**program for DES algorithm for decryption, the 16 keys (K1, K2, c, K16) are used in reverse order. Design a key-generation scheme with the appropriate shift schedule for the decryption process.**

#!/usr/bin/env python3

# Minimal DES (single-block) decryption demo: generate subkeys and use them in reverse order.

PC1 = [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]

PC2 = [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]

SHIFT\_SCHEDULE = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

# Essential DES tables (abbreviated layout kept short)

IP = [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]

IP\_INV = [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]

E = [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]

P = [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-boxes (compact)

S\_BOXES = [

[[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]]

]

# --- bit helpers ---

def hex\_to\_bits(h):

n = int(h,16)

return [(n >> i) & 1 for i in reversed(range(len(h)\*4))]

def bits\_to\_hex(bits):

v = 0

for b in bits: v = (v<<1)|b

return format(v, '0{}x'.format((len(bits)+3)//4)).upper()

def permute(bits, table): return [bits[i-1] for i in table]

def left\_shift(bits, n): return bits[n:]+bits[:n]

def xor(a,b): return [x^y for x,y in zip(a,b)]

def sbox\_sub(bits48):

out = []

for i in range(8):

b = bits48[i\*6:(i+1)\*6]

row = (b[0]<<1)|b[5]

col = (b[1]<<3)|(b[2]<<2)|(b[3]<<1)|b[4]

val = S\_BOXES[i][row][col]

out += [(val>>3)&1,(val>>2)&1,(val>>1)&1,val&1]

return out

def feistel(R, subkey):

# expand, xor, sbox, p

e = permute(R, E)

x = xor(e, subkey)

s = sbox\_sub(x)

return permute(s, P)

# --- key schedule: produce K1..K16 ---

def generate\_subkeys(key\_hex):

key\_bits = hex\_to\_bits(key\_hex)

k56 = permute(key\_bits, PC1) # 56 bits

C = k56[:28]; D = k56[28:]

subkeys = []

for sh in SHIFT\_SCHEDULE:

C = left\_shift(C, sh); D = left\_shift(D, sh)

subkeys.append(permute(C+D, PC2)) # 48-bit subkey

return subkeys # K1..K16 in encryption order

# --- DES single-block (use subkeys in given order) ---

def des\_block(block\_hex, subkeys):

b = hex\_to\_bits(block\_hex)

ip = permute(b, IP)

L, R = ip[:32], ip[32:]

for k in subkeys:

L, R = R, xor(L, feistel(R, k))

pre = R + L

return bits\_to\_hex(permute(pre, IP\_INV))

# --- wrapper for decryption using reversed keys ---

def des\_decrypt\_block(cipher\_hex, key\_hex):

enc\_keys = generate\_subkeys(key\_hex) # encryption K1..K16

dec\_keys = enc\_keys[::-1] # decryption uses K16..K1

return des\_block(cipher\_hex, dec\_keys)

# --- simple CLI ---

if \_\_name\_\_ == "\_\_main\_\_":

ct = input("Ciphertext (16 hex chars): ").strip().replace(" ","")

key = input("Key (16 hex chars): ").strip().replace(" ","")

if len(ct)!=16 or len(key)!=16:

print("Both must be 16 hex chars (64 bits).")

else:

pt\_hex = des\_decrypt\_block(ct, key)

print("Decrypted (hex):", pt\_hex)

try:

txt = bytes.fromhex(pt\_hex).decode('latin-1')

print("Decrypted (ASCII):", txt)

except:

pass