Asis CTF 2013: "RSAng"

原创

Gunther17 于 2017-08-12 22:06:18 发布 1968 收藏

本文链接：https://blog.csdn.net/dongyanwen6036/article/details/77131442

Writeup author:Gunther

Task text:

```
This message is encrypted using the prime factors of N. Decrypt the message.


N  =  136463296143190893248608270448493439350683115492680876168052084053504347848230208722375209906076869 05
enc_1 =  418816066259901017433354824768546553203335129738752481581177399267147997742261876440194689319723 54
enc_2 =  604904966174112956133930398825016261770335974197019935674937809953530879952111929616347581356745 86
enc_3 =  929780280856422867885792773522425861017683122345546248210815396295328331111250193377964254083155 85
enc_4 =  713095983409993528016630065648951013409887429450761159296479210782308084847050785300379490902448 64
enc_5 =  103097783116741358361451624751422998904453285709988318681568899902272752872049547121951413115492 79
enc_6 =  893953590726129543843388657738643296829192616227792491297994138018950400032231015978789461901555 00
```

http://www.shiyanbar.com/ctf/1811
RSANG(Asis CTF 2013):
这是一条被N的素因子加密的信息，尝试解密它。
解题链接： http://ctf5.shiyanbar.com/crypto/enc.txt
题目描述：这是一条被N的素因子加密的信息，尝试解密它。

分析：

1.首先题目已经给出了提示：‘RSAng’：RSA--需要的算法，ng--未知，推测利用RSA的模数超过2个素因子（不对，只是想多了）

这个模数究竟有多少位，我们看看便知道超级大哈哈,利用bin(N).

我们设N=2^h我们就通过现成的命令来求取，看下图：我们求得h=5568=4M=====>M=1392（下面要用）

```
>>> bin(8)
'0b1000'
>>> len(str(bin(8)))-3
3
>>> len(str(bin(16)))-3
4
>>> len(str(bin(13646329614319089324860827044849343935068311549268087616805208405350434784823020872237
52099060768690528968674833364057782055067774562398891952547514047204235995798564491183140299263415043 6
83220066032401571220068870523522605849754710457260615496198206486077565541882359273029321540828234247 4
54275351385175800590791826229826861686767276907109353344483424600073671005618671354567830676790521029 5
14003170122761357218436725574887117860970283001224875042462073434086906821751358168942479366907352466 7
39852775042370714098594011855552904791546899544423960785471663888719783218632151234452699483243471228 3
02705518541303657419611494430913054189008213602869677058106243805684221933109014914516642332714868354 2
80156027128285278392377607124982042081146064713624106908707104990208853041519285706323153304372353309 2
40712365285491521430923288334840163729768178092994303397521061232727066676909807741185453387613909190 4
68323389289130591063510870152731778953801492173733955809484285219619290522916182614097151450380269206 8
52978852356834341072182495485186595744932038368700700936747048784984651042381788778334716401689995519 8
10259582718655844575826495956644996354891678983988264290846862793999595377241872682107607183406399393 8
97666817032136954095510292657847491531800835643477336764344921222901669268228555521757123463501045339 0
10796056691922891784181655849073262103491526092655638815421510498626428820253606046920842647521827988 9
27024146381807814033982166166664728637583713300582994062333595677504137482579535988436516123997099456 0
83390731915785738779853626193017680164850178151714867439340379359021502282103400931873870365807639416 3
79895111555976437229840576270865174793921721080316632455532695)))-3
5568
>>> 5568/4
1392.0
```


由bin(N)得到的结果你会想到一些方法：那些很多0-1

介绍几个公式

The log 2为底 的模数 5568 so

$$N = (2^M + a)(2^M + b)(2^M + c)(2^M + d)$$

$$N = 2^{4M} + 2^{3M}(a+b+c+d) + 2^{2M}(ab+ac+ad+bc+bd+cd) + 2^M(bcd+acd+abd+abc) + abcd$$

Let:

$$p = a + b + c + d$$

$$q = ab + ac + ad + bc + bd + cd$$

$$r = bcd + acd + abd + abc$$

$$s = abcd$$

我们已经知道p q r s的情况下怎么去求a b c d.我们学过数学的知道这样的等式。这里2^M替换X理解

$$(X - x_1)(X - x_2)(X - x_3)(X - x_4) = 0$$

$$X^4 - (x_1+x_2+x_3+x_4)X^3 + (x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4)X^2 - (x_2x_3x_4+x_1x_3x_4+x_1x_2x_4+x_1x_2x_3)X + x_1x_2x_3x_4 = 0$$

