# ASSIGNMENT - 7

## // C_DES.py

```
import socket

# Create a socket object
s = socket.socket()

# Define the port on which you want to connect
port = 5001

print("Client program running...\n")
# connect to the server on local computer
s.connect(('127.0.0.1', port))
name = input(str("\nEnter your name : "))
print("Connected...to Server...\n")

s.send(name.encode())
s_name = s.recv(1024)
s_name = s_name.decode()
print(s_name, "has joined the Client\n")


flag = 0
while True:
    if flag == 1:
        break
#-------- Sending P  Client to Server -------------------
    P = input("Enter the Value of Prime Number P : ")
    Pstr =str(P)
    print ('Client sending P to Server : ',Pstr)
    s.send(Pstr.encode())
#------------------------------------------------------------------

#-------- Receiving G from Server to Client ----------------
    Gstr = s.recv(1024)
    Gstr = Gstr.decode()
    print ('Prime no G from Client',Gstr)
    G = int(Gstr)


#------------------------------------------------------------------
    flag = flag + 1

# Client will choose the private key b

    b = input("\nEnter Client Private Key : ")
    b = int(b)

    print('\nThe Private Key B for Client is : ', b)

            # gets the generated key
    y = int(pow(int(G),b,int(P)))

flag = 0
while True:
    if flag == 1:
        break
#-------- Sending Y  Client to Server -------------------
    Ystr =str(y)
    print ('\nClient sending Y to Server : ',Ystr)
    s.send(Ystr.encode())
#-----------------------------------------------------------------

#-------- Receiving X from Server to Client ----------------
    Xstr = s.recv(1024)
    Xstr = Xstr.decode()
    print ('\nX recevied from Client : ',Xstr)
    X = int(Xstr)
#-----------------------------------------------------------------
    flag = flag + 1


            # Secret key for Client
    kb = int(pow(X,b,int(P)))

    print('Shared Secret Key for the Client is : ', kb)
    print("\n----------------------------------------------------\n")

#------------End of Diffie - Hellman -----------------------------------

print ("\nModifying the key to make it 64 bits")
key = str(kb)
for i in range(15):
    key = key + str(kb)
    i = i + 1

print ("The modified key is : ", key)

#------------Start of DES -----------------------------------

# 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))

    # 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])

    # Combination
    combine = left + right

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

# 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))

#-------------------End of DES ------------------------------------------


flag = 0
while True:
   if flag == 1:
      break

#-------- Encrypting Plain Text and Sending Cipher Text form Client to Server -------------------

   pt = input("\nEnter the Plain Text : ")
   key = str(kb)
   print("\nPerforming Encryption")
   cipher_text = bin2hex(encrypt(pt, rkb, rk))
   print("Cipher Text : ",cipher_text)
   s.send(cipher_text.encode())
   print("\nSending Cipher text to Server\n")

#-------------------------------------------------------------------
```

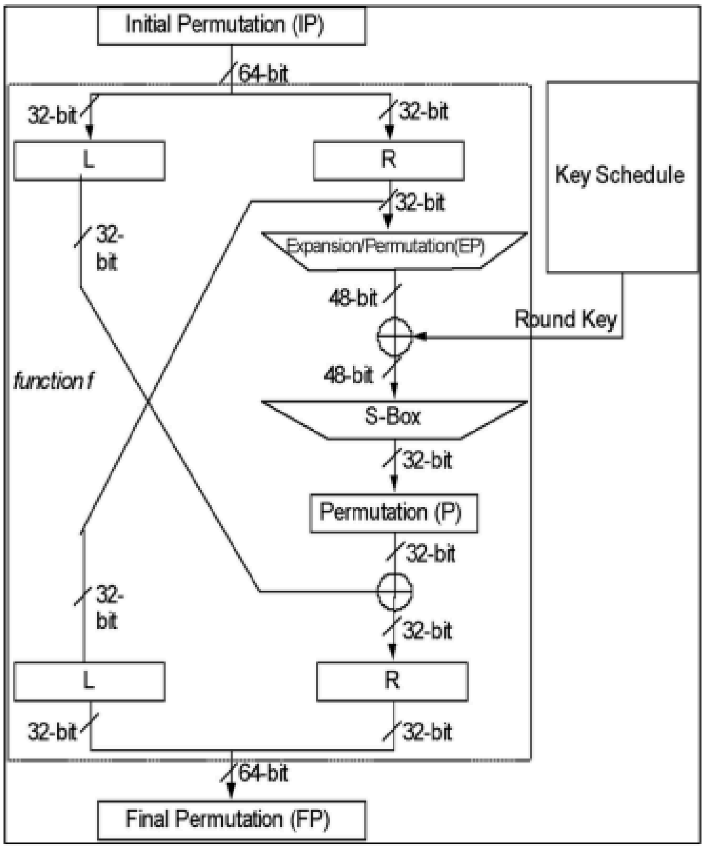
```
#-------- Receiving Cipher Text from Server and Decrypting it to Plain Text ----------------
    cipher_text = s.recv(1024)
    cipher_text = cipher_text.decode()
    print ('\nCipher Text Received From Server : ',cipher_text)
    print("\nPerforming Decryption")
    rkb_rev = rkb[::-1]
    rk_rev = rk[::-1]
    text = bin2hex(encrypt(cipher_text, rkb_rev, rk_rev))
    print("Plain Text : ",text)
#-----------------------------------------------------------------
    flag = flag + 1
```

