Cryptography and Network Security Lab

Assignment 6
Implementation and Understanding of Data Encryption Standard (DES) Cipher

2019BTECS00058
Devang K

Batch: B2

Title: Implementation and Understanding of Data Encryption Standard (DES)

Aim: To Study, Implement and Demonstrate the Data Encryption Standard (DES)

- Part A- Implementation of DES using Virtual Lab
- Part B- Implementation of DES using C/C++/Java/Python or any other programming language

Theory:

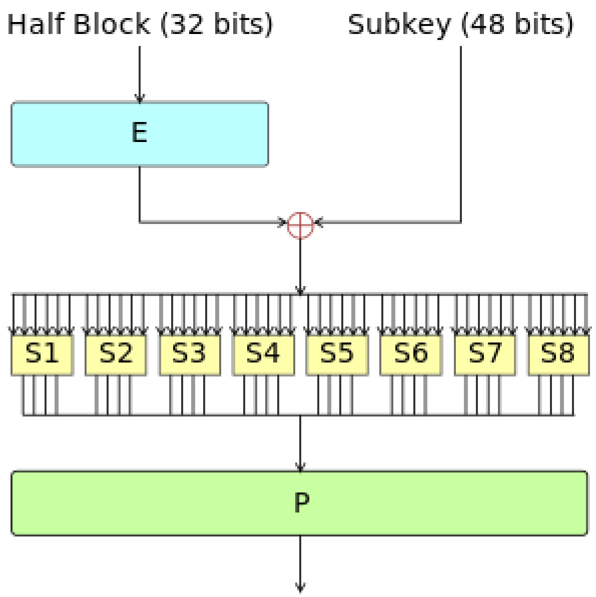
The Data Encryption Standard is a symmetric-key algorithm for the encryption of digital data. Although its short key length of 56 bits makes it too insecure for modern applications, it has been highly influential in the advancement of cryptography.

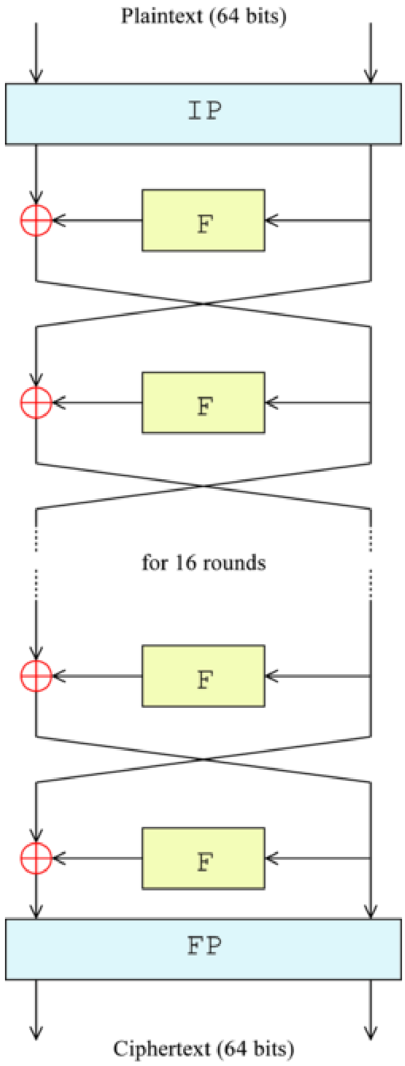
The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST). DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56bits, since 8 of the 64 bits of the key are not used by the encryption algorithm.

DES is the archetypal block cipher—an algorithm that takes a fixed-length string of plaintext bits and transforms it through a series of complicated operations into another ciphertext bitstring of the same length. In the case of DES, the block size is 64 bits. DES also uses a key to customize the transformation, so that decryption can supposedly only be performed by those who know the particular key used to encrypt. The key ostensibly consists of 64 bits; however, only 56 of these are actually used by the algorithm. Eight bits are used solely for checking parity, and are thereafter discarded. Hence the effective key length is 56 bits. The key is nominally stored or transmitted as 8 bytes, each with odd parity.

