Week4-123456

crypto

1.ToyCipher_Linear

这道题加密函数是这样的

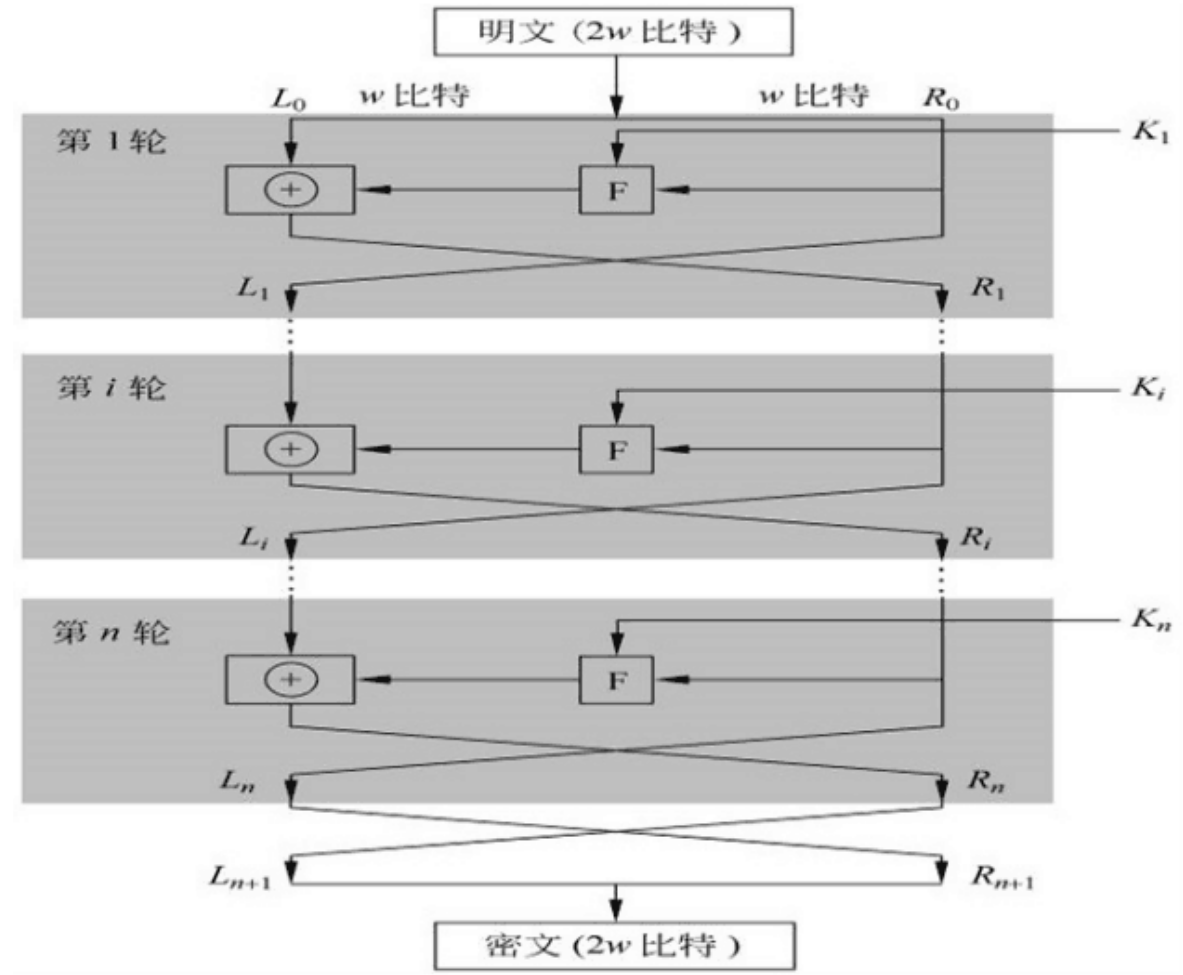
```
def f(x, roundkey):
    return rotL(x, 16, 7) ^ rotL(x, 16, 2) ^
roundkey


def ToyCipher(block, mode='enc'):
    '''Feistel networks'''
    roundKeys_ = ROUNDKEYS
    if mode == 'dec':
        roundKeys_ = roundKeys_[::-1]

    L, R = (block >> 16), (block % 2**16)
    for i in range(12):
        _R = R
        R = L ^ f( R, roundKeys_[i] )
        L = _R

    return (R << 16) | L
```

然后找到张这个图片



然后加密是先补全长度为4的倍数，然后每四个长度当成一块，然后分成l和r，然后左右交换一直变。然后源代码是加密12次，那么最后输出的L应该可以算作最开始的R经过变化得到，那我们可以再次encrypt一次，让L算作最开始的L经过变化得到。

然后因为我xor多个数，可以当成xor一个数。

最后的脚本：

```
import os, binascii
XOR = lambda s1, s2: bytes([x^y for x,y in zip(s1, s2)])
def rotL(x, nbits, lbits):
    mask = 2**nbits - 1
    return (x << lbits%nbits) & mask | ( (x & mask) >> (-lbits % nbits) )

def f(x):
    return rotL(x, 16, 7) ^ rotL(x, 16, 2)

def ToyCipher1(block, mode='enc'):
    '''Feistel networks'''
    # if mode == 'dec':
    #     roundKeys_ = roundKeys_[::-1]

    L, R = (block >> 16), (block % 2**16)
    for i in range(12):
        _R = R
        R = L ^ f(R)
        L = _R
```

```
    return (R << 16) | L

def encrypt1(plaintext):
    '''ECB mode'''
    # plaintext = pad(plaintext, BLOCKSIZE)
    ciphertext = b''
    for i in range( len(plaintext) // BLOCKSIZE ):
        block = plaintext[i*BLOCKSIZE:(i+1)*BLOCKSIZE]
        block = int.from_bytes(block, byteorder='big')
        E_block = ToyCipher1(block)
        ciphertext += E_block.to_bytes(BLOCKSIZE, byteorder='big')
    return ciphertext

BLOCKSIZE = 4

key = [0, 0]
test = b'just'
c = b'\x91a\xb1o'
c = encrypt1(c)
print(c)
for i in range(65536):
    if XOR(c[:2],i.to_bytes(2, byteorder='big')) == test[:2]:
        key[0] = i
    if XOR(c[2:4],i.to_bytes(2, byteorder='big')) == test[2:4]:
        key[1] = i

cipher =
b'\xe6\xf9\xda\xf0\xe18\xbc\xb4[\xfb\xbe\xd1\xfe\xa2\t\x8d\xdft:\xee\x1f\x1d\xe2
q\xe5\x92/$#DL\xOO\x1dD5@\xO1W?!7CQ\xc16V\xbO\x14q)\xaa2'
flag = b''
cipher = encrypt1(cipher)
for i in range(len(cipher) >> 1):
    block = cipher[i << 1 : i + 1 << 1]
    block = int.from_bytes(block, byteorder='big') ^ key[i % 2]
    flag += block.to_bytes(2, byteorder='big')
print(flag) #hgame{rOTAT!on_&&-xOr 4Re-bOtH~l1neaR_OpEr4t1On5}
```