ez_AES

题目给出了秘钥扩展过程中的的部分数据

根据源码的加密方式

有

$$W[i] = W[i-4] \oplus W[i-1] (i\text{不是4的倍数})$$

$$W[i] = W[i-4] \oplus T(W[i-1]) (i\text{是4的倍数})$$

这里的T包含字循环，字节代换，轮常量异或三个部分

即

$$W_4 = W_0 \oplus T(W_3)$$

$$W_5 = W_1 \oplus W_4$$

$$W_6 = W_2 \oplus W_5$$

$$W_7 = W_3 \oplus W_6$$

我们的目的是恢复初始秘钥

而根据这里就能使用已知的两轮部分秘钥向前异或得到

已知

$$K9_0, K9_2, K10_0, K10_2$$

$$\text{即}k10_0 = K9_0 \oplus T(K9_3)$$

$$K9_3 = T^{-1}(k10_0 \oplus K9_0)$$

$$k10_3 = k9_3 \oplus K10_2$$

$$k10_1 = k9_2 \oplus k10_2$$

此时就得到了秘钥扩展过程中的完整的一轮，

就能往前预测了

```
from aes import AES
def xor(a, b):
    return bytes([x ^ y for x, y in zip(a, b)])

from Crypto.Util.number import *
Sbox = [0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b,
0xfe, 0xd7, 0xab, 0x76, 0xca, 0x82,
        0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4,
0x72, 0xc0, 0xb7, 0xfd, 0x93, 0x26,
        0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96,
        0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75, 0x09, 0x83,
0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0,
        0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84, 0x53, 0xd1, 0x00, 0xed,
0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb,
        0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf, 0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d,
0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f,
        0x50, 0x3c, 0x9f, 0xa8, 0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5,
0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff,
        0xf3, 0xd2, 0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7,
0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
        0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14,
0xde, 0x5e, 0x0b, 0xdb, 0xe0, 0x32,
        0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95,
0xe4, 0x79, 0xe7, 0xc8, 0x37, 0x6d,
        0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6,
        0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a, 0x70, 0x3e,
0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e,
        0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e, 0xe1, 0xf8, 0x98, 0x11,
0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e,
        0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf, 0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6,
0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f,
        0xb0, 0x54, 0xbb, 0x16]

r_con = [
    0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36,
    0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6,
    0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91
]

ROT = lambda x: ((x & 0xffffff) << 8) | ((x >> 24) & 0xFF)
S = lambda x: (Sbox[(x >> 24) & 0xff] << 24) | (Sbox[(x >> 16) & 0xff] << 16) |
(Sbox[(x >> 8) & 0xff] << 8) | (Sbox[x & 0xff])

inv_Sbox = [0 for i in range(len(Sbox))]

for i in range(len(Sbox)):
    inv_Sbox[Sbox[i]] = i

inv_S = lambda x:(inv_Sbox[(x >> 24) & 0xff] << 24) | (inv_Sbox[(x >> 16) & 0xff]
<< 16) | (inv_Sbox[(x >> 8) & 0xff] << 8) | (inv_Sbox[x & 0xff])
inv_ROT = lambda x:((x & 0xFF) << 24) | ((x >> 8) & 0xFFFFFF)
def su_hex(K):
    grouped_bytes = []
    for i in range(0, len(K), 4):
        group = K[i:i+4]
        hex_group = ''.join(f'{num:02x}' for num in group)
        grouped_bytes.append(bytes.fromhex(hex_group))
    return grouped_bytes

ks1=[[127, 106, 114, 135], [201, 200, 41, 139]]
ks2=[[156, 228, 163, 135], [209, 238, 234, 51]]
cc = b'\xbb\x0f\n\t\x11\xbd\xec~\xbb\x1d\xcf\xe1\xd0\xfd\x14q'




k1= su_hex(ks1[0])[0]
k3= su_hex(ks2[0])[0]
k5= su_hex(ks1[1])[0]
k7= su_hex(ks2[1])[0]

k44 = bytes_to_long(xor(k5,k1)) ^ ((r_con[9])<<24)
k44 = inv_S(k44)
k4 = long_to_bytes(inv_ROT(k44))
k8 = xor(k7,k4)
k6 = xor(k7,k3)

k7 = [k5,k6,k7,k8]
def inv_key(k5, k6, k7, k8, i):
    k4 = k8 ^ k7
    k3 = k7 ^ k6
    k2 = k6 ^ k5
    k1 = k5 ^ S(ROT(k4)) ^ (r_con[i-1] << 24)
    return k1, k2, k3, k4

key  = [bytes_to_long(i) for i in k7]

for i in range(10, 0, -1):
    key = inv_key(*key, i)

keys=b""
for i in key:
    keys+=long_to_bytes(i)
aes = AES(keys)

print(aes.decrypt(cc))
```