KADI SARVA VISHWAVIDYALAYA

LDRP INSTITUTE OF TECHNOLOGY AND RESEARCH

GANDHINAGAR

Department of

Information Technology

Subject: Information Security (IT602-N)

Laboratory Manual

Prepared By

Patel Joy J

19BEIT30033

6th-IT

LDRP INSTITUTE OF TECHNOLOGY AND RESEARCH GANDHINAGAR

DEPARTMENT OF

INFORMATION TECHNOLOGY

CERTIFICATE

Mr./Miss_____

of 6th IT Enrollment Number _____

Exam No,_____has satisfactorily completed his/her term work in ***Information Security (IT602-N)*** for the term ending in **Dec-May-2022**.

Date: _____

_____ Dr. Mehul Barot

Subject Coordinator **HOD-IT**

Index

Lab Practical 1

AIM : To implement Caesar Cipher Encryption – Decryption.

Program:

```
Message = input("Enter Your Message = ")
Key = int(input("Enter your Key Value = "))

list1 = ["a" , "b" , "c" , "d" , "e" , "f" , "g" , "h" , "i" , "j" , "k", "l" , "m" , "n" , "o" , "p" , "q" ,
"r" , "s" , "t" , "u" , "v" , "w" , "x" , "y" , "z"]
list2 = ["A" , "B" , "C" , "D" , "E" , "F" , "G" , "H" , "I" , "J" , "K", "L" , "M" , "N" , "O" ,
"P" , "Q" , "R" , "S" , "T" , "U" , "V" , "W" , "X" , "Y" , "Z"]

Encrypted_Message = ""
Decrypted_Message = ""

#Encryption
for char in Message:
    if(char == " "):
        Encrypted_Message += char
    elif(char in list1):
        index = list1.index(char)

        Index = ((index + Key) % 26)

        New_Char = list1[Index]

        Encrypted_Message  = Encrypted_Message + New_Char
    elif(char in list2):
        index = list2.index(char)

        Index = ((index + Key) % 26)

        New_Char = list2[Index]

        Encrypted_Message = Encrypted_Message + New_Char

print("Encrypted Message is : "+Encrypted_Message)

#Decryption
for char in Encrypted_Message:
    if(char == " "):
        Decrypted_Message += char
    elif(char in list1):
        index = list1.index(char)

        Index = ((index - Key) % 26)
```

```
        New_Char = list1[Index]

        Decrypted_Message = Decrypted_Message + New_Char
    elif(char in list2):
        index = list2.index(char)

        Index = ((index - Key) % 26)

        New_Char = list2[Index]

        Decrypted_Message = Decrypted_Message + New_Char

print("Decyrpted or Original Message is : "+Decrypted_Message)
```

OUTPUT:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\JOY PATEL> python -u "c:\Users\JOY PATEL\Desktop\Untitled-1.py"
Enter Your Message = My name is Joy
Enter your Key Value = 3
Encrypted Message is : Pb qdph lv Mrb
Decyrpted or Original Message is : My name is Joy
PS C:\Users\JOY PATEL>
```

Lab Practical 2

AIM : To implement Hill Cipher Encryption.

Program:

```
print("------------------------------------------------------")
#input from user
n = int(input("Enter the value of dimension of matrix = "))
print("------------------------------------------------------")

list1 = []
key = []

print("Enter your key values row by row : ")
print("------------------------------------------------------")

#key values from users
for i in range(0,n*n):
    item = int(input("Enter your value = "))
    print("---- ---------------------- ")
    list1.append(item)

#make a key matrix
for i in range(0,n):
    list2 = []
    for j in range(0,n):
        list2.append(list1[3 * i + j])
    key.append(list2)

Plain_Text = input("Enter Your Message = ")
Plain_Text = Plain_Text.lower()
Plain_Text = Plain_Text.replace(" ","")

Character1 = ["a" , "b" , "c" , "d" , "e" , "f" , "g" , "h" , "i" , "j" , "k", "l" , "m" , "n" , "o" , "p"
, "q" , "r" , "s" , "t" , "u" , "v" , "w" , "x" , "y" , "z"]

if(len(Plain_Text) % 3 != 0):
    Plain_Text = Plain_Text + "x"

list3 = []
Plain_Text_list = []

#Plain text in converted into list
for item in Plain_Text:
    if(item in Character1):
        index = Character1.index(item)
        Plain_Text_list.append(index)
```

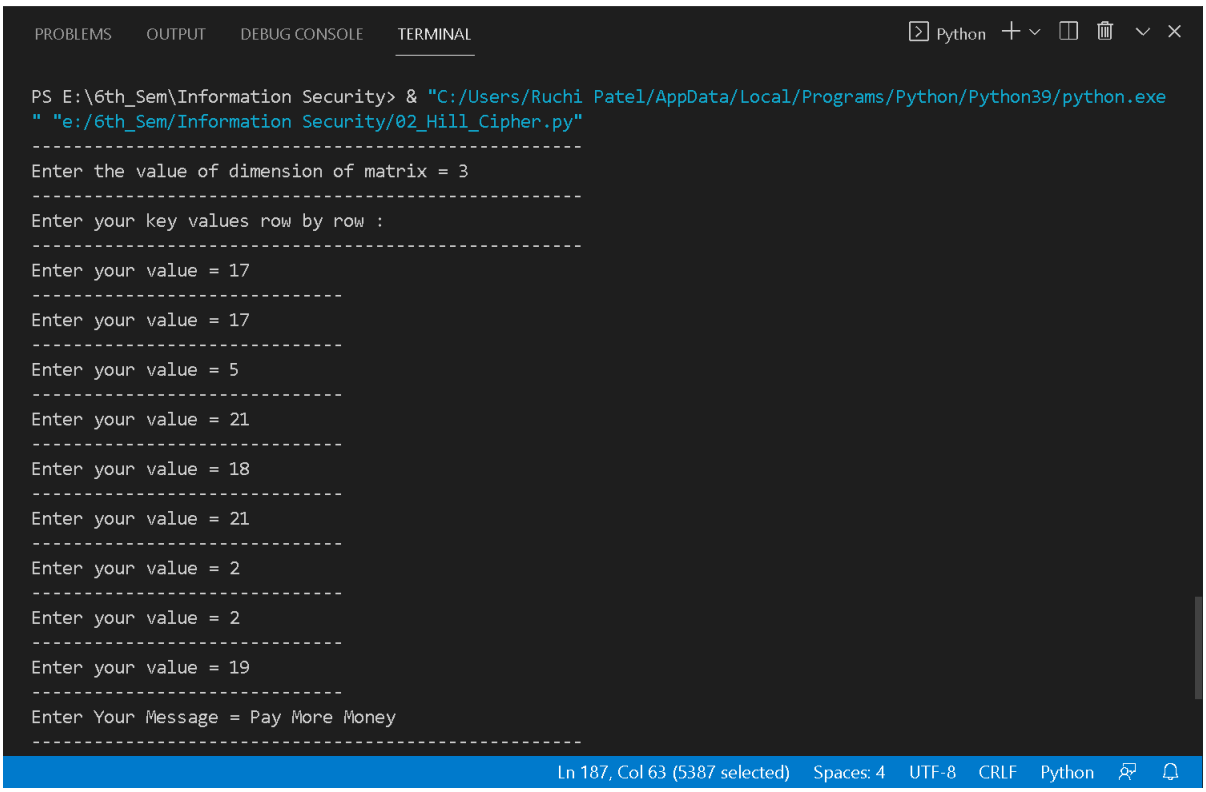
```
Cipher_Text_list = []
count = 0
k = 0

