#!/usr/bin/env python

# coding: utf-8

# In[2]:

# Line 2-28 = Fungsi Utama

def main():

print()

# Teks input untuk diisi user

plaintext = input("Enkripsi Teks : ")

key = input("Masukkan Kunci : ")

print()

# Mengecek apa kuncinya sudah benar atau belum

if len(key) != 8:

print("Kunci Invalid. Panjang kunci harus 8 bit!")

return

# Menentukan apakah padding diperlukan

isPaddingRequired = (len(plaintext) % 8 != 0)

# Enkripsi

ciphertext = DESEncryption(key, plaintext, isPaddingRequired)

# Dekripsi

plaintext = DESDecryption(key, ciphertext, isPaddingRequired)

# Menampilkan Hasil

print()

print("Hasil Enkripsi : %r " % ciphertext)

print("Hasil Dekripsi : ", plaintext)

print()

# Permutasi Matriks digunakan setelah setiap substitusi SBox untuk setiap putaran

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

]

# Permutasi Matriks akhir untuk data setelah 16

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

]

# Line 51-61 = Fungsi DES Enkripsi

def DESEncryption(key, text, padding):

# Menambahkan bantalan jika diperlukan

if padding == True:

text = addPadding(text)

# Enkripsi

ciphertext = DES(text, key, padding, True)

# Mengembalikan ciphertext

return ciphertext

# Line 64-75 = Fungsi DES Dekripsi

def DESDecryption(key, text, padding):

# Deskripsi

plaintext = DES(text, key, padding, False)

# Menghapus padding jika diperlukan

if padding == True:

# Menghapus padding dan mengembalikan plaintext

return removePadding(plaintext)

# Mengembalikan plaintext

return plaintext

# Matriks Permutasi Awal untuk data

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

]

# Perluas matriks untuk mendapatkan matriks data 48 bit untuk menerapkan xor dengan Ki

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):

"""Function to implement DES Algorithm."""

# Inisialisasi variabel diperlukan

isDecrypt = not isEncrypt

# Menghasilkan kunci

keys = generateKeys(key)

# Memisahkan teks menjadi blok 8 byte

plaintext8byteBlocks = nSplit(text, 8)

result = []

# Untuk semua blok teks 8-byte

for block in plaintext8byteBlocks:

# Ubah blok menjadi bit array

block = stringToBitArray(block)

# Melakukan permutasi awal

block = permutation(block, initialPermutationMatrix)

# Memisahkan blok menjadi dua blok berukuran 4 byte (32 bit)

leftBlock, rightBlock = nSplit(block, 32)

temp = None

# Menjalankan 16 Putaran DES identik untuk setiap blok teks

for i in range(16):

# Perluas rightBlock untuk mencocokkan ukuran kunci bulat (48-bit)

expandedRightBlock = expand(rightBlock, expandMatrix)

# Xor blok kanan dengan kunci yang sesuai

if isEncrypt == True:

# Untuk enkripsi, mulai dari kunci pertama dalam urutan normal

temp = xor(keys[i], expandedRightBlock)

elif isDecrypt == True:

# Untuk dekripsi, mulai dari kunci terakhir dalam urutan terbalik

temp = xor(keys[15 - i], expandedRightBlock)

# Langkah Substitusi Sbox

temp = SboxSubstitution(temp)

# Langkah Permutasi

temp = permutation(temp, eachRoundPermutationMatrix)

# XOR Langkah dengan LeftBlock

temp = xor(leftBlock, temp)

# Pertukaran blok

leftBlock = rightBlock

rightBlock = temp

result += permutation(rightBlock + leftBlock, finalPermutationMatrix)

# Mengubah array bit menjadi string

finalResult = bitArrayToString(result)

return finalResult

# Matriks yang digunakan untuk menggeser setelah setiap putaran tombol

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

# Matriks permutasi untuk kunci

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

]

# Matriks permutasi untuk kunci yang digeser untuk mendapatkan kunci berikutnya

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):

"""Function to generate keys for different rounds of DES."""

