2022年 HSC-1th中CRYPTO的BABY-RSA

原创

沐一 · 林 于 2022-02-28 16:57:18 发布 139 收藏

分类专栏： CTF 密码学 文章标签： ctf

本文链接：https://blog.csdn.net/xiao__1bai/article/details/123185826

CTF 同时被 2 个专栏收录

167 篇文章 6 订阅

订阅专栏

密码学

51 篇文章 1 订阅

订阅专栏

2022年 HSC-1th中CRYPTO的BABY-RSA

照例下载附件，是 py 文件：

```
from Crypto.Util.number import *


def lfsr(status,mask):
    out = (status << 1) & 0xffffffff
    i=(status&mask)&0xffffffff
    lastbit=0
    while i!=0:
        lastbit^=(i&1)
        i=i>>1
    out^=lastbit
    return (out,lastbit)

status= 1
mask = 0b10110001110010011100100010110101

num = bytes_to_long(m)

p = getPrime(1024)
q = getPrime(1024)
n = p*q
e = 65537

hp = bin(p)[2:]
c = pow(num, e, n)

print("n=",n)
print("c=",c)

f=open("key","w+",encoding='utf-8')
for i in range(568):
    curnum = int(hp[i])
    (status,out)=lfsr(status,mask)
    f.write(str(curnum ^ out))
f.close()

'''
n= 9363543374665338283861145656340115756598328744870620756798779080826725774691364161648335378062705433999048117
5643566370944219308861663698323534055278551045762026586476515247581799628556924611548599619035319901722797640991
9915718116777530795069049296985982992680895096412067808246044884792707448756861953656874030164998855547649020669
3181162301088156855926656544441682939839165455244630182978802660669255401576213941067679888164237586879364615664
9422342478962141952625109353459225128316323857417358101227301303665216128345565658386237088287800933233103482426
5477824729343085356605470399178143254262527139624650057670 3
c= 3641304537029815746727163894554557322382012539953948183406308231174259571263608621970973671202001456955622458
3713034247508150175781040699248778811627076739354969255294127486632098846283206570341907023489248147942630414832
6037796056953086938661992142541532391296430597977690959820020223691282396886748569610169187958079900024071577801
0424877093758489309380968229017074542588151574195295436881889313935734282141447498134543053106463951864974512375
3140914407131650471885906934319385998223405889345917125929956223345227999145635286307056876479508949289659131997
7220982550800127412055650822024806964785136056760965651778 9
```

.

.

一看 lfsr 我开心了一下，我研究过这个：

> https://blog.csdn.net/xiao__1bai/article/details/120392307

后来又哭了，果然还是研究得不透彻，稍微变一下式就卡住了，就那里怎么求出原来的 `out` 就想了很久~

后来在不断尝试后逐渐有了思路 `lfsr` 加密函数中传入的两个参数 `status`,`mask` 都是`固定`的，那么 `return (out,lastbit)` 不也是固定的吗？这样就可以求出原来的 out 了啊!

```
from Crypto.Util.number import *
#这里只取out即可
m = open("flag.txt","rb").read()
def lfsr(status,mask):
    out = (status << 1) & 0xffffffff
    i=(status&mask)&0xffffffff
    lastbit=0
    while i!=0:
        lastbit^=(i&1)
        i=i>>1
    out^=lastbit
    return (out,lastbit)

status= 1
mask = 0b10110000111001001110010000110110101

num = bytes_to_long(m)

p = getPrime(1024)
q = getPrime(1024)
n = p*q
e = 65537

hp = bin(p)[2:]
c = pow(num, e, n)

#print("n=",n)
#print("c=",c)
#print("hp=",hp)
f=open("key","w+",encoding='utf-8')
for i in range(568):
    curnum = int(hp[i])
    (status,out)=lfsr(status,mask)
    print(out,end='')
    f.write(str(curnum ^ out))
f.close()
'''
out=
11011101010110110111111000011011001111111011000000100111010011101011010111100001000011101111100100000000101111100110001011000110010100101100001001111011110010000011011001010110001100000111110001111110001010101001101011110011110000100110100110000110011001101010100111000100000100110010010011001110000010110010110100101101100100111000011010100001000101010000111001000010111010111000000000101010110000111111100010011111100101000110001001110100000100101100010010000111001010111111011010000100100100000010010010111111110100000001100010000100110000101101001010010
'''
```

```
└─$ python 21.py
1101110101011011011111100001101100111111110110000001001110100111010110101111000010000111011111001000000010111110
0110010110001100101001011000010011110111100100000110110010101100011000001111100011111100010101010011010111001111
1000010011010011000011001100110101010011100010000010011001001001100111000000101100101101001011011001001110000110
1010000100010101000011100100001011010111000000000101010110000111111100010011111100101000110001001110100000100101
1000100101111001000000100100011001010111110110100001001001000000100100101111111101000000011000100001001100001011
10010010
```

