[GKCTF 2021]XOR

原创

mortal15 于 2021-09-30 00:03:56 发布 1409 收藏 2
分类专栏： Crypto 文章标签： 算法 网络安全

简介

这题和之前做的dfs类似，算是个高级版的爆破吧，是个挺好的例子，前面的脚本可以解决第一部分，5space signin(未知低位的dfs和coppersmith)_a5555678744的博客-CSDN博客在这里就不赘述了。重点来说说本题的第二部分。

题解

参照知乎大佬的算法，自己手敲了个自己比较容易看懂的版本

[GKCTF] GKCTF x DASCTF应急挑战杯 个人(or团队) Crypto方向 writeup - 知乎

```
from Crypto.Util.number import *
from hashlib import md5
'''
a = getPrime(512)
b = getPrime(512)
c = getPrime(512)
d = getPrime(512)
d1 = int(bin(d)[2:][::-1] , 2)
n1 = a*b
x1 = a^b
n2 = c*d
x2 = c^d1
flag = md5(str(a+b+c+d).encode()).hexdigest()
print("n1 =",n1)
print("x1 =",x1)
print("n2 =",n2)
print("x2 =",x2)
'''
n1 = 838763494437926958008581070260411839823209237328177881964030384369078520459686780327443648205912546537
x1 = 470074176751536775598897975923770635978979028109069024580032435083767762464518452611002794398395269024
n2 = 652881484543771018418888718488067046944779065870107552864512166327018684577228481396960369285618888507
x2 = 360438668861232087414353226298838456221365979857858321089214326157690828111222335667890008387032752724
```

要搞定这种算法题，首先要确定约束条件，这些条件里面有的看似没有什么意义，但是却很关键，所以也很头疼。

首先设一些量，cl是c的低位，ch是c的高位，dl和dh同理

这道题里的约束有

1. $n2 = c * d$

2. $x2 = c^{d1}$

3. $cl * ch\%mask = n2\%mask$

4. 不确定位全部填1时，c*d > n2

5. 不确定位全部填0时，c*d < n2

有了这些约束后，耐心地写算法。

```
n1 = 652881484543771018418888718488067046944779065870107552864512166327018684577228481396960369285618888507

x1 = 360438668861232087414353226298838456221365979857858321089214326157690828111222335667890008387032752724

cl_list, dl_list, ch_list, dh_list = [1], [1], [1], [1]

x1_bits = [int(x) for x in f'{x1:0512b}'[::-1]]
print(x1_bits)
print('\n')
mask = 2
for i in range(1,256):
    mask*=2
    scl_list, sdl_list, sch_list, sdh_list = [], [], [], []
```

list数组存的是对四个变量的可能情况

s_list则是临时使用的用于迭代的四个变量

x1_bits就是把x1的二进制按位分开

mask跟每一轮处理位数相关了，mask*2就是要处理下一个位了

这里不太一样的就是，由于我们一开始就可以确定最高位和最低位必定是1，那么就可以直接从第二位开始，这样数值上也更方便。

随后按照约束写出如下循环:

```
    for j in range(len(cl_list)):
        for cl in range(2):
            for dl in range(2):
                for ch in range(2):
                    for dh in range(2):
                        if (cl ^ dh == x1_bits[511-i] and ch ^ dl == x1_bits[i]):
                            temp1 = ((mask // 2 * cl + cl_list[j]) * (mask // 2 * dl + dl_list[j]))%mask
                            temp2 = n1 % mask
                            if (temp1 == temp2):
                                g1 = bin(ch_list[j])[2:] + bin(ch)[2:] + '1' * (510 - 2 * i) + bin(cl)[2:]
                                g1 = int(g1, 2)
                                g2 = bin(dh_list[j])[2:] + bin(dh)[2:] + '1' * (510 - 2 * i) + bin(dl)[2:]
                                g2 = int(g2, 2)
                                if(g1 * g2 < n1):
                                    continue
                                g1 = bin(ch_list[j])[2:] + bin(ch)[2:] + '0' * (510 - 2 * i) + bin(cl)[2:]
                                g1 = int(g1, 2)
                                g2 = bin(dh_list[j])[2:] + bin(dh)[2:] + '0' * (510 - 2 * i) + bin(dl)[2:]
                                g2 = int(g2, 2)
                                if (g1 * g2 > n1):
                                    continue
                                scl_list.append(mask // 2 * cl + cl_list[j])
                                sch_list.append(ch_list[j]*2 + ch)
                                sdl_list.append(mask // 2 * dl + dl_list[j])
                                sdh_list.append(dh_list[j] * 2 + dh)
    cl_list,dl_list,ch_list,dh_list = scl_list,sdl_list,sch_list,sdh_list
```