#convert plain text into cipher text
while(count != (len(Plain_Text_list)/3)):

    for i in range(0,3):
        index = 0
        for j in range(0,3):
            index = index + ((key[i][j] * Plain_Text_list[k]))
            k = k + 1
        index = index % 26
        Cipher_Text_list.append(index)
        k = k - 3

    k = 3*(count + 1)
    count = count + 1

Cipher_Text_Message = ""

for item in Cipher_Text_list:
    for i in range(0,25):
        if(item == i):
            Cipher_Text_Message += Character1[i]

#print cipher text
print("------------------------------------------")
print("Cipher Text is = ",Cipher_Text_Message.upper())

#find the determinate
for i in range(0,n):
    if(i == 0):
        ans1 = key[0][i]*(key[1][(i+1)%3]*key[2][(i+2)%3] -
key[1][(i+2)%3]*key[2][(i+1)%3])
    elif(i == 1):
        ans2 = key[0][i]*(key[1][(i+1)%3]*key[2][(i+2)%3] -
key[1][(i+2)%3]*key[2][(i+1)%3])
        if(ans2 < 0):
            ans2 = ans2*(-1)
    elif(i == 2):
        ans3 = key[0][i]*(key[1][(i+1)%3]*key[2][(i+2)%3] -
key[1][(i+2)%3]*key[2][(i+1)%3])

determinate = (ans1 - ans2 + ans3) % 26

#Inverse of determinate
```

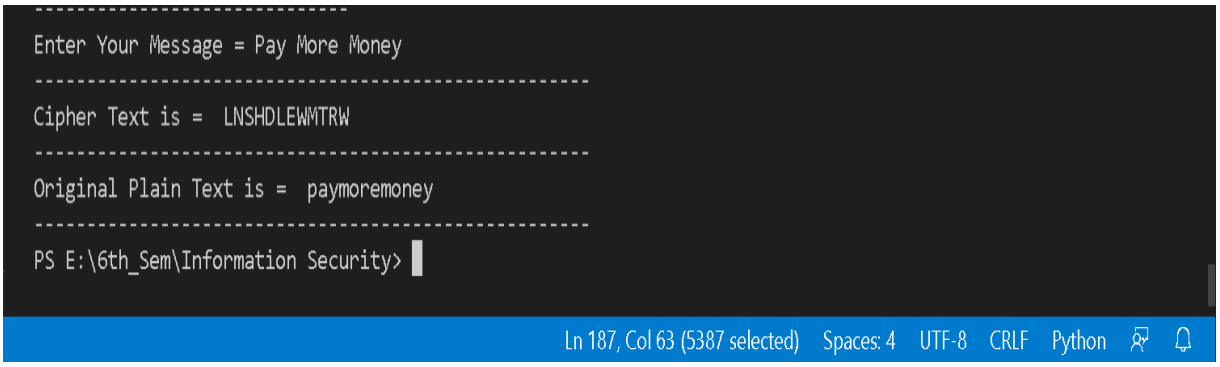
```
Inverse_of_determinate = 0
value = 0

for i in range(1,26):
    value = (i * determinate) % 26
    if(value == 1):
        Inverse_of_determinate = i
        break

#convert key into transpose key
Transpose_key = []

for i in range(0,n):
    list2 = []
    for j in range(0,n):
        list2.append(key[j][i])
    Transpose_key.append(list2)

#find the adjoint matrix
Adj_key = []

for i in range(0,n):
    list2 = []
    for j in range(0,n):
        item = ((Transpose_key[(i+1)%3][(j+1)%3] * Transpose_key[(i+2)%3][(j+2)%3]) -
(Transpose_key[(i+1)%3][(j+2)%3]*Transpose_key[(i+2)%3][(j+1)%3]))
        if(i == 0 and j == 1):
            if(item < 0):
                item = item * -1
            list2.append(item)
        elif(i == 1 and j == 0):
            if(item < 0):
                item = item * -1
            list2.append(item)
        elif(i == 2 and j == 1):
            if(item < 0):
                item = item * -1
            list2.append(item)
        elif(i == 1 and j == 2):
            if(item < 0):
                item = item * -1
            list2.append(item)
        else:
            list2.append(item)
    Adj_key.append(list2)

for i in range(0,n):
    for j in range(0,n):
```

```
        if(i == 0 and j == 1):
            Adj_key[i][j] = (Adj_key[i][j])*(-1)
            Adj_key[i][j] = (Adj_key[i][j]) % 26
        elif(i == 1 and j == 0):
            Adj_key[i][j] = (Adj_key[i][j])*(-1)
            Adj_key[i][j] = (Adj_key[i][j]) % 26
        elif(i == 2 and j == 1):
            Adj_key[i][j] = (Adj_key[i][j])*(-1)
            Adj_key[i][j] = (Adj_key[i][j]) % 26
        elif(i == 1 and j == 2):
            Adj_key[i][j] = (Adj_key[i][j])*(-1)
            Adj_key[i][j] = (Adj_key[i][j]) % 26
        else:
            Adj_key[i][j] = (Adj_key[i][j]) % 26


#lastly find the inverse of key(matrix)
Inverse_key = []

for i in range(0,n):
    list2 = []
    for j in range(0,n):
        Adj_key[i][j] = (Adj_key[i][j] * Inverse_of_determinate) % 26
        list2.append(Adj_key[i][j])
    Inverse_key.append(list2)

#get the original message back
Original_Message = ""
Original_Message_list = []
count = 0
k = 0

while(count != (len(Cipher_Text_list)/3)):

    for i in range(0,3):
        index = 0
        for j in range(0,3):
            index = index + ((Inverse_key[i][j] * Cipher_Text_list[k]))
            k = k + 1
        index = index % 26
        Original_Message_list.append(index)
        k = k - 3

    k = 3*(count + 1)
    count = count + 1

for item in Original_Message_list:
    for i in range(0,25):
```

```
        if(item == i):
            Original_Message += Character1[i]

print("------------------------------------------------")
print("Original Plain Text is = ",Original_Message)
print("------------------------------------------------")
```

OUTPUT:





Lab Practical 3

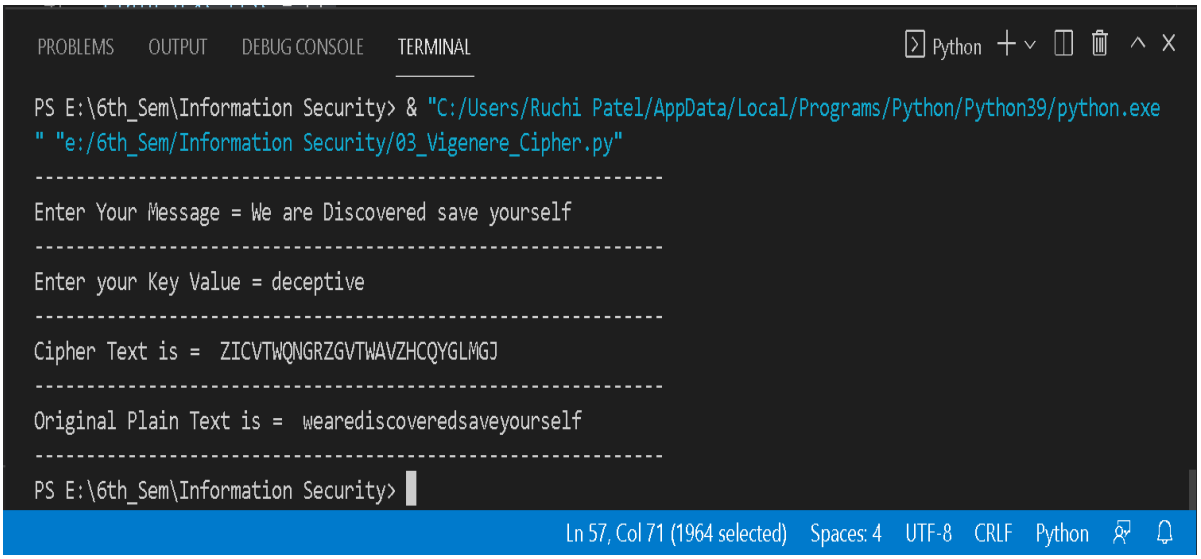
AIM : To implement Poly-alphabetic Cipher (Vigener Cipher) Technique.

Program:

```
print("------------------------------------------------------")
Plain_Text = input("Enter Your Message = ")
print("------------------------------------------------------")
Key = input("Enter your Key Value = ")
print("------------------------------------------------------")

list1 = ["a" , "b" , "c" , "d" , "e" , "f" , "g" , "h" , "i" , "j" , "k", "l" , "m" , "n" , "o" , "p" , "q" ,
"r" , "s" , "t" , "u" , "v" , "w" , "x" , "y" , "z"]
list2 = ["A" , "B" , "C" , "D" , "E" , "F" , "G" , "H" , "I" , "J" , "K", "L" , "M" , "N" , "O" ,
"P" , "Q" , "R" , "S" , "T" , "U" , "V" , "W" , "X" , "Y" , "Z"]

Plain_Text = Plain_Text.replace(" ","")
Key = Key.replace(" ","")

Plain_Text_list = []
key_list = []

for item in Plain_Text:
    if(item in list1):
        Plain_Text_list.append(list1.index(item))
    elif(item in list2):
        Plain_Text_list.append(list2.index(item))

for item in Key:
    if(item in list1):
        key_list.append(list1.index(item))
    elif(item in list2):
        key_list.append(list2.index(item))

Cipher_Text_list = []

for i in range(0,len(Plain_Text_list)):
    Cipher_Text_list.append((Plain_Text_list[i] + key_list[i % len(key_list)]) % 26)

Cipher_Text = ""

for item in Cipher_Text_list:
    Cipher_Text += list2[item]

print("Cipher Text is = ",Cipher_Text)
print("------------------------------------------------------")
```

```
Plain_Text_list = []
Original_Plain_Text_List = []

for item in Cipher_Text:
    if(item in list2):
        Plain_Text_list.append(list2.index(item))

for i in range(0,len(Cipher_Text_list)):
    Original_Plain_Text_List.append((Cipher_Text_list[i] - key_list[i % len(key_list)]) % 26)

Original_Plain_Text = ""

for item in Original_Plain_Text_List:
    Original_Plain_Text += list1[item]

print("Original Plain Text is = ",Original_Plain_Text)
print("------------------------------------------------------")
```

OUTPUT:



Lab Practical 4

AIM : To implement Play-Fair Cipher Technique.

Program:

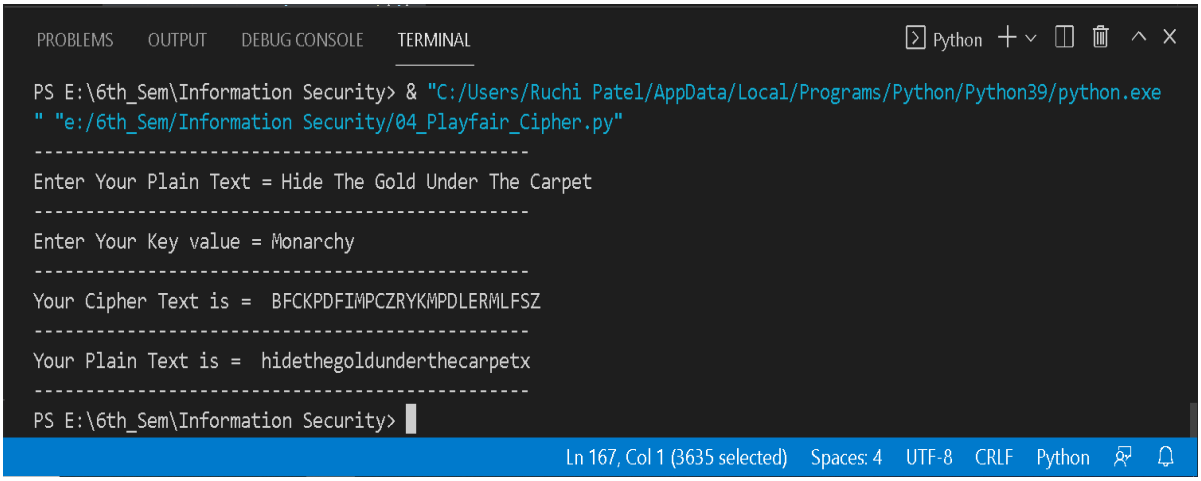
```
def function(char,Matrix1):

    i1 = []

    if(char == "j"):
        char = "i"

    for i in range(0,5):
        for j in range(0,5):
            if(char == Matrix1[i][j]):
                i1.append(i)
                i1.append(j)

    return i1

print("---- ----------▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬ .....")
Plain_Text = input("Enter Your Plain Text = ")
print("---- ----------▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬ .....")
Key = input("Enter Your Key value = ")
print("---- ----------▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬ .....")
Key = Key.lower()
Plain_Text = Plain_Text.lower()
Key = Key.replace(" ","")
Plain_Text = Plain_Text.replace(" ","")

Character1 = ["a" , "b" , "c" , "d" , "e" , "f" , "g" , "h" , "i" , "k", "l" , "m" , "n" , "o" , "p" , "q"
, "r" , "s" , "t" , "u" , "v" , "w" , "x" , "y" , "z"]
Character2 = ["A" , "B" , "C" , "D" , "E" , "F" , "G" , "H" , "I" , "K", "L" , "M" , "N" , "O" ,
"P" , "Q" , "R" , "S" , "T" , "U" , "V" , "W" , "X" , "Y" , "Z"]

Matrix = []
list1 = []

for item in Key:
    if(not item in list1):
        list1.append(item)

for i in range(0,25):
    if(not Character1[i] in list1):
        list1.append(Character1[i])

for i in range(0,5):
```

```
    list2 = []
    for j in range(0,5):
        list2.append(list1[5 * i + j])
    Matrix.append(list2)

list3 = []

for item in Plain_Text:
    list3.append(item)

i = 0

while(i < (len(list3) - 1)):
    str1 = list3[i]
    str2 = list3[i+1]

    if(str1 == str2):
        list3.insert(i+1,"x")

    i = i + 2

# print(Matrix)
# print(list3)

Plain_Text = ""

for item in list3:
    Plain_Text += item

if(len(Plain_Text) % 2 != 0):
    Plain_Text += "x"

# print(Plain_Text)

Cipher_Text = ""

d = 0

while(d < len(Plain_Text)):

    a1  = function(Plain_Text[d] , Matrix)
    a2 = function(Plain_Text[d+1] , Matrix)

    # print(a1,a2)

    if(a1[1] == a2[1]):
        x1 = (a1[0] + 1) % 5
        y1 = a1[1]
```

```
        x2 = (a2[0] + 1) % 5
        y2 = a2[1]

        Cipher_Text += (Matrix[x1][y1])
        Cipher_Text += (Matrix[x2][y2])
    elif(a1[0] == a2[0]):
        x1 =  a1[0]
        y1 = (a1[1] + 1) % 5

        x2 = a2[0]
        y2 = (a2[1] + 1) % 5

        Cipher_Text += (Matrix[x1][y1])
        Cipher_Text += (Matrix[x2][y2])
    else:
        x1 = a1[0]
        y1 = a1[1]

        x2 = a2[0]
        y2 = a2[1]