.

.

用上面的结果再异或回结果 `key`，就能得出 `568` 位的 `curnum` ，这就是 `RSA` 关键参数 `P` 的一部分啊：

```
out=list("1101110101011011011111100001101100111111110110000001001110100111010110101111000010000111011111001000000
0010111110011000101100011001010010110000100111101111001000001101100101011000110000011111000111111000101010100110
1011100111100001001101001100001100110011010101001110001000001001100100100110011100000010110010110100101011011001001
1100001101010000100010101000011100100001011010111000000000101010110000111111100010011111100101000110001001110100000
0010010110001000101111001000000100100011001010111110110100001001001000000100100101111111101000000011000100001001
100001011010010010")
#out输出与RSA无关
key=list("0101110100100111011011011000111010000111101000101010100100100011010111011000010010100101110110011101110
1100101000101110011100100111010101110110011000110110101100010100111111111101001101010101011101001100110101101101
1101100001100101010100000101110100110111011000111010101100001111010001101110010110110100100011001010011100011100111
1101010101101111111001011110010111100101000010001010000100011101001101111101001110110001110101101001101011000110
1110110110000011001001100110110000011000011010010101001001011010110010111110111000001001110111001010111010001110101
100110111111001010")
for i in range(568):
 print(int(out[i])^int(key[i]),end='')
#hp=list("1000000001111100000100111001010110111000000100101000111001101101111010000110010110101011001000001101110
0100101010001110010110100001001111011010000011000101000110010001010101001111000110010100100000101011100111111100001
0100001100001000110010010011001010111101101101001001110001101001000100111110001110101111010010000001010101110100
0001011000111111101101111011000101100010001100011001000010100100010110000000100010100011101111111001011110001011101
1100100000001110111100000010110010001001111111001010010011011110010110111100101010110110000011001011011000001010000
00011110010101011000")
```

```
└─$ python 29.py
1000000001111100000100111001010110111000000100101000111001101101111010000110010110101011001000001101110100101010
0011100101101000010011110110100000110001010001100100010101010011110001100101001000010101110011111111000101000011000
0001000110010010011001010111101101101001001110001101001000100111110001110101111010010000001010101110100001011000
1111110110111101100010110001000110001100100001010010001011000000100010100011101111111001011110001011101100100000
0011101111000001011001000100111111100101001001101111001011011110010101011011000001100101101100000010100000111100101011000
```

.

.

然后继续用关键字查资料，发现 `568` 位可以实现 `RSA` 高位攻击，恢复 `P`：

```
相关的脚本一大堆：https://www.jianshu.com/p/e407be39a22b?from=singlemessage
```

```
由于没下sage，用在线网址：https://sagecell.sagemath.org/
```