这是可以用现成的python包解决的，网站上有很多网站，其代码实现了4次方程解决方案。 问题是系数很大，所以精度很可能会受到影响。 我使用了第一个在Google搜索http://adorio-research.org/wordpress/?p=4499(可能打不开)上弹出的东西，并用gmpy2:gmpy2 is a C-coded Python extension module that supports multiple-precision arithmetic.等价物代替了数学运算。

#这个代码不重要看一下懂就行其实就是求固定的参数，p q r s

```
import math
```

```
import gmpy2

gmpy2.get_context().precision=5000



from polycubicroot import *
def Carpenter(p, q,r, s):
 p = gmpy2.mpfr(p)
 q = gmpy2.mpfr(q)
 r = gmpy2.mpfr(r)
 s = gmpy2.mpfr(s)
 """
 Solves for all roots of the quartic polynomial P(x) = x^4 + px^3 + qx^2 + rx + s.
 """
 #print "@@@ inside Carpenter", p,q,r,s
 pby4 = p/4.0
 C = ((6 * pby4) - 3*p)*pby4 + q
 D = (((-4*pby4) + 3*p)*pby4 - 2*q)*pby4 + r
 E = (((pby4 - p)* pby4 + q)*pby4 - r)*pby4 + s
 #print "C, D, E=",C, D, E
 root = None
 for zero in polyCubicRoots(2*C, (C**2 - 4*E), -D**2):
  #print "zero = ", zero
  if type(zero)== type(gmpy2.mpfr(1.0)) and zero > 0.0:
     root = zero
     #print "found a positive root."
     break
 if root == None:
  return None
 sqroot = gmpy2.sqrt(root)
 Q1 = -root/4.0 - C/2.0 - D/2.0 / sqroot
 Q2 = -root/4.0 - C/2.0 + D/2.0 / sqroot
 #print "Q1,Q2=", Q1, Q2
 sqy2 = sqroot/2.0
 if Q1 >= 0.0:
    sqQ1 = gmpy2.sqrt(Q1)
    z1 = sqy2 + sqQ1 -pby4
    z2 = sqy2 - sqQ1 -pby4
 else:
    sqQ1 = gmpy2.sqrt(-Q1)
    z1 = (sqy2-pby4,   sqQ1)
    z2 = (sqy2-pby4, - sqQ1)
 if Q2 >= 0.0:
    sqQ2 = gmpy2.sqrt(Q2)
    z3 = -sqy2 - sqQ2 -pby4
    z4 = -sqy2 + sqQ2 -pby4
 else:
    sqQ2 = gmpy2.sqrt(-Q2)
    z3 = (-sqy2-pby4, sqQ2)
    z4 = (-sqy2-pby4, -sqQ2)
 return (z1, z2,z3, z4)
if __name__=='__main__':
 p= 0b10001110000110111100100111111000100111
 q= 0b11101100101010001000010100111100000101001000101110111100111100011010001010101
 r= 0b10101111001010011000110111011010000001100110101000100010100011000100001110010100111101110110001100100
 s= 0b11000010011110000010000010000101010000011101100011101110110110011111000000001011001101111100111001111

 A=[]
 B=[]
 C=[]
```

```
C=[]
D=[s]

for i in range(0,3):
  A.append(p * (2**i))
  B.append(q * (2**i))
  C.append(r * (2**i))

for i in A:
  print("====================")
  for j in B:
    for k in C:
      for l in D:
        (x1, x2, x3, x4) = Carpenter(-i,j,-k,l)
        # since sometimes the solution will consist of complex numbers, we'll just disregard those with a
        try:
          #approximate value of x1
          aprox = int(x1.__floor__())
          for step in range(0,10000):
            #try to the right
            if s % (aprox + step) == 0:
              print(aprox + step)
            #and to the left
            if s % (aprox - step) == 0:
              print(aprox - step)
        except:
          pass
```

那个polycubicroot去下载文件，反正我是网上没找到,不过没关系，只是心理上没有验证pqrs的正确性。下图是验

```
====================
152587894837
152587893565
152587892895
152587895947
====================


>>> s
542101138623964531063804924003409641945886325L
>>> s - 152587894837 * 152587893565 * 152587892895 * 152587895947
0L
```

接着下面求解四个素因子，通过M和abcd并且验证是不是

```
>>> f1=2**1392+152587894837
>>> f2=2**1392+152587893565
>>> f3=2**1392+152587892895
>>> f4=2**1392 + 152587895947
>>> f1
108082147276398906822234149167480016132157014049560913761488880190018027488520386318253742675423286348
552334110023434741671427911613197684395221211646299519273129194692306445874938199068586137486874290442
314459278649345469626426790676801658394799404284116771456479272808343825651929906737811050557836671896
732124546721747709022607151231423494815945385193624295868730390462068156825588342737037490320356361648
437686599733
>>> f2
108082147276398906822234149167480016132157014049560913761488880190018027488520386318253742675423286348
552334110023434741671427911613197684395221211646299519273129194692306445874938199068586137486874290442
314459278649345469626426790676801658394799404284116771456479272808343825651929906737811050557836671896
732124546721747709022607151231423494815945385193624295868730390462068156825588342737037490320356361648
437686598461
>>> f3
108082147276398906822234149167480016132157014049560913761488880190018027488520386318253742675423286348
552334110023434741671427911613197684395221211646299519273129194692306445874938199068586137486874290442
314459278649345469626426790676801658394799404284116771456479272808343825651929906737811050557836671896
732124546721747709022607151231423494815945385193624295868730390462068156825588342737037490320356361648
437686597791
>>> f4
108082147276398906822234149167480016132157014049560913761488880190018027488520386318253742675423286348
552334110023434741671427911613197684395221211646299519273129194692306445874938199068586137486874290442
314459278649345469626426790676801658394799404284116771456479272808343825651929906737811050557836671896
732124546721747709022607151231423494815945385193624295868730390462068156825588342737037490320356361648
437686600843
>>>
```



```
>>> N - (2**1392 + 152587894837) * (2**1392 + 152587893565) * (2**1392 + 152587892895) * (2**1392 + 1525878
0L
>>> N % (2**1392 + 152587894837)
0L
>>> N %  (2**1392 + 152587893565)
0L
>>> N % (2**1392 + 152587892895)
0L
>>> N % (2**1392 + 152587895947)
0L
```

2. 本题中模数同其他密文数比较相对较大，这就提示我们接下来该怎么做，推断可能@1公钥短，明文也非常短@2公钥是相当最大的（标准65537）但是大小不是N

提示：密码位长度是N的大约一半。那就应该是这种情况

3. 因此这里有根据代码pqrs都是已知求出的，在加上题目的N可以求出4个素数因子a,b,c,d（代码中用f1,f2,f3,f4），所有N'模数组合可能是ab,ac,ad,bc,bd,cd六个。