# Inisialisasi variabel diperlukan

keys = []

key = stringToBitArray(key)

# Initial permutation on key

key = permutation(key, keyPermutationMatrix1)

# Pisahkan kunci ke (leftBlock->LEFT), (rightBlock->RIGHT)

leftBlock, rightBlock = nSplit(key, 28)

# 16 putaran kunci

for i in range(16):

# Lakukan shift kiri (berbeda untuk putaran yang berbeda)

leftBlock, rightBlock = leftShift(leftBlock, rightBlock, SHIFT[i])

# Gabungkan mereka

temp = leftBlock + rightBlock

# Permutasi pada tombol yang digeser untuk mendapatkan kunci berikutnya

keys.append(permutation(temp, keyPermutationMatrix2))

# Kembalikan kunci yang dihasilkan

return keys

# Sbox yang digunakan dalam Algoritma DES

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):

"""Function to substitute all the bytes using Sbox."""

# Pisahkan bit array menjadi 6 potongan berukuran

# Untuk pengindeksan Sbox

blocks = nSplit(bitArray, 6)

result = []

for i in range(len(blocks)):

block = blocks[i]

# Nomor baris yang akan diperoleh dari bit pertama dan terakhir

row = int( str(block[0]) + str(block[5]), 2 )

# Mendapatkan nomor kolom dari bit posisi 2,3,4,5

column = int(''.join([str(x) for x in block[1:-1]]), 2)

# Mengambil nilai dari ith Sbox di ith round

sboxValue = SboxesArray[i][row][column]

# Ubah nilai sbox ke biner

binVal = binValue(sboxValue, 4)

# Menambahkan ke hasil

result += [int(bit) for bit in binVal]

# Mengembalikan hasil

return result

def addPadding(text):

"""Function to add padding according to PKCS5 standard."""

# Menentukan panjang padding

paddingLength = 8 - (len(text) % 8)

# Menambahkan jumlah panjang padding dari chr (paddingLength) ke teks

text += chr(paddingLength) \* paddingLength

# Returning text

return text

def removePadding(data):

"""Function to remove padding from plaintext according to PKCS5."""

# Mendapatkan panjang padding

paddingLength = ord(data[-1])

# Mengembalikan data dengan padding yang dihapus

return data[ : -paddingLength]

def expand(array, table):

"""Function to expand the array using table."""

# Mengembalikan hasil yang diperluas

return [array[element - 1] for element in table]

def permutation(array, table):

"""Function to do permutation on the array using table."""

# Mengembalikan hasil permutasi

return [array[element - 1] for element in table]

def leftShift(list1, list2, n):

"""Function to left shift the arrays by n."""

# Kiri menggeser dua array

return list1[n:] + list1[:n], list2[n:] + list2[:n]

def nSplit(list, n):

"""Function to split a list into chunks of size n."""

# Memotong dan mengembalikan array potongan ukuran n

# dan sisa terakhir

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

def xor(list1, list2):

"""Function to return the XOR of two lists."""

# Mengembalikan xor dari dua daftar

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

def binValue(val, bitSize):

"""Function to return the binary value as a string of given size."""

binVal = bin(val)[2:] if isinstance(val, int) else bin(ord(val))[2:]

# Menambahkan dengan jumlah nol yang diperlukan di depan

while len(binVal) < bitSize:

binVal = "0" + binVal

# Mengembalikan nilai biner

return binVal

def stringToBitArray(text):

"""Funtion to convert a string into a list of bits."""

# Inisialisasi variabel diperlukan

bitArray = []

for letter in text:

# Mendapatkan nilai biner (8-bit) dari huruf

binVal = binValue(letter, 8)

# Membuat daftar bit

binValArr = [int(x) for x in list(binVal)]

# Menambahkan bit ke array

bitArray += binValArr

# Mengembalikan jawaban

return bitArray

def bitArrayToString(array):

"""Function to convert a list of bits to string."""

# memotong array menjadi ukuran 8 bite

byteChunks = nSplit(array, 8)

# Inisialisasi variabel diperlukan

stringBytesList = []

stringResult = ''

# Untuk setiap byte

for byte in byteChunks:

bitsList = []

for bit in byte:

bitsList += str(bit)

# Menambahkan byte dalam bentuk string ke stringBytesList

stringBytesList.append(''.join(bitsList))

# Mengonversi setiap stringByte menjadi char (konversi basis 2 int terlebih dahulu)

# dan kemudian digabungkan

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

# Mengembalikan hasil

return result

if \_name\_ == '\_main\_':

main()

# In[ ]: