Roll. No. A016	Name: Varun Khadayate
Class B.Tech CsBs	Batch: 1
Date of Experiment: 27-08-2021	Subject: Cryptology

Aim

To Implement DES Encryption and Decryption standard

Theory

Until the introduction of the Advanced Encryption Standard (AES) in 2001, the Data Encryption Standard (DES) was the most widely used encryption scheme. DES was issued in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46). The algorithm itself is referred to as the Data Encryption Algorithm (DEA).[7] For DEA, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption.

Over the years, DES became the dominant symmetric encryption algorithm, especially in financial applications. In 1994, NIST reaffirmed DES for federal use for another five years; NIST recommended the use of DES for applications other than the protection of classified information. In 1999, NIST issued a new version of its standard (FIPS PUB 46-3) that indicated that DES should be used only for legacy systems and that triple DES (which in essence involves repeating the DES algorithm three times on the plaintext using two or three different keys to produce the ciphertext) be used. We study triple DES in Chapter 6. Because the underlying encryption and decryption algorithms are the same for DES and triple DES, itremains important to understand the DES cipher.

The overall scheme for DES encryption is illustrated in Figure 3.5. As with any encryption scheme, there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the plaintext must be 64 bits in length and thekey is 56 bits in length.[8]

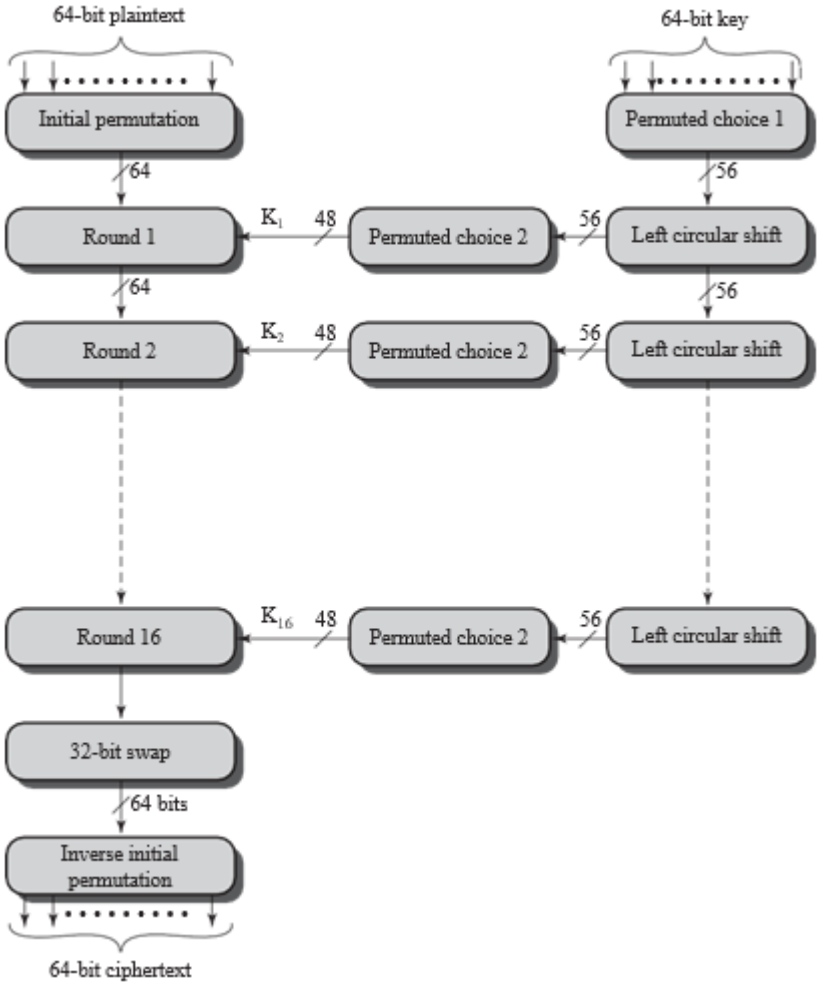
Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in three phases. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*. This is followed by a phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions. The output of the last (six- teenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the **preoutput**.

Finally, the preoutput is passed through a permutation [IP^{-1}] that is the inverse ofthe initial permutation function, to produce the 64-bit ciphertext. With the excep-
tion of the initial and final permutations, DES has the exact structure of a Feistel cipher, as shown in Figure 3.3.

The right-hand portion of Figure 3.5 shows the way in which the 56-bit key is used. Initially, the key is passed through a permutation function. Then, for each of the sixteen rounds, a *subkey* (K_i) is produced by the combination of a left circular shift and a permutation. The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

DES Decryption

As with any Feistel cipher, decryption uses the same algorithm as encryption, except that the application of the subkeys is reversed. Additionally, the initial and final permutations are reversed.


64-bit plaintext
64-bit key
Initial permutation
Permuted choice 1
64
56
Round 1
K1
48
Permuted choice 2
56
Left circular shift
64
56
Round 2
K2
48
Permuted choice 2
56
Left circular shift
Round 16
K16
48
Permuted choice 2
56
Left circular shift
32-bit swap
64 bits
Inverse initial permutation
64-bit ciphertext

Code

```
def main():

    while True:
        ch = int(input('Welcome to DES Encryption and Decryption Program Made
by Varun Khadayate..\n [*] Press 1 for Encryption \n [*] Press 2 for
Decryption \n [*] Press 0 to exit..\n \nYour Choice:: '))

        if ch == 1:
            print("================================================")
            print("             !!!!DES Encryption!!!!             ")
            print()
            plaintext = input("Enter the message to be encrypted : ")
            key = input("Enter a key of 8 length (64-bits) (characters or
numbers only) : ")
            print()
            if len(key) != 8:
                print("Invalid Key. Key should be of 8 length (8 bytes).")
                return

            isPaddingRequired = (len(plaintext) % 8 != 0)
            ciphertext = DESEncryption(key, plaintext, isPaddingRequired)
            print()
            print("Encrypted Ciphertext is : %r " % ciphertext)
            print()
            print("================================================")

        elif ch == 2:
            print("\n================================================")
            print("               !!!DES Decryption!!!               ")
            plaintext = DESDecryption(key, ciphertext, isPaddingRequired)
            print()
            print("Enter the message to be decrypted : %r" % ciphertext)
            print("Enter the Key : ",key)
            print("Decrypted plaintext is  : ", plaintext)
            print()
            print("================================================")

        elif ch == 0:
            print("\n============================================")
            print("     Thank You for using the Software ;)      ")
            print("                 Exiting Now.                 ")
            print("============================================")
            exit()

eachRoundPermutationMatrix = [
    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
]

finalPermutationMatrix = [
    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 DESEncryption(key, text, padding):
    if padding == True:
        text = addPadding(text)
    ciphertext = DES(text, key, padding, True)
    return ciphertext

def DESDecryption(key, text, padding):
    plaintext = DES(text, key, padding, False)
    if padding == True:
        return removePadding(plaintext)
    return plaintext

initialPermutationMatrix = [
    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
]

expandMatrix = [
    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
]
```

```
def DES(text, key, padding, isEncrypt):
    isDecrypt = not isEncrypt
    keys = generateKeys(key)
    plaintext8byteBlocks = nSplit(text, 8)
    result = []
    for block in plaintext8byteBlocks:
        block = stringToBitArray(block)
        block = permutation(block, initialPermutationMatrix)
        leftBlock, rightBlock = nSplit(block, 32)
        temp = None
        for i in range(16):
            expandedRightBlock = expand(rightBlock, expandMatrix)
            if isEncrypt == True:
                temp = xor(keys[i], expandedRightBlock)
            elif isDecrypt == True:
                temp = xor(keys[15 - i], expandedRightBlock)
            temp = SboxSubstitution(temp)
            temp = permutation(temp, eachRoundPermutationMatrix)
            temp = xor(leftBlock, temp)
            leftBlock = rightBlock
            rightBlock = temp
        result += permutation(rightBlock + leftBlock, finalPermutationMatrix)
    finalResult = bitArrayToString(result)
    return finalResult

SHIFT = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

keyPermutationMatrix1 = [
    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
]

keyPermutationMatrix2 = [
    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 generateKeys(key):
    keys = []
    key = stringToBitArray(key)

    key = permutation(key, keyPermutationMatrix1)

    leftBlock, rightBlock = nSplit(key, 28)

    for i in range(16):
        leftBlock, rightBlock = leftShift(leftBlock, rightBlock, SHIFT[i])
        temp = leftBlock + rightBlock
        keys.append(permutation(temp, keyPermutationMatrix2))
    return keys

SboxesArray = [
    [
        [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],
    ]
]

def SboxSubstitution(bitArray):
    blocks = nSplit(bitArray, 6)
    result = []
    for i in range(len(blocks)):
        block = blocks[i]
        row = int( str(block[0]) + str(block[5]), 2 )
        column = int(''.join([str(x) for x in block[1:-1]]), 2)
        sboxValue = SboxesArray[i][row][column]
        binVal = binValue(sboxValue, 4)
        result += [int(bit) for bit in binVal]
    return result


def addPadding(text):
    paddingLength = 8 - (len(text) % 8)
    text += chr(paddingLength) * paddingLength
    return text

def removePadding(data):
    paddingLength = ord(data[-1])
    return data[ : -paddingLength]

def expand(array, table):
    return [array[element - 1] for element in table]

def permutation(array, table):
    return [array[element - 1] for element in table]

def leftShift(list1, list2, n):
    return list1[n:] + list1[:n], list2[n:] + list2[:n]

def nSplit(list, n):
```

```
    return [ list[i : i + n] for i in range(0, len(list), n)]

def xor(list1, list2):
    return [element1 ^ element2 for element1, element2 in zip(list1,list2)]

def binValue(val, bitSize):
    binVal = bin(val)[2:] if isinstance(val, int) else bin(ord(val))[2:]
    while len(binVal) < bitSize:
        binVal = "0" + binVal
    return binVal

def stringToBitArray(text):
    bitArray = []
    for letter in text:
        binVal = binValue(letter, 8)
        binValArr = [int(x) for x in list(binVal)]
        bitArray += binValArr
    return bitArray

def bitArrayToString(array):
    byteChunks = nSplit(array, 8)
    stringBytesList = []
    stringResult = ''
    for byte in byteChunks:
        bitsList = []
        for bit in byte:
            bitsList += str(bit)
        stringBytesList.append(''.join(bitsList))

    result = ''.join([chr(int(stringByte, 2)) for stringByte in
stringBytesList])
    return result

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

Output

