class Blowfish:

def \_\_init\_\_(self, key):

self.P = [0x243F6A88, 0x85A308D3, 0x13198A2E, 0x03707344,

0xA4093822, 0x299F31D0, 0x082EFA98, 0xEC4E6C89,

0x452821E6, 0x38D01377, 0xBE5466CF, 0x34E90C6C,

0xC0AC29B7, 0xC97C50DD, 0x3F84D5B5, 0xB5470917,

0x9216D5D9, 0x8979FB1B]

self.S = [[0] \* 256 for \_ in range(4)]

self.key\_schedule(key)

def key\_schedule(self, key):

key\_length = len(key)

j = 0

for i in range(len(self.P)):

self.P[i] ^= (key[j % key\_length] << 24) | (key[(j + 1) % key\_length] << 16) | (key[(j + 2) % key\_length] << 8) | key[(j + 3) % key\_length]

j = (j + 4) % key\_length

# Generate subkeys

L = 0

R = 0

for i in range(0, len(self.P), 2):

L, R = self.encrypt\_block(L, R)

self.P[i] = L

self.P[i + 1] = R

for i in range(4):

for j in range(0, 256, 2):

L, R = self.encrypt\_block(L, R)

self.S[i][j] = L

self.S[i][j + 1] = R

def encrypt\_block(self, L, R):

L ^= self.P[0]

for i in range(1, 17):

L, R = R, L ^ self.f(R) ^ self.P[i]

L, R = R, L

L ^= self.P[17]

return L, R

def f(self, x):

return ((self.S[0][(x >> 24) & 0xFF] + self.S[1][(x >> 16) & 0xFF]) ^ self.S[2][(x >> 8) & 0xFF]) + self.S[3][x & 0xFF]

def encrypt(self, plaintext):

L = (plaintext >> 32) & 0xFFFFFFFF

R = plaintext & 0xFFFFFFFF

L, R = self.encrypt\_block(L, R)

return (L << 32) | R

# Example usage

key = [0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08]

blowfish = Blowfish(key)

plaintext = 0x1234567890ABCDEF

ciphertext = blowfish.encrypt(plaintext)

print(f"Ciphertext: {ciphertext:016X}")