六个密文,但是有六个模数N'（下面N=136...没有用到，它求出f1f2f3f4有关，它是伪装N就像题目说的加密的N'）

```
#这个代码很重要


f1 = 10808214727639890682223414916748001613215701404956091376148888019001802748852038631825374267542328634 8
f2 = 10808214727639890682223414916748001613215701404956091376148888019001802748852038631825374267542328634 8
f3 = 10808214727639890682223414916748001613215701404956091376148888019001802748852038631825374267542328634 8
f4 = 10808214727639890682223414916748001613215701404956091376148888019001802748852038631825374267542328634 8

N  =  13646329614319089324860827044849343935068311549268087616805208405350434784823020872237520990607686905
enc_1 =41881606625990101743335482476854655320333512973875248158117739926714799774226187644019468931972354 49
enc_2 =60490496617411295613393039882501626177033597419701993567493780995353087995211192961634758135674586 44
enc_3 =92978028085642286788579277352242586101768312234554624821081539629532833111125019337796425408315585 23
enc_4 =71309598340999352801663006564895101340988742945076115929647921078230808484705078530037949090244864 74
enc_5 =10309778311674135836145162475142299890445328570998831868156889990227275287204954712195141311549279 04
enc_6 =89395359072612954384338865773864329682919261622779249129799413801895040003223101597878946190155500 96
import gmpy2
import binascii
e = 65537


N = f3 * f4
phi =  (f3-1)*(f4-1)
d = gmpy2.invert(e, phi)
print(binascii.unhexlify(hex(pow(enc_1, d, N))[2:]))

N = f2 * f4
phi =  (f2-1)*(f4-1)
d = gmpy2.invert(e, phi)
print(binascii.unhexlify(hex(pow(enc_2, d, N))[2:]))

N = f1 * f4
phi =  (f1-1)*(f4-1)
d = gmpy2.invert(e, phi)
print(binascii.unhexlify(hex(pow(enc_3, d, N))[2:]))
N = f1 * f3
phi =  (f1-1)*(f3-1)
d = gmpy2.invert(e, phi)
print(binascii.unhexlify(hex(pow(enc_4, d, N))[2:]))

N = f1 * f2
phi =  (f1-1)*(f2-1)
d = gmpy2.invert(e, phi)
print(binascii.unhexlify(hex(pow(enc_5, d, N))[2:]))

N = f2 * f3
phi =  (f2-1)*(f3-1)
d = gmpy2.invert(e, phi)
print(binascii.unhexlify(hex(pow(enc_6, d, N))[2:]))
```

```
E:\p\password>python rsaNencrypto.py
b'the first part of flag is: ASIS_13f'
b'the second part of flag is: 1c2154'
b'the third part of flag is: d59e27'
b'the forth part of flag is: b5405b'
b'the fifth part of flag is: c8591'
b'the last part of flag is: 49b970'

E:\p\password>_
```

最后实验吧提交的答案不对也是够了

那是因为en_c数字被换了但是N没有换，我的个去

```
f1 = 10808214727639890682223414916748001613215701404956091376148888019001802748852038631825374267542328634 8
f2 = 10808214727639890682223414916748001613215701404956091376148888019001802748852038631825374267542328634 8
f3 = 10808214727639890682223414916748001613215701404956091376148888019001802748852038631825374267542328634 8
f4 = 10808214727639890682223414916748001613215701404956091376148888019001802748852038631825374267542328634 8

N   =  13646329614319089324860827044849343935068311549268087616805208405350434784823020872237520990607686 90

enc_1 =  71068983348659559627046632301857664459518899784405276703796417143706812858599498699491878323590825

enc_2 =  60490496617411295613393039882501626177033597419701993567493780995353087995211192961634758135674586

enc_3 =  92978028085642286788579277352242586101768312234554624821081539629532833111125019337796425408315585

enc_4 =  71309598340999352801663006564895101340988742945076115929647921078230808484705078530037949090244864

enc_5 =  10309778311674135836145162475142299890445328570998831868156889990227275287204954712195141311549279

enc_6 =  89395359072612954384338865773864329682919261622779249129799413801895040003223101597878946190155500
import gmpy2
import binascii
e = 65537


N = f3 * f4
phi =  (f3-1)*(f4-1)
d = gmpy2.invert(e, phi)
print(binascii.unhexlify(hex(pow(enc_1, d, N))[2:]))

N = f2 * f4
phi =  (f2-1)*(f4-1)
d = gmpy2.invert(e, phi)
print(binascii.unhexlify(hex(pow(enc_2, d, N))[2:]))

N = f1 * f4
phi =  (f1-1)*(f4-1)
d = gmpy2.invert(e, phi)
print(binascii.unhexlify(hex(pow(enc_3, d, N))[2:]))
N = f1 * f3
phi =  (f1-1)*(f3-1)
d = gmpy2.invert(e, phi)
print(binascii.unhexlify(hex(pow(enc_4, d, N))[2:]))

N = f1 * f2
phi =  (f1-1)*(f2-1)
d = gmpy2.invert(e, phi)
print(binascii.unhexlify(hex(pow(enc_5, d, N))[2:]))

N = f2 * f3
phi =  (f2-1)*(f3-1)
d = gmpy2.invert(e, phi)
print(binascii.unhexlify(hex(pow(enc_6, d, N))[2:]))
```

最后稳稳得到flag

```
E:\p>python test.py
b'CTF_13f'
b'the second part of flag is: 1c2154'
b'the third part of flag is: d59e27'
b'the forth part of flag is: b5405b'
b'the fifth part of flag is: c8591'
b'the last part of flag is: 49b970'
```