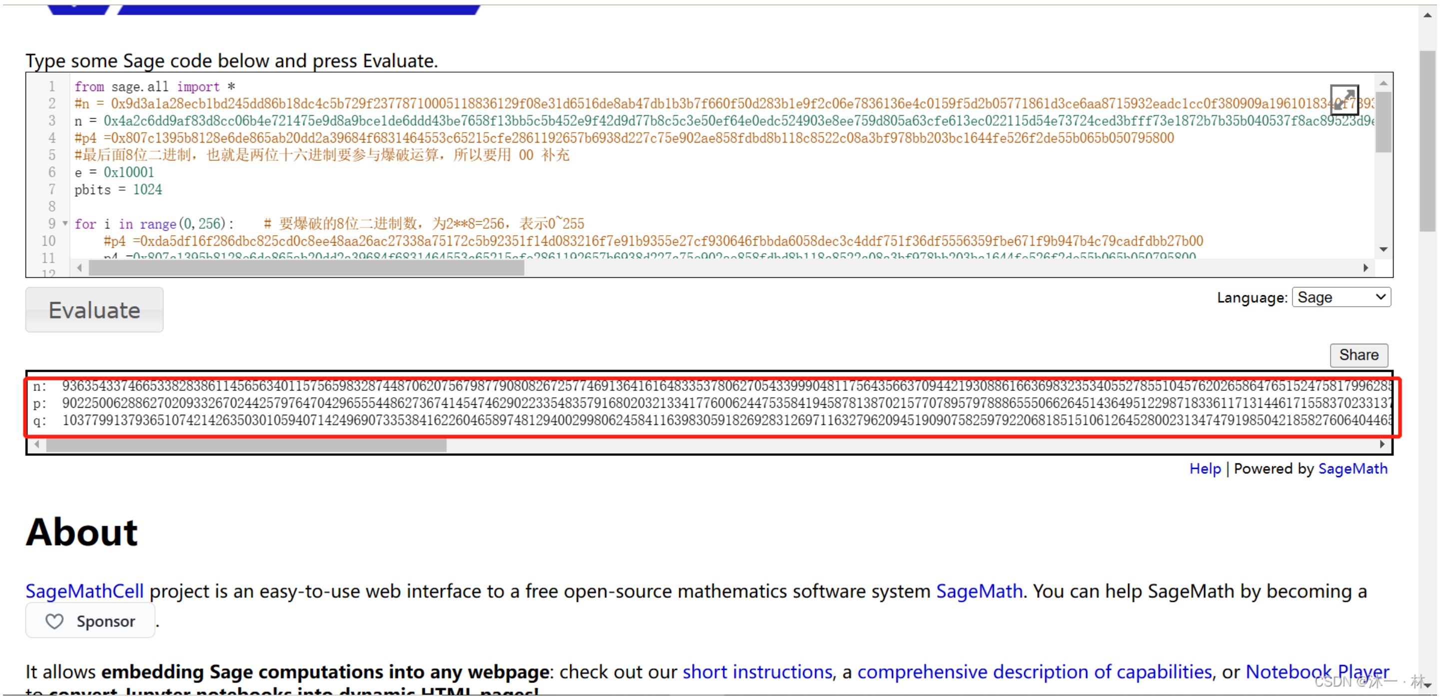
.

.