One bit in each 8-bit byte of the KEY may be utilized for error detection in key generation, distribution, and storage. Bits 8, 16, ... , 64 are for use in ensuring that each byte is of odd parity. Like other block ciphers, DES by itself is not a secure means of encryption, but must instead be used in a mode of operation. FIPS-81 specifies several modes for use with DES. Further comments on the usage of DES are contained in FIPS-74.

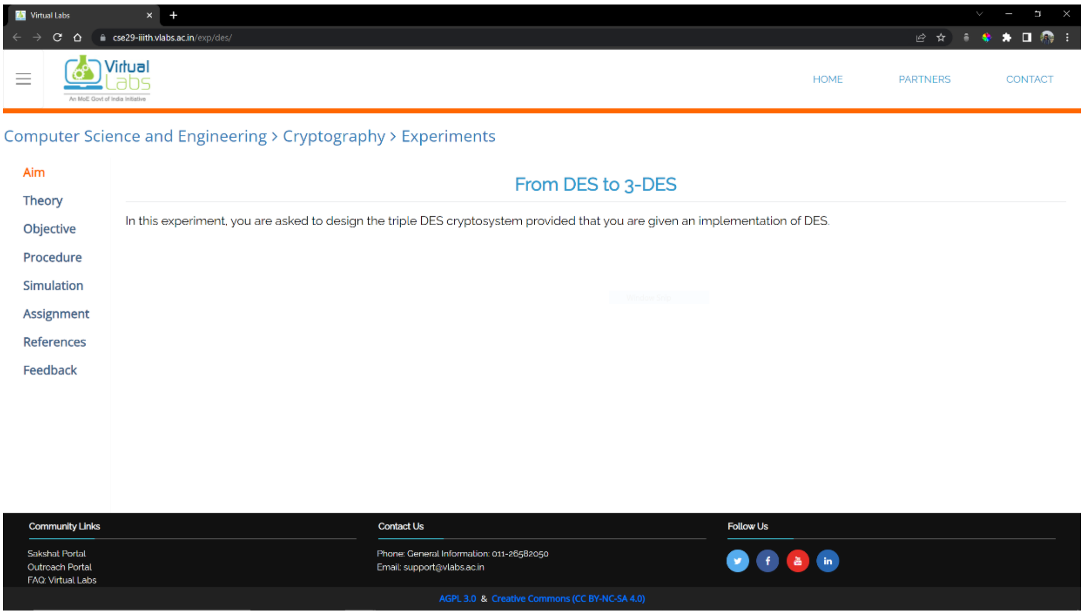
Decryption uses the same structure as encryption, but with the keys used in reverse order. (This has the advantage that the same hardware or software can be used in both directions.)