这里要特别注意低位的补零问题，高位零不补可以，但是低位如果最高位是零，会使高位整体往下降一级，所以必须要用zfill(i)来填充。

分析的细节就不说了，总之就是按照五重约束来构造算法。

这样就得到c，d的低位和高位了，拼接起来

```
d=int(bin(ch_list[0])[2:]+bin(cl_list[0])[2:].zfill(256),2)
c=int(bin(dh_list[0])[2:]+bin(dl_list[0])[2:].zfill(256),2)
```

完整的题解代码如下：

```
import itertools

n1 = 652881484543771018418888718488067046944779065870107552864512166327018684577228481396960369285618888507

x1 = 360438668861232087414353226298838456221365979857858321089214326157690828111222335667890008387032752724

cl_list, dl_list, ch_list, dh_list = [1], [1], [1], [1]

x1_bits = [int(x) for x in f'{x1:0512b}'[::-1]]
print(x1_bits)
print('\n')
mask = 2
for i in range(1,256):
    mask*=2
    scl_list, sdl_list, sch_list, sdh_list = [], [], [], []
    for j in range(len(cl_list)):
        for cl in range(2):
```

```
            for dl in range(2):
                for ch in range(2):
                    for dh in range(2):
                        if (cl ^ dh == x1_bits[511-i] and ch ^ dl == x1_bits[i]):
                            temp1 = ((mask // 2 * cl + cl_list[j]) * (mask // 2 * dl + dl_list[j]))%mask
                            temp2 = n1 % mask
                            if (temp1 == temp2):
                                g1 = bin(ch_list[j])[2:] + bin(ch)[2:] + '1' * (510 - 2 * i) + bin(cl)[2:]
                                g1 = int(g1, 2)
                                g2 = bin(dh_list[j])[2:] + bin(dh)[2:] + '1' * (510 - 2 * i) + bin(dl)[2:]
                                g2 = int(g2, 2)
                                if(g1 * g2 < n1):
                                    continue
                                g1 = bin(ch_list[j])[2:] + bin(ch)[2:] + '0' * (510 - 2 * i) + bin(cl)[2:]
                                g1 = int(g1, 2)
                                g2 = bin(dh_list[j])[2:] + bin(dh)[2:] + '0' * (510 - 2 * i) + bin(dl)[2:]
                                g2 = int(g2, 2)
                                if (g1 * g2 > n1):
                                    continue
                                scl_list.append(mask // 2 * cl + cl_list[j])
                                sch_list.append(ch_list[j]*2 + ch)
                                sdl_list.append(mask // 2 * dl + dl_list[j])
                                sdh_list.append(dh_list[j] * 2 + dh)
    cl_list,dl_list,ch_list,dh_list = scl_list,sdl_list,sch_list,sdh_list

print(cl_list)
print(dl_list)
print(ch_list)
print(dh_list)




d=int(bin(ch_list[0])[2:]+bin(cl_list[0])[2:].zfill(256),2)
c=int(bin(dh_list[0])[2:]+bin(dl_list[0])[2:].zfill(256),2)

print(c)
print(d)
print(c * d - n1)
print(((c ^ int(bin(d)[2:][::-1],2)) - x1))

a=78361471396106552237114697472001640694848788946261668706647406377866094681645553548746194977532775602809 3
b=10703774182571073361112791376032380096360697926840362483242105878115552437021674861528714598089603406032 8
c=80469254367102041924383040558747788658954169969708438696988588656039534111703695269977842242104917691403 8
d=81134277890200785266828179169439421534891877861073079587655860326107413542892805392648534697836213150493 8
print(bin(c ^ int(bin(d)[2:][::-1],2)))
import hashlib
flag = hashlib.md5(str(a+b+c+d).encode()).hexdigest()
print("flag{%s}"%flag)
```