```
Welcome to DES Encryption and Decryption Program Made by Varun Khadayate..
 [*] Press 1 for Encryption
 [*] Press 2 for Decryption
 [*] Press 0 to exit..

Your Choice:: 1
==============================================
             !!!!DES Encryption!!!!

Enter the message to be encrypted : varun mahendra khadayate
Enter a key of 8 length (64-bits) (characters or numbers only) : monarchy


Encrypted Ciphertext is : '\t6Ô[Â\x18\x87\x1c´\x05MGC\\ãWc\x85fÀSì\x15#'

==============================================
Welcome to DES Encryption and Decryption Program Made by Varun Khadayate..
 [*] Press 1 for Encryption
 [*] Press 2 for Decryption
 [*] Press 0 to exit..

Your Choice:: 2

==============================================
               !!!DES Decryption!!!

Enter the message to be decrypted : '\t6Ô[Â\x18\x87\x1c´\x05MGC\\ãWc\x85fÀSì\x15#'
Enter the Key :  monarchy
Decrypted plaintext is  :  varun mahendra khadayate

==============================================
Welcome to DES Encryption and Decryption Program Made by Varun Khadayate..
 [*] Press 1 for Encryption
 [*] Press 2 for Decryption
 [*] Press 0 to exit..

 Your Choice:: 0

 ==========================================
      Thank You for using the Software ;)
                 Exiting Now.
 ==========================================
```

Conclusion

Hence we were able to perform DES encryption and Decryption.