        Cipher_Text += (Matrix[x1][y2])
        Cipher_Text += (Matrix[x2][y1])

    d = d + 2

Cipher_Text = Cipher_Text.upper()
print("Your Cipher Text is = ",Cipher_Text)
print("---- ----------▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬ .....")
Cipher_Text = Cipher_Text.lower()
Plain_Text = ""

d = 0
while(d < len(Cipher_Text)):

    a1  = function(Cipher_Text[d] , Matrix)
    a2 = function(Cipher_Text[d+1] , Matrix)

    # print(a1,a2)

    if(a1[1] == a2[1]):
        x1 = (a1[0] - 1) % 5
        y1 = a1[1]

        x2 = (a2[0] - 1) % 5
        y2 = a2[1]
```

```
        Plain_Text += (Matrix[x1][y1])
        Plain_Text += (Matrix[x2][y2])
    elif(a1[0] == a2[0]):
        x1 =  a1[0]
        y1 = (a1[1] - 1) % 5

        x2 = a2[0]
        y2 = (a2[1] - 1) % 5

        Plain_Text += (Matrix[x1][y1])
        Plain_Text += (Matrix[x2][y2])
    else:
        x1 = a1[0]
        y1 = a1[1]

        x2 = a2[0]
        y2 = a2[1]

        Plain_Text += (Matrix[x1][y2])
        Plain_Text += (Matrix[x2][y1])

    d = d + 2


print("Your Plain Text is = ",Plain_Text)
print("---- ----------------------------------- .....")
```

OUTPUT:



Lab Practical 5

AIM : Write a program to implement Rail-Fence Encryption Technique

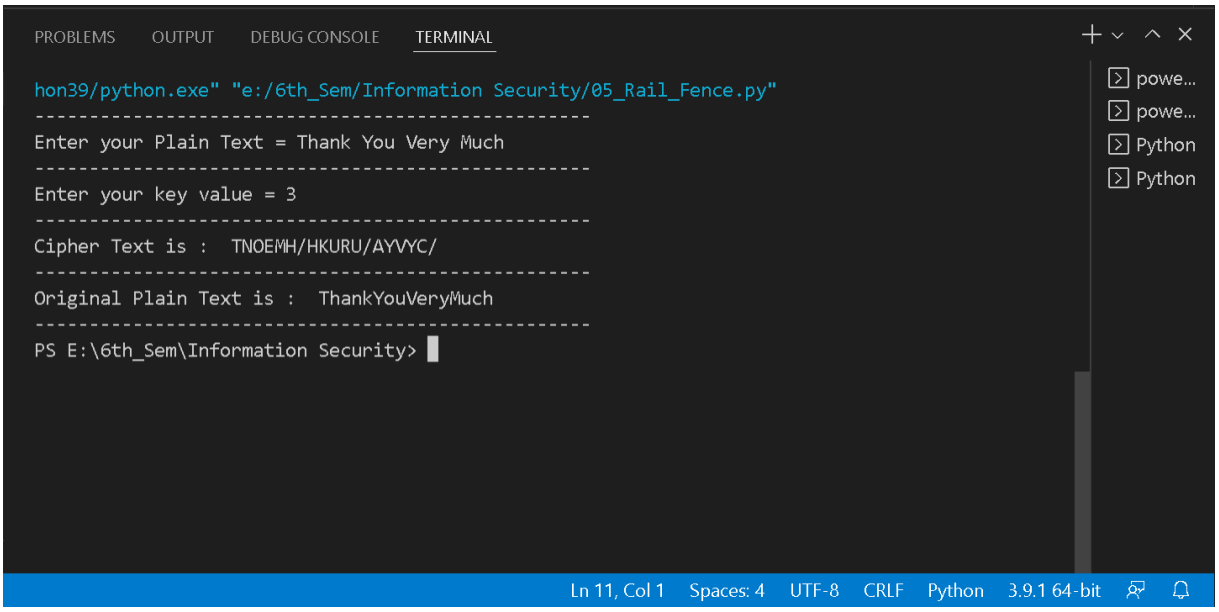
Program:

```
print(" ------------------------------------------------- ")
plain_text = input("enter your  plain text =  ")
print(" ------------------------------------------------- ")
key = int(input("enter your key value = "))
print(" ------------------------------------------------- ")

#encryption
plain_text = plain_text.replace(" ","")

list1 = []

for item in plain_text:
   list1.append(item)

for i in range(0,key):
   list1.append("\n")

cipher_text_list = []

for i in range(0,key):
   j = i
   while(j != len(list1)):
      if(list1[j] != "\n"):
         cipher_text_list.append(list1[j])
         j = j + key
      else:
         if(len(plain_text)%key != 0):
            cipher_text_list.append("/")
         break

cipher_text = ""

for item in cipher_text_list:
   cipher_text += item

print("cipher text is : ",cipher_text.upper())
print(" ------------------------------------------------- ")

#decryption
list2 = []
```

```
for i in range(len(cipher_text)):
    if(cipher_text[i] != "/"):
        list2.append(cipher_text[i])

val = int(len(list2)%key)

if(val == 0):
    new_key = int(len(list2)/key)
else:
    new_key = int(len(list2)/key) + 1

for i in range(0,key):
    cipher_text_list.append("\n")

plain_text_list = []

i = 0
k = 0
count = 0

while(i != len(cipher_text)):

    j = 0
    while(j != key):
        if(cipher_text_list[k] != "\n"):
            if(val == 0):
                plain_text_list.append(cipher_text_list[k])
                k = k + new_key
                j = j + 1
            else:
                if(j == 0):
                    plain_text_list.append(cipher_text_list[k])
                else:
                    plain_text_list.append(cipher_text_list[k+1])
                k = k + new_key
                j = j + 1
        else:
            break

    i = i + key
    count = count + 1
    k = count
    if(k == new_key):
        break

original_plain_text = ""

for i in range(len(plain_text_list)):
```

```
    if(plain_text_list[i] != "/"):
        original_plain_text += plain_text_list[i]

print("original plain text is : ",original_plain_text)
print(" ------------------------------------------------- ")
```

OUTPUT:



Lab Practical 6

AIM : To implement S-DES algorithm for data encryption.

Program:

```
print("------------------------------------------------")
# Hexadecimal to binary conversion
def hex2bin(s):
    mp = {'0' : "0000",
          '1' : "0001",
          '2' : "0010",
          '3' : "0011",
          '4' : "0100",
          '5' : "0101",
          '6' : "0110",
          '7' : "0111",
          '8' : "1000",
          '9' : "1001",
          'A' : "1010",
          'B' : "1011",
          'C' : "1100",
          'D' : "1101",
          'E' : "1110",
          'F' : "1111" }
    bin = ""
    for i in range(len(s)):
        bin = bin + mp[s[i]]
    return bin

# Binary to hexadecimal conversion
def bin2hex(s):
    mp = {"0000" : '0',
          "0001" : '1',
          "0010" : '2',
          "0011" : '3',
          "0100" : '4',
          "0101" : '5',
          "0110" : '6',
          "0111" : '7',
          "1000" : '8',
          "1001" : '9',
          "1010" : 'A',
          "1011" : 'B',
          "1100" : 'C',
          "1101" : 'D',
          "1110" : 'E',
```

```
        "1111" : 'F' }
    hex = ""
    for i in range(0,len(s),4):
        ch = ""
        ch = ch + s[i]
        ch = ch + s[i + 1]
        ch = ch + s[i + 2]
        ch = ch + s[i + 3]
        hex = hex + mp[ch]

    return hex

# Binary to decimal conversion
def bin2dec(binary):

    binary1 = binary
    decimal, i, n = 0, 0, 0
    while(binary != 0):
        dec = binary % 10
        decimal = decimal + dec * pow(2, i)
        binary = binary//10
        i += 1
    return decimal