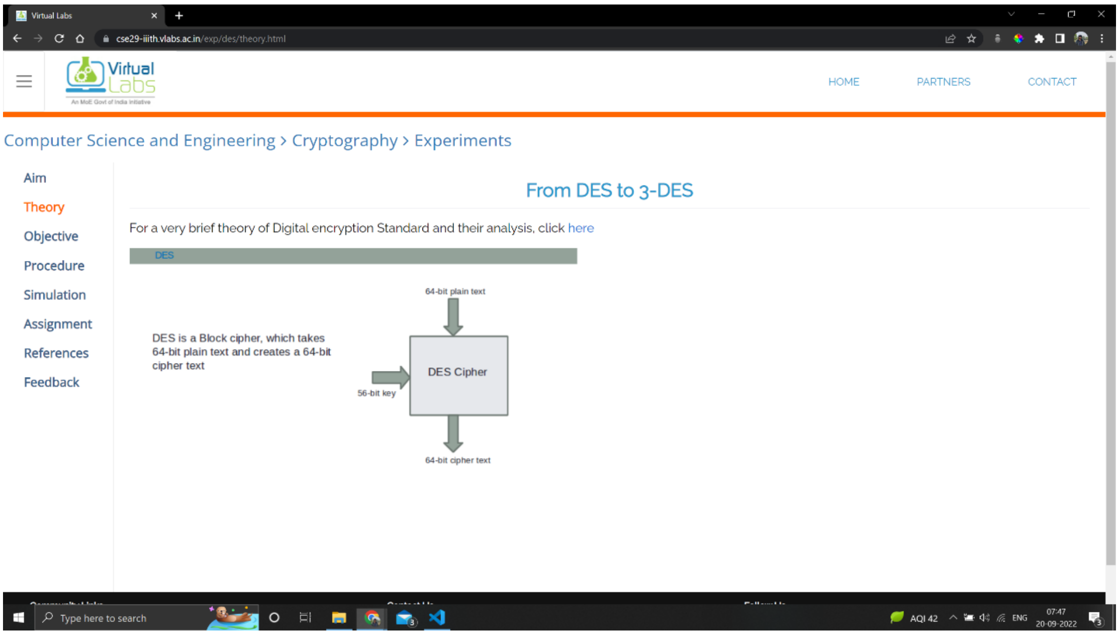



## V-Lab Implementation:

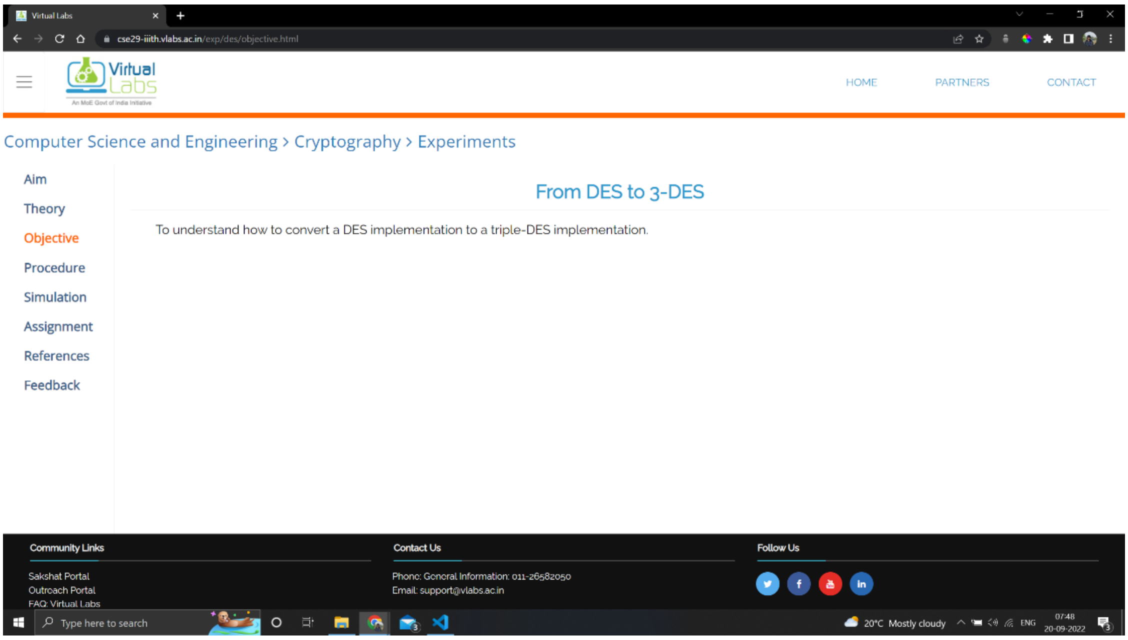
Let us work on the simulator. This simulator is performing the 3DES algorithm.



