











import struct

def to\_32bit\_blocks(values):

while len(values) % 4 != 0:

values.append(0)

blocks = []

for i in range(0, len(values), 4):

block = struct.unpack('>I', bytes(values[i:i + 4]))[0]

blocks.append(block)

return blocks

def from\_32bit\_blocks(blocks):

values = []

for block in blocks:

bytes\_block = struct.pack('>I', block)

values.extend(bytes\_block)

return values

def ror32(value, shift):

shift &= 31

return ((value >> shift) | (value << (32 - shift))) & 0xFFFFFFFF

def rol32(value, shift):

shift &= 31

return ((value << shift) | (value >> (32 - shift))) & 0xFFFFFFFF

def xor32(value1, value2):

return (value1 ^ value2) & 0xFFFFFFFF

def reg\_encrypt(block, key):

for i in range(0, 3):

block = xor32(block, key)

return block

def reg\_decrypt(block, key):

for i in range(0, 3):

block = xor32(block, key)

return block

def encrypt(key, blocks):

key\_value = key

result = []

for block in blocks:

encrypted\_block = reg\_encrypt(block, key\_value)

result.append(encrypted\_block)

key\_value = block

return result

def decrypt(key, blocks):

key\_value = key

result = []

for block in blocks:

decrypted\_block = reg\_decrypt(block, key\_value)

result.append(decrypted\_block)

key\_value = decrypted\_block

return result

key = 0xABAD1DEA

encrypted\_blocks = [

0xD68C2DDA,

0x4D115F42,

0x721D2A00,

0x0745681C,

0x0C454800,

0x79631E29,

0x650C2409,

0x140D0B11,

0x040B1B1C

]

decrypted\_blocks = decrypt(key, encrypted\_blocks)

result = from\_32bit\_blocks(decrypted\_blocks)

original\_secret = ''.join(map(chr, result)).rstrip('\x00')[::-1]

print("Original Secret: " + original\_secret)