

HGAME2020 Week4 Writeup

原创

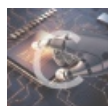
[wh201906](#) 于 2020-02-14 23:24:36 发布 988 收藏

分类专栏: [CTF试水](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/wh201906/article/details/104321705>

版权



[CTF试水](#) 专栏收录该内容

4 篇文章 0 订阅

订阅专栏

完结撒花

Crypto - ToyCipher_Linear

题目:

Why encryption based on XOR and Rotation is easy to break?

还有加解密的脚本:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import os, binascii

from secret import flag

def rotL(x, nbits, lbits):
    mask = 2**nbits - 1
    return (x << lbits%nbits) & mask | ( (x & mask) >> (-lbits % nbits) )

def rotR(x, nbits, rbits):
    return rotL(x, nbits, -rbits%nbits)

def keySchedule(masterkey):
    roundKeys = [ ( rotR(masterkey, 64, 16*i) % 2**16 ) for i in range(12) ]
    return roundKeys

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

def pad(data, blocksize):
    pad_len = blocksize - (len(data) % blocksize)
    return data + bytes( [pad_len] * pad_len )

def unpad(data, blocksize):
    pad_len = data[-1]
    _data = data[:-pad_len]
    assert pad(_data, blocksize)==data, "Invalid padding."
    return _data

def encrypt(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 = ToyCipher(block)
        ciphertext += E_block.to_bytes(BLOCKSIZE, byteorder='big')
    return ciphertext

def decrypt(ciphertext):
    '''ECB mode'''
    plaintext = b''
    for i in range( len(ciphertext) // BLOCKSIZE ):
        block = ciphertext[i*BLOCKSIZE:(i+1)*BLOCKSIZE]
        block = int.from_bytes(block, byteorder='big')
        D_block = ToyCipher(block, 'dec')
        plaintext += D_block.to_bytes(BLOCKSIZE, byteorder='big')
    plaintext = unpad(plaintext, BLOCKSIZE)
    return plaintext

masterkey = os.urandom(8)
masterkey = int.from_bytes(masterkey, byteorder='big')
ROUNDKEYS = keySchedule(masterkey)
BLOCKSIZE = 4

cipher = encrypt(b'just a test')

print(cipher)
print(decrypt(cipher))
print(encrypt(flag))

# b'\x91a\xb1o\xed_\xb2\x8c\x00\x1b\xdfp'
# b'just a test'
# b'\xe6\xf9\xda\xf0\xe18\xbc\xb4[\xfb\xbe\xd1\xfe\xa2\t\x8d\xdf:\xee\x1f\x1d\xe2q\xe5\x92/$#DL\x00\x1dD5@\x01W
?!7CQ\xc16V\xb0\x14q)\xaa2'

```

感谢出题人在校内群里提供的[资料](#)，要不然一点思路都没有

先大概分析了一下代码。RotL函数会将x左移lbits位，并将多出来的部分接在x的右侧。例如设x=110101，则RotL(x,6,4)的结果为011101。RotR函数则是右移rbits位，操作与RotL函数类似。

根据资料和题目名，在网上搜索线性攻击，发现线性攻击当中涉及到随机置换，而脚本中似乎不涉及搜索针对Feistel网络结构的攻击，无果

仔细阅读资料，发现提问者设计了一个基于旋转和异或操作的加密方法（与此题类似），之后还给出了根据明文和密文恢复密钥的方法

$$\begin{array}{lll} c1 + k1 = p3 & 1 + k1 = 1 & k3 = 1 \\ c0 + k0 = p2 & ==> 0 + k0 = 0 & ==> k2 = 0 \\ c3 + k3 = p1 & 1 + k3 = 0 & k1 = 0 \\ c2 + k2 = p0 & 1 + k2 = 1 & k0 = 0 \end{array} \quad (A)$$

进一步分析，本题的加/解密过程本质还是若干个位的异或操作，而旋转改变了位与位之间的对应关系。只要找到解密过程中明文每一个位是由哪些位异或而得到的（明文bit与密文bit、密钥bit的对应关系），就可以得到一个异或方程组，通过已知的明文和密文就能把密钥恢复出来，进而解密出flag

因为加解密过程是ECB模式，因此接下来只需要对单个block进行分析即可

一开始准备手动模拟一遍，后来意识到任务量巨大，于是仿照解密逻辑写了个脚本，寻找并显示对应关系：