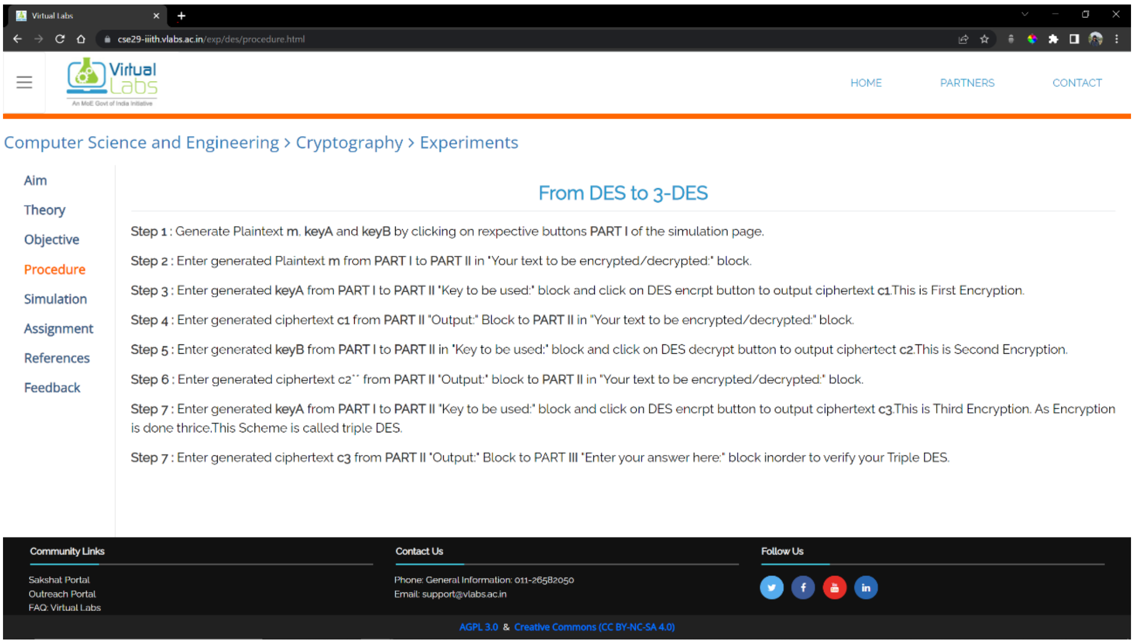

Looking at the theory:



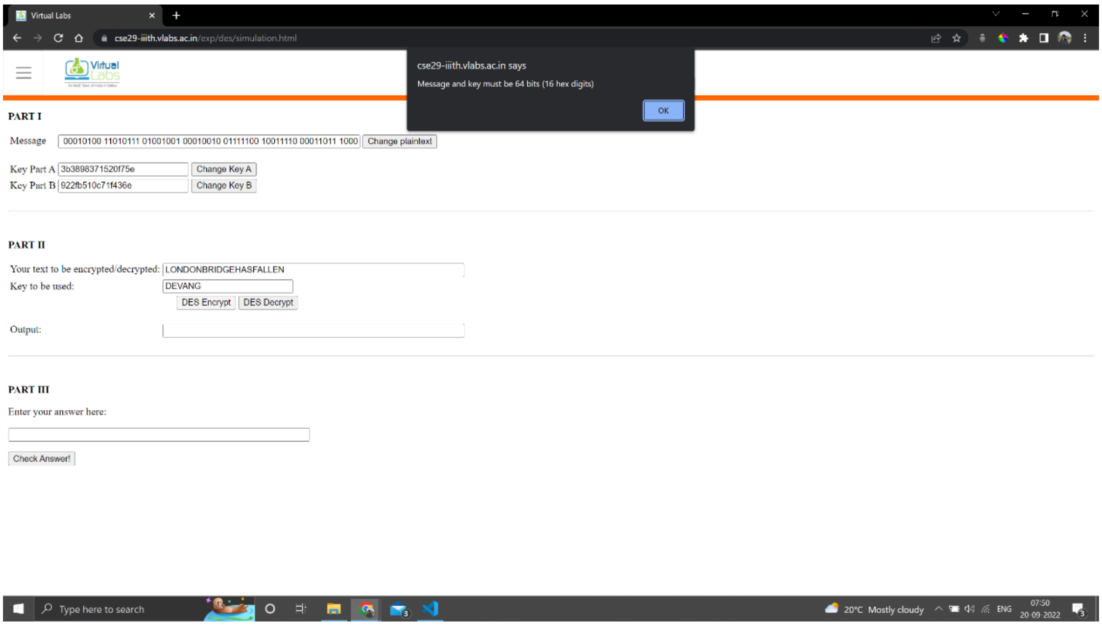

Objective:




# Procedure



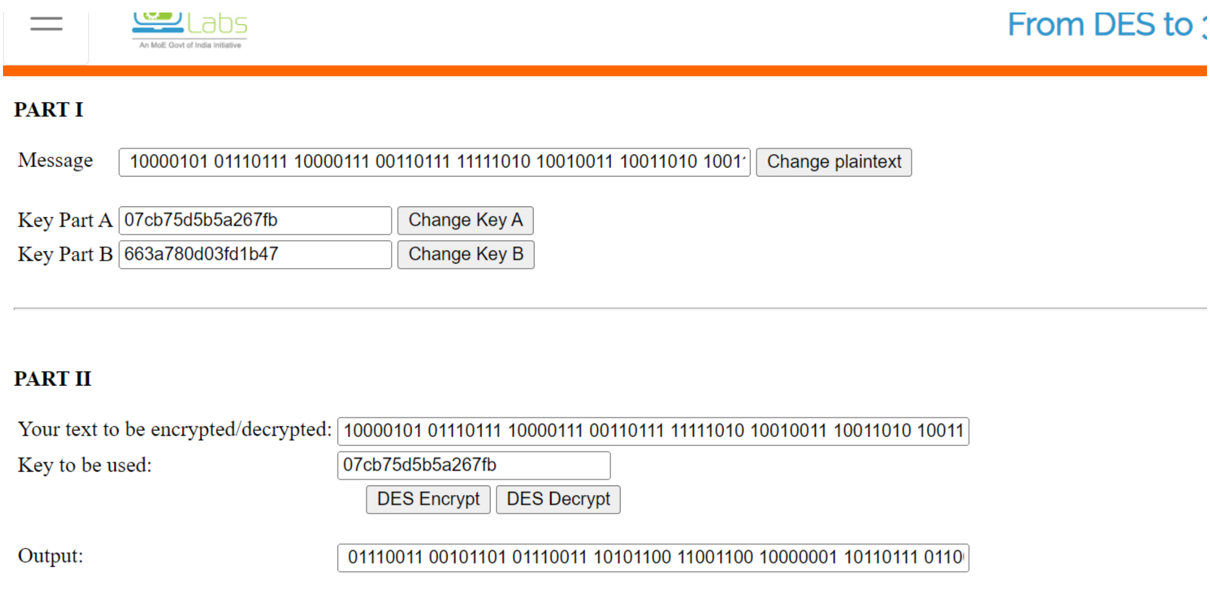

We try entering a value




Simple DES

Plaintext: 10000101 01110111 10000111 00110111 11111010 10010011 10011010 1001111

Key: 07cb75d5b5a267fb




EncText: 01110011 00101101 01110011 10101100 11001100 10000001 10110111 01100010

Decryption

Plaintext: 10000101 01110111 10000111 00110111 11111010 10010011 10011010 10011110

**PART I**

Message 10000101 01110111 10000111 00110111 11111010 10010011 10011010 1001 Change plaintext

Key Part A 07cb75d5b5a267fb Change Key A

Key Part B 663a780d03fd1b47 Change Key B

---

**PART II**

Your text to be encrypted/decrypted: 01110011 00101101 01110011 10101100 11001100 10000001 10110111 0110

Key to be used: 07cb75d5b5a267fb

DES Encrypt DES Decrypt

Output: 10000101 01110111 10000111 00110111 11111010 10010011 10011010 1001

---

Let's do once from Part 1 values

DES -> 3DES

Plaintext -> 00010100 11010111 01001001 00010010 01111100 10011110 00011011 1000001

Key A -> 3b3898371520f75e

Key B -> 922fb510c71f436e

---

**PART II**

Your text to be encrypted/decrypted: 593428AE137D8346

Key to be used: 975321BA72BA9361

DES Encrypt DES Decrypt

Output: 95438200 31173864 00000000 10010011 10000010 00001101 00100011 011

---

Text - 593428AE137D8346
Key - 975321BA72BA9361

Encryption: 95438200 31173864 00000000 10010011 10000010 00001101 00100011 01100100

Then we take this encrypted text and encrypt with another key.

**PART II**

Your text to be encrypted/decrypted: 95438200 31173864 00000000 10010011 10000010 00001101 00100011 011

Key to be used: 975321BA72BA9361

DES Encrypt DES Decrypt

Output: 59342800 13708346 00000000 00000000 00000000 00000000 00000000 00(

Plaintext -> 00010100 11010111 01001001 00010010 01111100 10011110 00011011 1000001

Key A -> 3b3898371520f75e

Key B -> 922fb510c71f436e

An MoE Govt of India Initiative

**PART I**

Message 00010100 11010111 01001001 00010010 01111100 10011110 00011011 1000 Change plaintext

Key Part A 3b3898371520f75e Change Key A

Key Part B 922fb510c71f436e Change Key B

**PART II**

Your text to be encrypted/decrypted: 00010100 11010111 01001001 00010010 01111100 10011110 00011011 1000(

Key to be used: 3b3898371520f75e

DES Encrypt DES Decrypt

Output: 00111110 11010100 11010111 01101101 10000110 11100111 00010001 01111

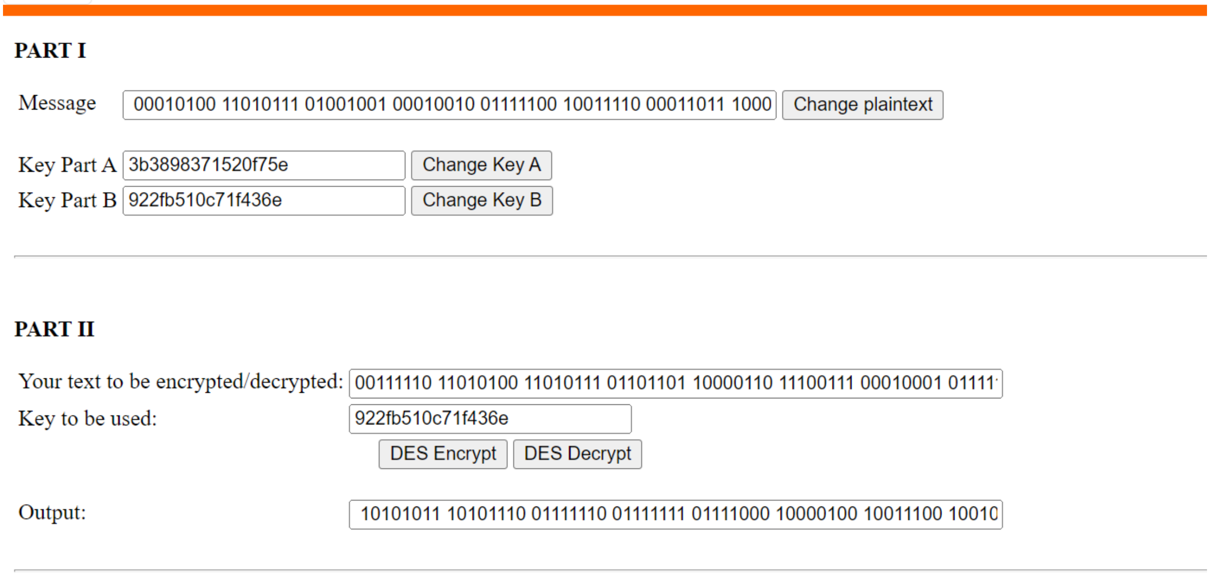
Basically, we need to do this:

Plaintext + KeyA -> C1 Enc

C1 + KeyB -> C2 Dec

C2 + KeyA -> C3 Enc

C3 is the 3 DES Cipher




PT + KA -> C1 Enc

00010100 11010111 01001001 00010010 01111100 10011110 00011011 1000001 + 3b3898371520f75e => 00111110 11010100 11010111 01101101 10000110 11100111 00010001 01111101

**PART I**

Message 00010100 11010111 01001001 00010010 01111100 10011110 00011011 1000 Change plaintext

Key Part A 3b3898371520f75e Change Key A
Key Part B 922fb510c71f436e Change Key B

---

**PART II**

Your text to be encrypted/decrypted: 10101011 10101110 01111110 01111111 01111000 10000100 10011100 10010
Key to be used: 3b3898371520f75e
DES Encrypt DES Decrypt

Output: 00011101 11100100 10001000 01101111 11010001 00011011 00110000 1100

---

C1 + KB -> C2 Dec

00111110 11010100 11010111 01101101 10000110 11100111 00010001 01111101 + 922fb510c71f436e => 10101011 10101110 01111110 01111111 01111000 10000100 10011100 10010110

**PART III**

Enter your answer here:

00011101 11100100 10001000 01101111 11010001 00011011 00110000 1100

Check Answer!

CORRECT!

C2 + KA -> C3 Enc

10101011 10101110 01111110 01111111 01111000 10000100 10011100 10010110 + 3b3898371520f75e => 00011101 11100100 10001000 01101111 11010001 00011011 00110000 11000000

Code:

```
# The plaintext and ciphertext would be hexadecimal
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

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]
```

```python
        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):
    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
```

```python
# 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]

# --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]

# 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]

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
```

```python
        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

# 64bit PT and 64bit Key
print("DES Algorithm")

print("What do you wish to do?")
print("1. Encrypt")
print("2. Decrypt\n")

n = int(input())
if n == 1:
    print("Enter Plaintext: ", end='')
    plaintext = input()
    print("Enter Key: ", end='')
    key = input()
    key = hex2bin(key)
    # 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))
    cipher_text = bin2hex(encrypt(plaintext, rkb, rk))
    print("Cipher Text : ", cipher_text)
else:
    print("Enter Ciphertext: ", end='')
    ciphertext = input()
    print("Enter Key: ", end='')
```

```python
    key = input()
    key = hex2bin(key)
    # 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))
    rkb_rev = rkb[::-1]
    rk_rev = rk[::-1]
    text = bin2hex(encrypt(ciphertext, rkb_rev, rk_rev))
    print("Plaintext: ", text)
```

We now solve some examples with the code.

Say we wish to encrypt: ‘123456ABCD132536’
and we take our key as ‘AABB09182736CCDD’

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Temp\DES> py .\script.py

DES Algorithm
What do you wish to do?
1. Encrypt
2. Decrypt

1
Enter Plaintext: 123456ABCD132536
Enter Key: AABB09182736CCDD
```

```
After initial permutation 14A7D67818CA18AD
Round  1   18CA18AD   9DAF94C4   A1DB4D5057F0
Round  2   9DAF94C4   908A3267   AE149ADCF814
Round  3   908A3267   D19BF56B   7E025C2146FC
Round  4   D19BF56B   A12C1D36   0ED81899B883
Round  5   A12C1D36   E03FDA4D   0E297EA64635
Round  6   E03FDA4D   DFD06779   AE6C091B2BC6
Round  7   DFD06779   CB3473C9   4B2F28B4C191
Round  8   CB3473C9   2CDB0C31   C8BC99432647
Round  9   2CDB0C31   428FF863   9940A66093D3
Round  10   428FF863   8C2A99C3   B00BBC17A42F
Round  11   8C2A99C3   FAC20EAE   9432256E1DC0
Round  12   FAC20EAE   23E02501   831E7408E17F
Round  13   23E02501   3F786CF1   CC72E467DC80
Round  14   3F786CF1   8520A7C2   92D768C8057B
Round  15   8520A7C2   543378E7   C853638FDA0C
Round  16   C3B1E73B   543378E7   3FA232090E6A
Cipher Text :  77678609B93FCE56
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Temp\DES>
```

We now decipher:

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Temp\DES> py .\script.py

DES Algorithm
What do you wish to do?
1. Encrypt
2. Decrypt

2
Enter Ciphertext: 77678609B93FCE56
Enter Key: AABB09182736CCDD
```

```
After initial permutation C3B1E73B543378E7
Round  1   543378E7   8520A7C2   3FA232090E6A
Round  2   8520A7C2   3F786CF1   C853638FDA0C
Round  3   3F786CF1   23E02501   92D768C8057B
Round  4   23E02501   FAC20EAE   CC72E467DC80
Round  5   FAC20EAE   8C2A99C3   831E7408E17F
Round  6   8C2A99C3   428FF863   9432256E1DC0
Round  7   428FF863   2CDB0C31   B00BBC17A42F
Round  8   2CDB0C31   CB3473C9   9940A66093D3
Round  9   CB3473C9   DFD06779   C8BC99432647
Round  10   DFD06779   E03FDA4D   4B2F28B4C191
Round  11   E03FDA4D   A12C1D36   AE6C091B2BC6
Round  12   A12C1D36   D19BF56B   0E297EA64635
Round  13   D19BF56B   908A3267   0ED81899B883
Round  14   908A3267   9DAF94C4   7E025C2146FC
Round  15   9DAF94C4   18CA18AD   AE149ADCF814
Round  16   14A7D678   18CA18AD   A1DB4D5057F0
Plaintext:  123456ABCD132536
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Temp\DES>
```

Therefore, we get our plaintext back.

Let's take another example:

Say we wish to encrypt: '9307805348ABCDEF'
and we take our key as '1234567890ABCDEF'

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Temp\DES> py .\script.py

DES Algorithm
What do you wish to do?
1. Encrypt
2. Decrypt

1
Enter Plaintext: 9307805348ABCDEF
Enter Key: 1234567890ABCDEF
```

```
After initial permutation D809C2EBE5A0F0AB
Round  1   E5A0F0AB   7FE539B3   62748A4D1D71
Round  2   7FE539B3   A6857342   1432A5ECD2A0
Round  3   A6857342   2B17BFB8   931C70D04E6B
Round  4   2B17BFB8   2FDCAD0C   CC62E49E9A18
Round  5   2FDCAD0C   B19B053B   92D70C917770
Round  6   B19B053B   62172319   48136339AA20
Round  7   62172319   C21F8045   A1D86DF06C16
Round  8   C21F8045   3450C626   8163C22D229E
Round  9   3450C626   867834B2   76025E8227B1
Round  10   867834B2   832F3E50   2ED8007B2B05
Round  11   832F3E50   D69C7FCF   0A297E72419A
Round  12   D69C7FCF   7CE4784C   AC641945310F
Round  13   7CE4784C   B2D19C9D   470F28E630E8
Round  14   B2D19C9D   F943D4D6   CAB891609B6F
Round  15   F943D4D6   CACB1412   1DAE4A169CBA
Round  16   E5314441   CACB1412   A10B9C88754E
Cipher Text :  71A24CA01A50E5E0
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Temp\DES>
```

We now decipher:

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Temp\DES> py .\script.py

DES Algorithm
What do you wish to do?
1. Encrypt
2. Decrypt

2
Enter Ciphertext: 71A24CA01A50E5E0
Enter Key: 1234567890ABCDEF
```

```
After initial permutation E5314441CACB1412
Round  1   CACB1412   F943D4D6   A10B9C88754E
Round  2   F943D4D6   B2D19C9D   1DAE4A169CBA
Round  3   B2D19C9D   7CE4784C   CAB891609B6F
Round  4   7CE4784C   D69C7FCF   470F28E630E8
Round  5   D69C7FCF   832F3E50   AC641945310F
Round  6   832F3E50   867834B2   0A297E72419A
Round  7   867834B2   3450C626   2ED8007B2B05
Round  8   3450C626   C21F8045   76025E8227B1
Round  9   C21F8045   62172319   8163C22D229E
Round  10   62172319   B19B053B   A1D86DF06C16
Round  11   B19B053B   2FDCAD0C   48136339AA20
Round  12   2FDCAD0C   2B17BFB8   92D70C917770
Round  13   2B17BFB8   A6857342   CC62E49E9A18
Round  14   A6857342   7FE539B3   931C70D04E6B
Round  15   7FE539B3   E5A0F0AB   1432A5ECD2A0
Round  16   D809C2EB   E5A0F0AB   62748A4D1D71
Plaintext:  9307805348ABCDEF
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Temp\DES>
```

We get the plaintext back.

Thus, we demonstrated the working of the code with examples.

## Conclusion:

Thus, the Data Encryption Standard (DES) algorithm was studied and demonstrated with the code.