一开始直接用别人的脚本发现很多错，后来发现是`sage`语法变了，`print`要加括号了：

```
from sage.all import *
#n = 0x9d3a1a28ecb1bd245dd86b18dc4c5b729f23778710005118836129f08e31d6516de8ab47db1b3b7f660f50d283b1e9f2c06e7836136e4c0159f5d2b05771861d3ce6aa8715932eadc1cc0f380909a1961018340f7393142f9c177b1187151f97ac8cdc4ad17fa59a0f39d192af555f27de9cc800846eb2ca6ce78f87c0c0fbf47828328392b81771af624389fd779d130d80739bb7a608961125ba3f1800c766440fa70bfd3f834294d47d7ed9cfffd6d14ae18310f6c1d6d8f88b6c5d72a0b45608b4e21bbb8e314220ed7a2d6a8c95454e571c71b50f1d6a823778ca47131f5b889a1ed1957248bee8c4ac66872a5fd58a121560a27bad4958f1c763f2ffddL
n = 0x4a2c6dd9af83d8cc06b4e721475e9d8a9bce1de6ddd43be7658f13bb5c5b452e9f42d9d77b8c5c3e50ef64e0edc524903e8ee759d805a63cfe613ec022115d54e73724ced3bfff73e1872b7b35b040537f8ac89523d9e2860199d6d0b1c4d7830ee5b468bd7406990ffa29caa2d8fad285b3dba209b34b427d749d7e2aebded78f49e5017bfeec1cb9f72e63506d82af561a4858f652d3fb152526c10c7e4c5e15c84803efac675fb9297d915bd1e2eda5a5de3d48bbf68380303e0d8de81704fff8c9f07ae4d15212b9066227583345425ba7a04e06fd0c16ec6bfdd764318587d1bfe76a9834043b16392018e192456cb3ea994d2a187cabfa706efbee8dbfL
#p4 =0x807c1395b8128e6de865ab20dd2a39684f6831464553c65215cfe2861192657b6938d227c75e902ae858fdbd8b118c8522c08a3bf978bb203bc1644fe526f2de55b065b050795800
#最后面8位二进制，也就是两位十六进制要参与爆破运算，所以要用 00 补充
e = 0x10001
pbits = 1024

for i in range(0,256):    # 要爆破的8位二进制数，为2**8=256，表示0~255
    #p4 =0xda5df16f286dbc825cd0c8ee48aa26ac27338a75172c5b92351f14d083216f7e91b9355e27cf930646fbbda6058dec3c4ddf751f36df5556359fbe671f9b947b4c79cadfdbb27b00
    p4 =0x807c1395b8128e6de865ab20dd2a39684f6831464553c65215cfe2861192657b6938d227c75e902ae858fdbd8b118c8522c08a3bf978bb203bc1644fe526f2de55b065b050795800
    p4=p4+int(hex(i),16)
#    print hex(p4)
    kbits = pbits - p4.nbits()
#    print p4.nbits()
    p4 = p4 << kbits
    PR.<x> = PolynomialRing(Zmod(n))
    f = x + p4
    roots = f.small_roots(X=2^kbits, beta=0.4)
#经过以上一些函数处理后，n和p已经被转化为10进制
    if roots:
        p = p4+int(roots[0])
        print ("n: ", n)
        print ("p: ", p)
        print ("q: ", n/p)
        break
```



.

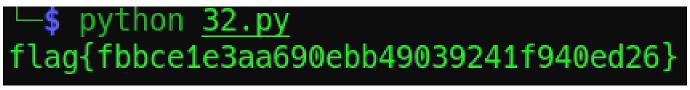
.

得出 n、p、q 后直接常规解密即可：

```
import libnum
from Crypto.Util.number import long_to_bytes

p = 90225006288627020933267024425797647042965555448627367414547462902233548357916802032133417760062447535841945878138702157707895797888655506626451436495122987183361171314461715583702331375674171604199315915509352276941674246168381004104536192633494611554748723427252091424949695486490446763447116750968954990847 7
q = 1037799137936510742142635030105940714249690733538416226046589748129400299806245841163983059182692831269711632796209451909075825979220681851510612645280023134747919850421858276064044656147150822788765916004528092853543075827672659991342372777325066714638341019562139613093669517061067890058307727841518630393 39
n = p * q
e = 65537
c=36413045370298157467271638945545573223820125399539481834063082311742595712636086219709736712020014569556224583713034247508150175781040699248778811627076739354969255294127486632098846283206570341907023489248147942630414832603779605695308693866199214254153239129643059797769095982002022369128239688674856961016918795807990002407157780104248770937584893093809682290170745425881515741952954368818893139357342821414474981345430531064639518649745123753140914407131650471885906934319385998223405889345917125929956223345227999145635286307056876479508492896591319977220982550800127412055650822024806964785136056760965651 7789

d = libnum.invmod(e, (p - 1) * (q - 1))
m = pow(c, d, n)
string = long_to_bytes(m)
print(string.decode())
```



.

.

解毕！

敬礼！