# Decimal to binary conversion
def dec2bin(num):
    res = bin(num).replace("0b", "")
    if(len(res)%4 != 0):
        div = len(res) / 4
        div = int(div)
        counter =(4 * (div + 1)) - len(res)
        for i in range(0, counter):
            res = '0' + res
    return res

# Permute function to rearrange the bits
def permute(k, arr, n):
    permutation = ""
    for i in range(0, n):
        permutation = permutation + k[arr[i] - 1]
    return permutation

# shifting the bits towards left by nth shifts
def shift_left(k, nth_shifts):
    s = ""
    for i in range(nth_shifts):
        for j in range(1,len(k)):
            s = s + k[j]
```

```
        s = s + k[0]
        k = s
        s = ""
    return k

# calculating xow of two strings of binary number a and b
def xor(a, b):
    ans = ""
    for i in range(len(a)):
        if a[i] == b[i]:
            ans = ans + "0"
        else:
            ans = ans + "1"
    return ans

# Table of Position of 64 bits at initial level: Initial Permutation Table
initial_perm = [58, 50, 42, 34, 26, 18, 10, 2,
                60, 52, 44, 36, 28, 20, 12, 4,
                62, 54, 46, 38, 30, 22, 14, 6,
                64, 56, 48, 40, 32, 24, 16, 8,
                57, 49, 41, 33, 25, 17, 9, 1,
                59, 51, 43, 35, 27, 19, 11, 3,
                61, 53, 45, 37, 29, 21, 13, 5,
                63, 55, 47, 39, 31, 23, 15, 7]

# Expansion D-box Table
exp_d = [32, 1 , 2 , 3 , 4 , 5 , 4 , 5,
         6 , 7 , 8 , 9 , 8 , 9 , 10, 11,
         12, 13, 12, 13, 14, 15, 16, 17,
         16, 17, 18, 19, 20, 21, 20, 21,
         22, 23, 24, 25, 24, 25, 26, 27,
         28, 29, 28, 29, 30, 31, 32, 1 ]

# Straight Permutation Table
per = [ 16, 7, 20, 21,
        29, 12, 28, 17,
         1, 15, 23, 26,
         5, 18, 31, 10,
         2,  8, 24, 14,
        32, 27,  3,  9,
        19, 13, 30,  6,
        22, 11,  4, 25 ]

# S-box Table
sbox =  [[[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
          [ 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
          [ 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
          [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 ]],
```

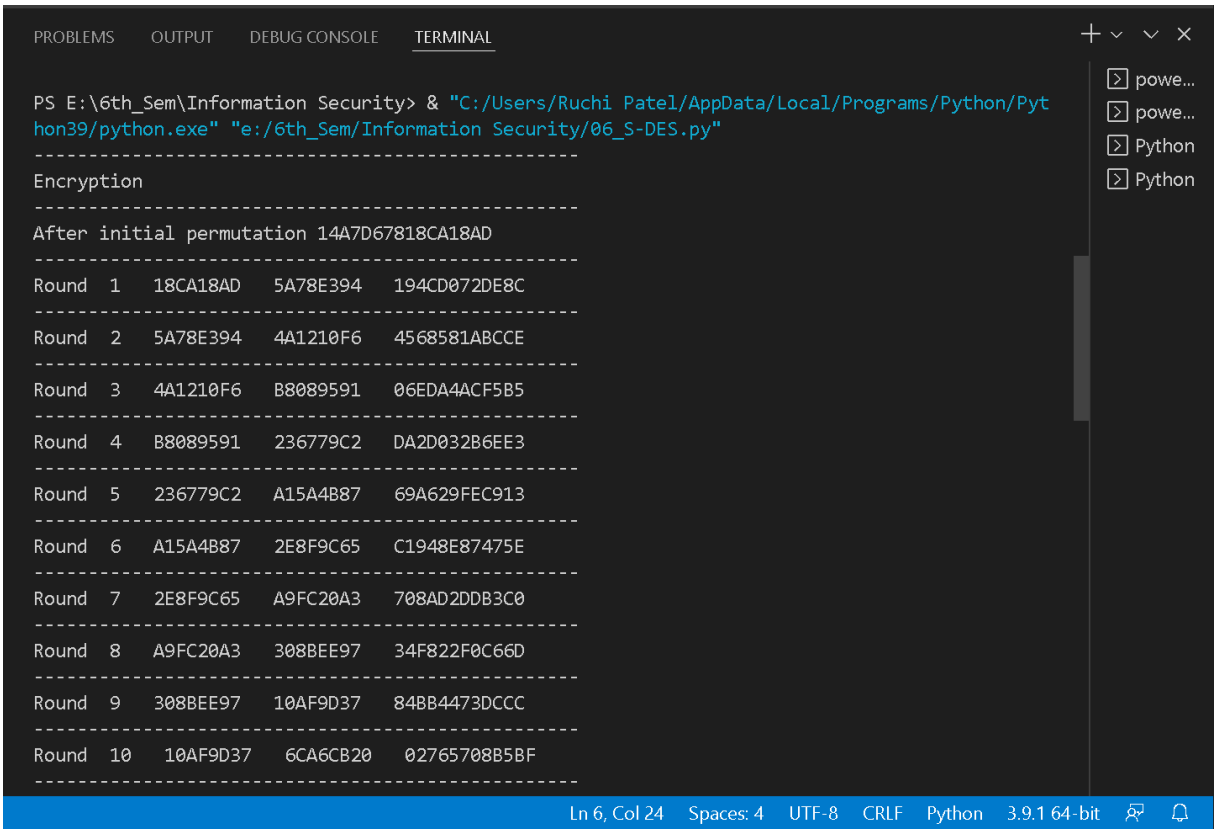
```
        [[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],
          [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],
          [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],
         [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 ]],

        [ [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],
          [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],
          [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],
          [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 ]],

        [ [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],
          [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
          [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],
          [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14] ],

        [ [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],
          [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],
          [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],
          [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 ]],

        [ [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],
          [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],
          [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
          [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13] ],

        [ [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],
          [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
          [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
          [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12] ],

        [ [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],
          [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
          [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],
          [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11] ] ]

# Final Permutation Table
final_perm = [ 40, 8, 48, 16, 56, 24, 64, 32,
               39, 7, 47, 15, 55, 23, 63, 31,
               38, 6, 46, 14, 54, 22, 62, 30,
               37, 5, 45, 13, 53, 21, 61, 29,
               36, 4, 44, 12, 52, 20, 60, 28,
               35, 3, 43, 11, 51, 19, 59, 27,
               34, 2, 42, 10, 50, 18, 58, 26,
               33, 1, 41, 9, 49, 17, 57, 25 ]