## //S_DES.py

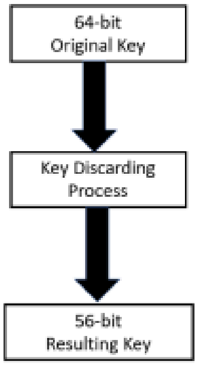
```
import socket

# next create a socket object
s = socket.socket()
print ("Socket successfully created")

# reserve a port on your computer in our
# case it is 12345 but it can be anything
host = '127.0.0.1'
port = 5001

# Next bind to the port
# we have not typed any ip in the ip field
# instead we have inputted an empty string
# this makes the server listen to requests
# coming from other computers on the network

s.bind((host, port))
print ("socket binded to %s" %(port))

# put the socket into listening mode
s.listen(5)
print ("\n\n Server Socket is listening.... Waiting for Client to Get Connected\n")

# a forever loop until we interrupt it or
# an error occurs
# Establish connection with client.
conn, addr = s.accept()
print ('Got connection from', addr )

name = input(str("\nEnter your name : "))

s_name = conn.recv(1024)
s_name = s_name.decode()
print(s_name, "has connected to the Server")
conn.send(name.encode())


flag = 0
while True:
    if flag == 1:
        break
#-------- Receiving P from Client to Server ----------------
    Pstr = conn.recv(1024)
    Pstr = Pstr.decode()
    print ('Prime no P from Client : ',Pstr)
    P = int(Pstr)
#--------------------------------------------------------------

#-------- Sending G from Server to Client ------------------
    G = input("Enter the Value of Prime Number G : ")
    Gstr =str(G)
    print ('Client sending P to Server : ',Gstr)
    conn.send(Gstr.encode())

#----------------------------------------------------------------
    flag = flag + 1

# Server will choose the private key a
    a = input("Enter Server Private Key : ")
    a = int(a)

    print('\nThe Private Key A for Server is : ', a)

# gets the generated key
    x = int(pow(int(G),a,int(P)))

flag = 0
while True:
    if flag == 1:
        break

#-------- Receiving Y from Client to Server ----------------
    Ystr = conn.recv(1024)
    Ystr = Ystr.decode()
    print ('\nY received from Server : ',Ystr)
    Y = int(Ystr)
#-------------------------------------------------------------

#-------- Sending X from Server to Client -----------------
    Xstr =str(x)
    print ('\nClient sending P to Server : ',Xstr)
    conn.send(Xstr.encode())
#---------------------------------------------------------------
    flag = flag + 1


# Secret key for Server
    ka = int(pow(Y,a,P))

    print ('\nShared Secret key for the Server is : ', ka)
    print("\n----------------------------------------------------\n")
#-------------End of Diffie - Hellman ----------------------------------

print ("\nModifying the key to make it 64 bits")
key = str(ka)
for i in range(15):
    key = key + str(ka)
    i = i + 1

print ("The modified key is : ", key)

#-------------Start of DES ----------------------------------
# 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))

    # 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])

    # Combination
    combine = left + right

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

# 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 = []
```

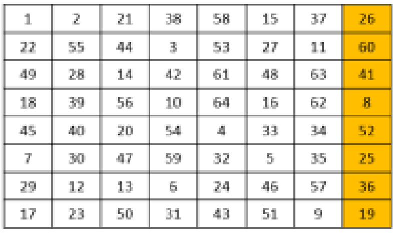
```
rk = []
for i in range(0, 16):
            # Shifting the bits by nth shifts by checking from shift table
            left = shift_left(left, shift_table[i])
            right = shift_left(right, shift_table[i])

            # 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))

#-------------------End of DES ------------------------------------------


flag = 0
while True:
    if flag == 1:
        break


#-------- Receiving Cipher Text from  and Decrypting it to Plain Text ----------------
    cipher_text = conn.recv(1024)
    cipher_text = cipher_text.decode()
    print ('\nCipher Text Received From Server : ',cipher_text)
    print("\nPerforming Decryption")
    rkb_rev = rkb[::-1]
    rk_rev = rk[::-1]
    text = bin2hex(encrypt(cipher_text, rkb_rev, rk_rev))
    print("\nPlain Text : ",text)
#-------------------------------------------------------------------
#-------- Encrypting Plain Text and Sending Cipher Text form Server to Client ------------------

    pt = input("\nEnter the Plain Text : ")
    key = str(ka)
    print("\nPerforming Encryption")
    cipher_text = bin2hex(encrypt(pt, rkb, rk))
    print("\nCipher Text : ",cipher_text)
    conn.send(cipher_text.encode())
    print("\nSending Cipher text to Server\n")

#-------------------------------------------------------------------

    flag = flag + 1
```



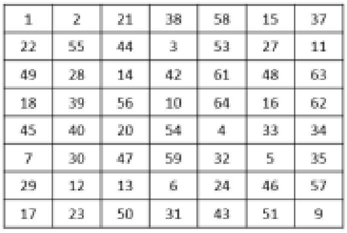
**Block Diagram of DES Algorithm**



Every 8th Bit of Original Key is Discarded



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



| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 21 | 38 | 58 | 15 | 37 |
| 22 | 55 | 44 | 3 | 53 | 27 | 11 |
| 49 | 28 | 14 | 42 | 61 | 48 | 63 |
| 18 | 39 | 56 | 10 | 64 | 16 | 62 |
| 45 | 40 | 20 | 54 | 4 | 33 | 34 |
| 7 | 30 | 47 | 59 | 32 | 5 | 35 |
| 29 | 12 | 13 | 6 | 24 | 46 | 57 |
| 17 | 23 | 50 | 31 | 43 | 51 | 9 |

**Key Transformation from 64 to 56 Bit Key**