain Text): ", plain_text)
```

➢ OUTPUT:

```
PS D:\CLG\SEM-6\Prac\SEM-6\CNS> python -u "d:\CLG\SEM-6\Prac\SEM-6\CNS\Prac-10\Prac10.py"
Enter prime p: 157
Enter prime q (!=p): 149
Prime number p:  157
Prime number q:  149
Generating Public/Private key-pairs!
Value of n:  23393
Value of phi(n):  23088
Enter e such that is co-prime to  23088 :
23
Value of exponent(e) entered is:  23
Your public key is (e,n)  (23, 23393)
Your private key is (d,n)  (6023, 23393)
Enter the message: malay thakkar
Encrypted message (Cipher Text):  [1871, 0, 15143, 0, 20078, 8898, 6686, 0, 11117, 11117, 0, 8290]
Plain text array:  [12, 0, 11, 0, 24, 19, 7, 0, 10, 10, 0, 17]
Decrypted message (Plain Text):  malaythakkar
PS D:\CLG\SEM-6\Prac\SEM-6\CNS>
```

Ln 1, Col 1 (3209 selected)   Spaces: 4   UTF-8   LF

❖