```

from copy import deepcopy

def rotL(x, lbits):
    return x[lbits:] + x[:lbits]

roundkeys = []
for i in range(12):
    rndkey = []
    for j in range(16):
        out = 'k{k_index}_{k_bit}'.format(k_index=i, k_bit=j)
        rndkey.append(out)
    roundkeys.append(rndkey)

block = []
for i in range(32):
    b = []
    b.append(i)
    block.append(b)

L, R = block[:16], block[16:]
for i in range(12):
    _R = deepcopy(R)

    currkey = deepcopy(roundkeys[i])
    tmp = sum(R, [])
    rt12 = rotL(tmp, 2)
    rt17 = rotL(tmp, 7)
    for j in range(16):
        L[j].append(currkey[j])
        L[j].append(rt12[j])
        L[j].append(rt17[j])
    R = deepcopy(L)
    L = deepcopy(_R)

for i in range(16):
    for j in R[i]:
        times = R[i].count(j)
        if times % 2 == 0:
            for k in range(times):
                R[i].remove(j)
        else:
            for k in range(times - 1):
                R[i].remove(j)
    print(R[i])
for i in range(16):
    for j in L[i]:
        times = L[i].count(j)
        if times % 2 == 0:
            for k in range(times):
                L[i].remove(j)
        else:
            for k in range(times - 1):
                L[i].remove(j)
    print(L[i])

```

理论上通过这个脚本可以得到明文的第*i*位与密文密钥的对应关系，并且这个关系不会因为加密内容和密钥的不同而改变也就是对于明文M，密文C和密钥K(roundkeys)，无论三者如何变化，总有

```
M[0]=C[a]^C[b]^...^K[c]^K[d]^...
M[1]=C[e]^C[f]^...^K[g]^K[h]^...
M[2]=C[i]^C[j]^...^K[k]^K[l]^...
M[...]=...
```

而脚本要寻找的就是a,b,c,d,...的值并将其显示出来

之后就可以通过已知的M和已知的C求出K，进而解出flag

进一步分析，将上述每一条表达式中C[x]^C[x]^...的结果记为midC，K[x]^K[x]^...的结果记为midK，则上述式子可以改写为

```
M[0]=midC_0^midK_0
M[1]=midC_1^midK_1
M[2]=midC_2^midK_2
M[...]=...
```

那么选取两组不同的明文和密文（从已知信息中选取两个不同的block），通过刚才的方法可以得到两个不同的midC_0，如果两者异或的结果等于两者对应M[0]相异或的结果，则可以验证刚才的分析结论

然而可能是因为写出来的脚本有问题，选取b'just', b' a t'（因为题中BLOCKSIZE = 4）和其对应的密文进行上述操作，无法验证刚才的结论

之后仔细阅读题目代码，发现如果将f函数中的

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

改为

```
return rotL(x, 16, 7) ^ rotL(x, 16, 2)
```

也可以达到计算midC的效果，并且由于是直接修改题目，所以出错的机率要低一些

复制题目代码，删掉 ^ roundkey以及加密flag的部分，然后插入以下语句：

```
midC1 = encrypt(C[:4])
midC1 = int.from_bytes(midC1, 'big')
print(bin(midC1))

midC2=encrypt(C[4:8])
midC2 = int.from_bytes(midC2, 'big')
print(bin(midC2))
```

得到

```
midC1 = 0b1000000100100011010011011011110
midC2 = 0b000101010000101111101011011110
```

计算midC1 ^ midC2 = 1242845952

直接将M[:4] (b'just')和M[4:8] (b' a t')转为数字后异或，得到的结果也是1242845952，说明这一思路是正确的

因为M=midC ^ midK，所以midK=M^midC。将M[:4] (b'just')与midC1异或，得到midK=719639978，而这个midK对于题中的所有密文block都是适用的

之后将加密后的flag

```
b'\xe6\xf9\xda\xf0\xe18\xbc\xb4[\xfb\xbe\xd1\xfe\xa2\t\x8d\xdft:\xee\x1f\x1d\xe2q\xe5\x92/$#DL\x00\x1dD5@\x01W?!
7CQ\xc16V\xb0\x14q)\xaa2'
```

拆成13个长度为4的block，分别计算每个块的midC，并将其与midK异或即可得到flag

最终flag: hgame{r0TAT!on_&&-x0r 4Re-b0tH~l1nearR_0pEr4t1On5}

希望有师傅能帮我找一下之前的脚本错在哪儿，不胜感激！

2020.02.14