def encrypt(pt, rkb, rk):
    pt = hex2bin(pt)
```

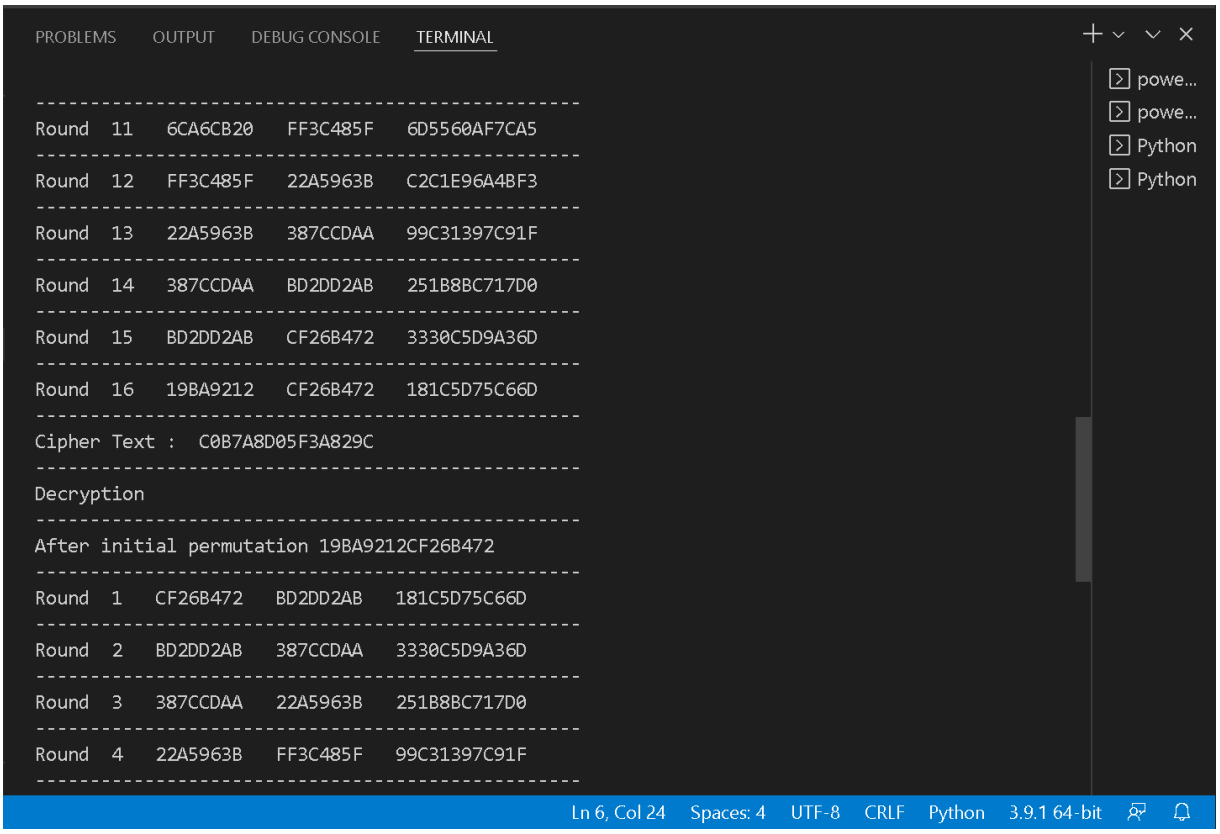
```
    # Initial Permutation
    pt =  permute(pt, initial_perm, 64)
    print("After initial permutation", bin2hex(pt))
    print("------  ----------------------------------  --")

    #  Splitting
    left = pt[0:32]
    right  =  pt[32:64]
    for i in range(0, 16):
        # Expansion D-box: Expanding the 32 bits data into 48 bits
        right_expanded = permute(right, exp_d, 48)

        # XOR RoundKey[i] and right_expanded
        xor_x = xor(right_expanded, rkb[i])

        # S-boxex: substituting the value from s-box table by calculating row and column
        sbox_str = ""
        for j in range(0, 8):
            row = bin2dec(int(xor_x[j * 6] + xor_x[j * 6 + 5]))
            col = bin2dec(int(xor_x[j * 6 + 1] + xor_x[j * 6 + 2] + xor_x[j * 6 + 3] + xor_x[j * 6
+ 4]))
            val = sbox[j][row][col]
            sbox_str = sbox_str + dec2bin(val)

        # Straight D-box: After substituting rearranging the bits
        sbox_str = permute(sbox_str, per, 32)

        # XOR left and sbox_str
        result = xor(left, sbox_str)
        left = result

        # Swapper
        if(i != 15):
            left, right = right, left
        print("Round ", i + 1, " ", bin2hex(left), " ", bin2hex(right), " ", rk[i])
        print("------  ----------------------------------  -----")

    # Combination
    combine = left + right

    # Final permutation: final rearranging of bits to get cipher text
    cipher_text = permute(combine, final_perm, 64)
    return cipher_text

pt = "123456ABCD132536"
key = "AABB09182736CCDD"
```

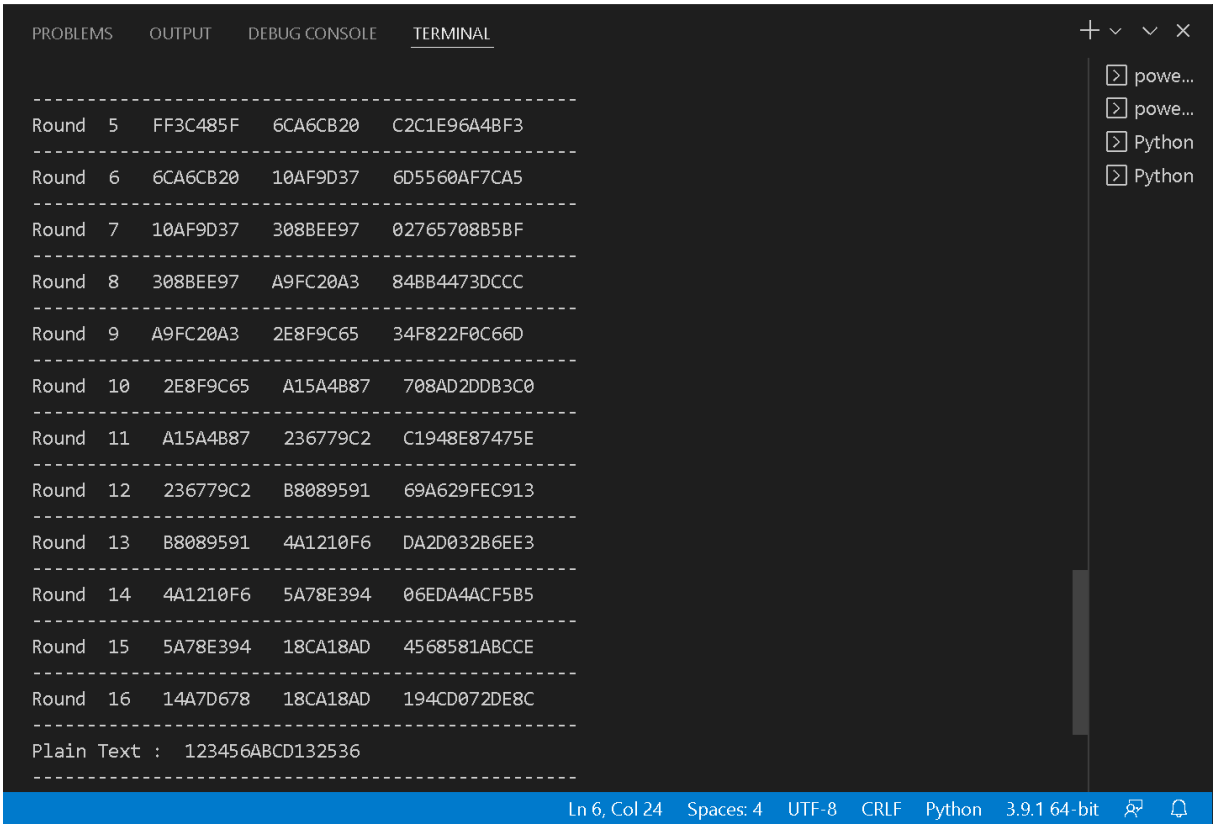
```
# Key generation
# --hex to binary
key = hex2bin(key)

# --parity bit drop table
keyp = [57, 49, 41, 33, 25, 17, 9,
        1, 58, 50, 42, 34, 26, 18,
        10, 2, 59, 51, 43, 35, 27,
        19, 11, 3, 60, 52, 44, 36,
        63, 55, 47, 39, 31, 23, 15,
        7, 62, 54, 46, 38, 30, 22,
        14, 6, 61, 53, 45, 37, 29,
        21, 13, 5, 28, 20, 12, 4 ]

# getting 56 bit key from 64 bit using the parity bits
key = permute(key, keyp, 56)

# Number of bit shifts
shift_table = [1, 1, 2, 2,
               2, 2, 2, 2,
               1, 2, 2, 2,
               2, 2, 2, 1 ]

# Key- Compression Table : Compression of key from 56 bits to 48 bits
key_comp = [14, 17, 11, 24, 1, 5,
            3, 28, 15, 6, 21, 10,
            23, 19, 12, 4, 26, 8,
            16, 7, 27, 20, 13, 2,
            41, 52, 31, 37, 47, 55,
            30, 40, 51, 45, 33, 48,
            44, 49, 39, 56, 34, 53,
            46, 42, 50, 36, 29, 32 ]

# Splitting
left = key[0:28]    # rkb for RoundKeys in binary
right = key[28:56]  # rk for RoundKeys in hexadecimal

rkb = []
rk = []
for i in range(0, 16):
    # Shifting the bits by nth shifts by checking from shift table
    left = shift_left(left, shift_table[i])
    right = shift_left(right, shift_table[i])

    # Combination of left and right string
    combine_str = left + right

    # Compression of key from 56 to 48 bits
```

```
    round_key = permute(combine_str, key_comp, 48)

    rkb.append(round_key)
    rk.append(bin2hex(round_key))

print("Encryption")
print("------------------------------------------------")
cipher_text = bin2hex(encrypt(pt, rkb, rk))
print("Cipher Text : ",cipher_text)
print("------------------------------------------------")

print("Decryption")
print("------------------------------------------------")
rkb_rev = rkb[::-1]
rk_rev = rk[::-1]
text = bin2hex(encrypt(cipher_text, rkb_rev, rk_rev))
print("Plain Text : ",text)
print("------------------------------------------------")
```

OUTPUT:




PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
powe...
powe...
Python
Python
Round 11 6CA6CB20 FF3C485F 6D5560AF7CA5
Round 12 FF3C485F 22A5963B C2C1E96A4BF3
Round 13 22A5963B 387CCDAA 99C31397C91F
Round 14 387CCDAA BD2DD2AB 251B8BC717D0
Round 15 BD2DD2AB CF26B472 3330C5D9A36D
Round 16 19BA9212 CF26B472 181C5D75C66D
Cipher Text : C0B7A8D05F3A829C
Decryption
After initial permutation 19BA9212CF26B472
Round 1 CF26B472 BD2DD2AB 181C5D75C66D
Round 2 BD2DD2AB 387CCDAA 3330C5D9A36D
Round 3 387CCDAA 22A5963B 251B8BC717D0
Round 4 22A5963B FF3C485F 99C31397C91F
Ln 6, Col 24
Spaces: 4
UTF-8
CRLF
Python
3.9.1 64-bit


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
powe...
powe...
Python
Python
Round 5 FF3C485F 6CA6CB20 C2C1E96A4BF3
Round 6 6CA6CB20 10AF9D37 6D5560AF7CA5
Round 7 10AF9D37 308BEE97 02765708B5BF
Round 8 308BEE97 A9FC20A3 84BB4473DCCC
Round 9 A9FC20A3 2E8F9C65 34F822F0C66D
Round 10 2E8F9C65 A15A4B87 708AD2DDB3C0
Round 11 A15A4B87 236779C2 C1948E87475E
Round 12 236779C2 B8089591 69A629FEC913
Round 13 B8089591 4A1210F6 DA2D032B6EE3
Round 14 4A1210F6 5A78E394 06EDA4ACF5B5
Round 15 5A78E394 18CA18AD 4568581ABCCE
Round 16 14A7D678 18CA18AD 194CD072DE8C
Plain Text : 123456ABCD132536
Ln 6, Col 24
Spaces: 4
UTF-8
CRLF
Python
3.9.1 64-bit

Lab Practical 7

AIM : Write a program to implement RSA asymmetric (public key and private key)-Encryption.

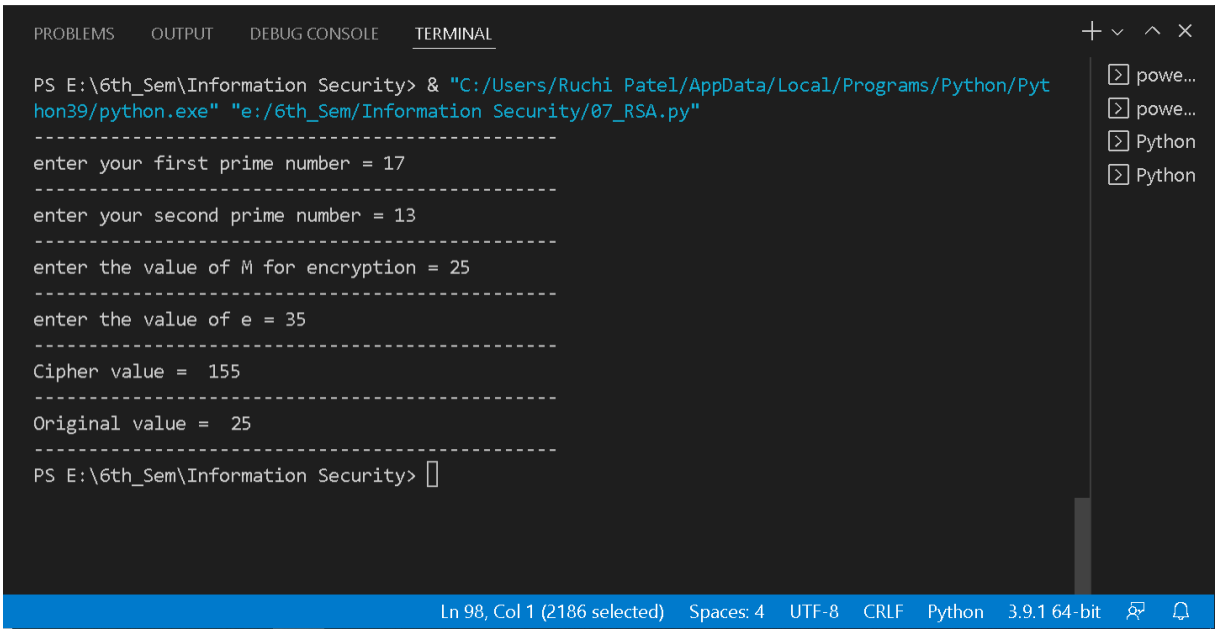
Program:

```
print("---- ---------------------------- -----")
p = int(input("enter your first prime number = "))
print("---- ---------------------------- -----")
q = int(input("enter your second prime number = "))
print("---- ---------------------------- -----")
M = int(input("enter the value of M for encryption = "))
print("------ ---------------------------- -----")

n = p * q

totient_function = (p-1)*(q-1)

e = int(input("enter the value of e = "))
print("---- ---------------------------- -----")

a = (totient_function,e)

while(a[1] != 0):
   b = (a[1],a[0]%a[1])
   a  =  b

flag = True

while(flag):

   condition1 = 0
   condition2 = 0
   condition3 = 0

   if(e > 1):

      condition1 = 1

      if(e < totient_function):

         condition2 = 1

         if(a[0] == 1):

            condition3 = 1
```

```
            flag = False
    else:

        if(condition1 == 0):
            print("Your e value is less than 1 please ,re-enter")
            e = int(input("enter the value of e = "))
            print("-----------------------------------------")

        if(condition2 == 0):
            print("Your e value is greater than tiotent function ,please re-enter")
            e = int(input("enter the value of e = "))
            print("-----------------------------------------")

        if(condition3 == 0):
            print("gcd of e and tiotent function is not 1 ,please re enter")
            e = int(input("enter the value of e = "))
            print("-----------------------------------------")


T1 = 0
T2 = 1

d = (totient_function,e)

while(d[1] != 0):
    Q = int(d[0] / d[1])

    R = d[0] % d[1]

    d = (d[1],R)

    T = T1 - (T2 * Q)
    T1 = T2
    T2 = T

public_key = (e,n)
private_key = (T1,n)
C = 1

for i in range(0,e):

    C = (M * C) % n

print("Cipher value = ",C)
print("---- ----------------------------------")

M = 1
```

```
for i in range(0,T1):

    M = (C * M) % n

print("Original value = ",M)
print("---- ----------———————————————————— .....")
```

OUTPUT:



Lab Practical 8

AIM : Study of MD5 hash function and implement the hash code using MD5.

Program:

```
print("------------------------------------------------------------")
import hashlib

# initializing string
str2hash = input("Enter your input = ")
print("------------------------------------------------------------")

# encoding GeeksforGeeks using encode()
# then sending to md5()
result = hashlib.md5(str2hash.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of hash is : ", end ="")
print(result.hexdigest())
print("------------------------------------------------------------")
```

OUTPUT:

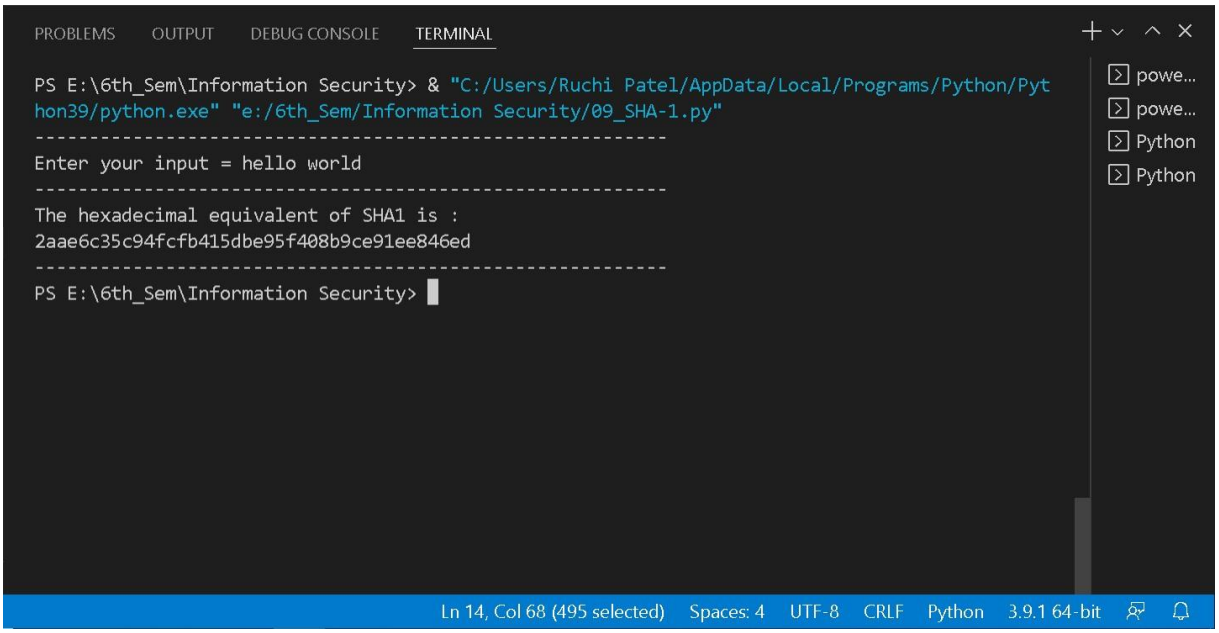
Lab Practical 9

AIM : Study of SHA-1 hash function and implement the hash code using SHA-1.

Program:

```
print("------------------------------------------------------------")
import hashlib

str = input("Enter your input = ")
print("------------------------------------------------------------")

# encoding GeeksforGeeks using encode()
# then sending to SHA1()
result =  hashlib.sha1(str.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA1 is : ")
print(result.hexdigest())
print("------------------------------------------------------------")
```

OUTPUT:



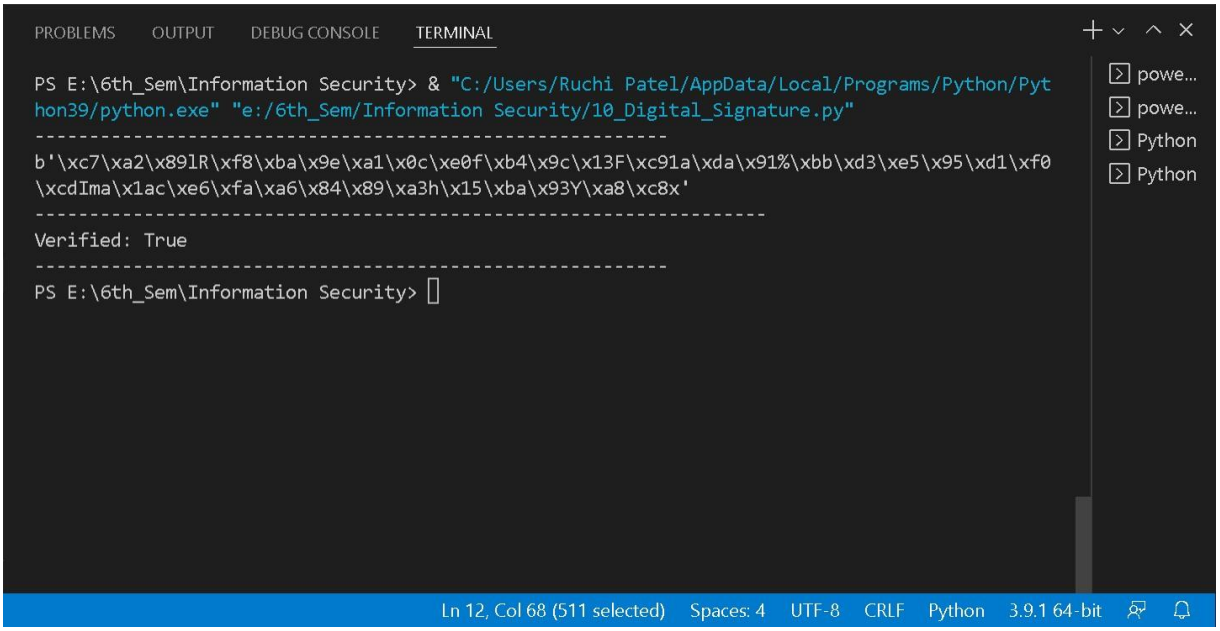
Lab Practical 10

AIM : Write a program to generate digital signature using Hash code.

Program:

```
print("------------------------------------------------------")
from ecdsa import SigningKey

private_key = SigningKey.generate() # uses NIST192p

signature = private_key.sign(b"Educative authorizes this shot")

print(signature)
print("------------------------------------------------------")
public_key = private_key.verifying_key
print("Verified:", public_key.verify(signature, b"Educative authorizes this shot"))
print("------------------------------------------------------")
```

OUTPUT:



Lab Practical 11

AIM : Implement a code to simulate buffer overflow attack.

Program:

```
buffer = [None]*10
flag = 0

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

for i in range(0,len(password)):
    buffer[i] = (password[i])


if(password == "12345"):
    print("Correct Password")
    flag = 1
else:
    print("Wrong Password")


if(flag  ==  1):
    print("Access Allowed")
```

OUTPUT: