

CTF中的RSA套路

👤 Von's Blog 📅 2019-08-08 ✓ 20454 words 👁 & views

前言

RSA是在CTF中经常出现的一类题目。~~一般难度不高，并且有一定的套路。~~(10.1补:我错了，我不配!我不配密码学)在此我写篇文章进行总结。本文不过多赘述RSA的加解密，仅从做题角度提供方法。虽然说不赘述加解密，但是我们还是需要清楚在RSA里面的几个基本参数。

N: 大整数N，我们称之为模数 (modulus)
p 和 q : 大整数N的两个因子 (factor)
e 和 d: 互为模反数的两个指数 (exponent)
c 和 m: 分别是密文和明文

而{N,e}称为公钥,{N,d}称为私钥。总的来说,明文m(一般为flag)就像是一个锁,而私钥就是打开这个锁的钥匙。我们要做的就是根据公钥来生成这把钥匙来打开锁。而私钥中的N又是可以从公钥中获得的,所以关键就是在d的获取,d的值和p,q,e有关。p,q又是N的两个因子,所以RSA题目关键便是对N的分解,分解出N的两个因子题目便解决了。这便是RSA题目的思路。

具体类型

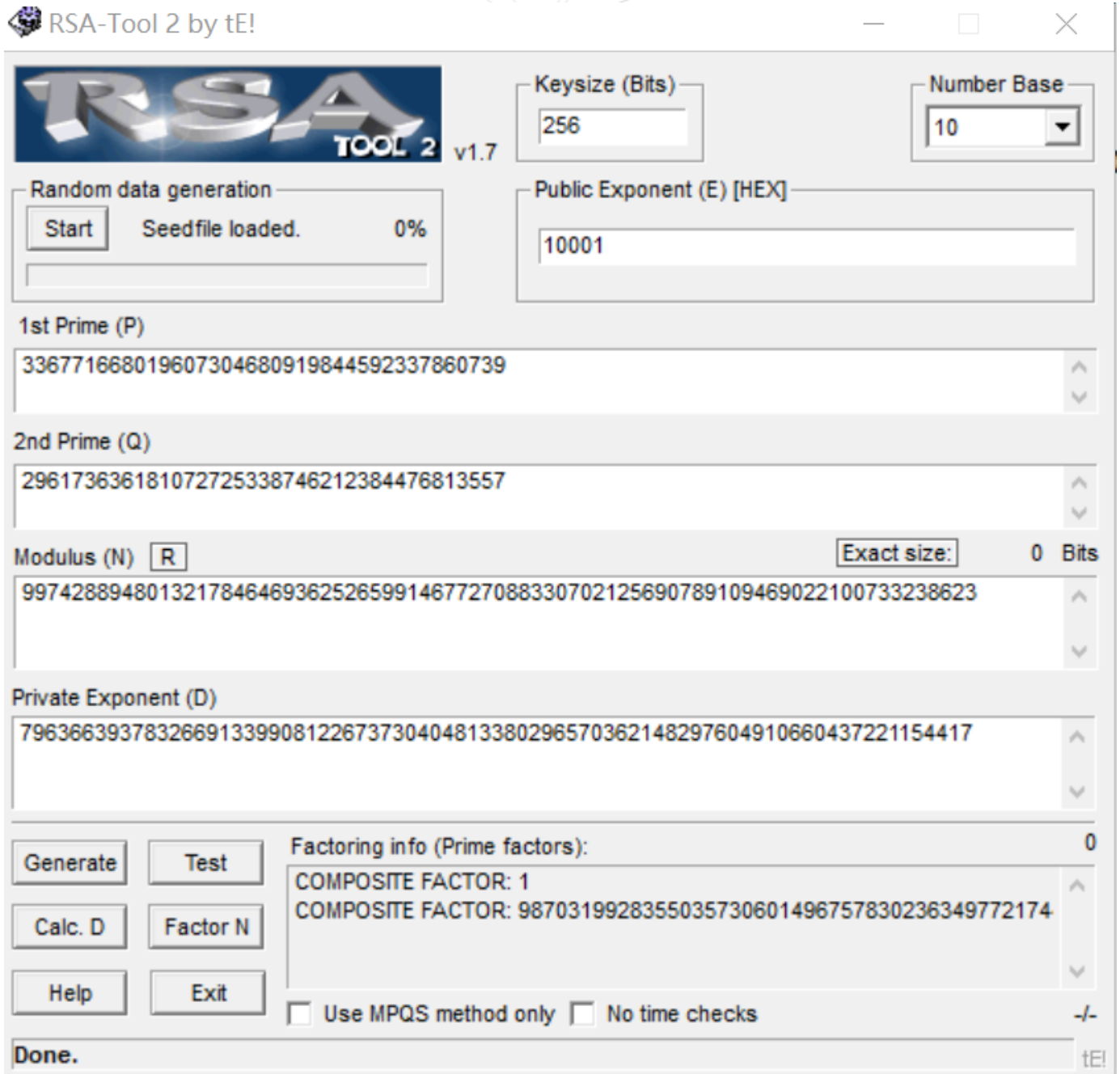
已知p,q,e, 获取d

这种题目一般不难，是RSA里面的入门题目。通常可以使用python脚本解题。



```
import gmpy2
p =gmpy2.mpz(336771668019607304680919844592337860739)
q =gmpy2.mpz(296173636181072725338746212384476813557)
e =gmpy2.mpz(65537)
phi_n= (p - 1) * (q - 1)
d = gmpy2.invert(e, phi_n)
print("d is:")
print (d)
```

也可以通过RSA-Tool解出d.



RSA-Tool 2 by tE!

Keysize (Bits): 256 **Number Base:** 10

Random data generation: Start Seedfile loaded. 0%

Public Exponent (E) [HEX]: 10001

1st Prime (P): 336771668019607304680919844592337860739

2nd Prime (Q): 296173636181072725338746212384476813557

Modulus (N) [R]: 99742889480132178464693625265991467727088330702125690789109469022100733238623 **Exact size:** 0 Bits

Private Exponent (D): 79636639378326691339908122673730404813380296570362148297604910660437221154417

Buttons: Generate, Test, Calc. D, Factor N, Help, Exit

Factoring info (Prime factors): 0
 COMPOSITE FACTOR: 1
 COMPOSITE FACTOR: 9870319928355035730601496757830236349772174

☐ Use MPQS method only ☐ No time checks

Done. tE!

已知e,d,N,求p,q

python代码如下:



```

# coding=utf-8 import random
import libnum
d = 79636639378326691339908122673730404813380296570362148297604910660437221154417
e = 65537
n = 99742889480132178464693625265991467727088330702125690789109469022100733238623
k = e * d - 1
r = k
t = 0
while True:
    r = r / 2
    t += 1
    if r % 2 == 1:
        break
success = False
for i in range(1, 101):
    g = random.randint(0, n)
    y = pow(g, r, n)
    if y == 1 or y == n - 1:
        continue
    for j in range(1, t):
        x = pow(y, 2, n)
        if x == 1:
            success = True
            break
        elif x == n - 1:
            continue
        else:
            y = x
    if success:
        break
    else:
        continue
if success:
    p = libnum.gcd(y - 1, n)
    q = n / p
    print 'P: ' + '%s' % p
    print 'Q: ' + '%s' % q
else:
    print 'Cannot compute P and Q'

```

已知N,e,c, 求m

这种题目要先分解出p,q。之后的python代码如下:



```
#!/usr/bin/env python # -*- coding: utf-8 -*- import gmpy2
p = 336771668019607304680919844592337860739
q = 296173636181072725338746212384476813557
e = 65537
c = 55907434463693004339309251502084272273011794908408891123020287672115136392494
n = p * q
fn = (p - 1) * (q - 1)
d = gmpy2.invert(e, fn)
h = hex(gmpy2.powmod(c, d, n))[2:]
if len(h) % 2 == 1:
    h = '0' + h
s = h.decode('hex')
print s
```

给出公钥文件和密文文件获取明文

出题人会给你一个公钥文件（通常是以.pem或.pub结尾的文件）和密文（通常叫做flag.enc之类的），你需要分析公钥，提取出（N，e），通过各种攻击手段恢复私钥，然后去解密密文得到flag。EG:一般先用openssl提取公钥文件中的N和e。

```
root@kali:~/桌面/RSA# openssl rsa -pubin -text -modulus -in public.pem
RSA Public-Key: (256 bit)
Modulus:
    00:dc:84:79:8f:78:6d:6d:ab:33:14:46:3e:2c:5f:
    27:cd:0d:c4:8a:0f:97:13:da:fc:f9:18:02:eb:bc:
    b7:1d:5f
Exponent: 65537 (0x10001)
Modulus=DC84798F786D6DAB3314463E2C5F27CD0DC48A0F9713DAFCF91802EBBCB71D5F
writing RSA key
-----BEGIN PUBLIC KEY-----
MDwwDQYJKoZIhvcNAQEBBQADKwAwKAIhANYEeY94bW2rMxRGPixfJ80NxIoPlxPa
/PKYAuu8tx1fAgMBAAE=
-----END PUBLIC KEY-----
```

公钥: 65537 (0x10001)

模数: DC84798F786D6DAB3314463E2C5F27CD0DC48A0F9713DAFCF91802EBBCB71D5F

转化为十进制:

99742889480132178464693625265991467727088330702125690789109469022100733238623

分解N得到p:336771668019607304680919844592337860739

q:296173636181072725338746212384476813557

写个python脚本解出flag



```

import gmpy2
p = 336771668019607304680919844592337860739
q = 296173636181072725338746212384476813557
e = 65537
f = int(open('flag.enc', 'rb').read().encode('hex'), 16)
print f
n = p * q
fn = (p - 1) * (q - 1)
d = gmpy2.invert(e, fn)
h = hex(gmpy2.powmod(f, d, n))[2:]
if len(h) % 2 == 1:
    h = '0' + h
s = h.decode('hex')
print s

```

分解N

我们在上面的类型题目中经常提到分解N。但只是一笔概括，下面具体讲讲怎么来分解N。

尝试在线网站分解N：

<http://factordb.com/> (<http://factordb.com/>)

Search	Sequences	Report results	Factor tables	Status	Downloads	Login
<input type="text" value="86804467865189181998675682302645596768517985924006311724377177674474176386743"/> <input type="button" value="Factorize!"/> (?)						
Result:						
status (?)	digits	number				
FF	77 (show)	8680446786...43 <77> = 293086410338424676391341741631987307899 <39> · 296173636181072725338746212384476813557 <39>				
More information						

Yafu分解N：

在RSA中，当p、q的取值差异过大或过于相近的时候，使用yafu可以快速的把n值分解出p、q值，原理是使用Fermat方法与Pollard rho方法等。



```

fac: factoring 99742889480132178464693625265991467727088330702125690789109469022100733238623
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
fmt: 1000000 iterations
rho: x^2 + 3, starting 1000 iterations on C77
rho: x^2 + 2, starting 1000 iterations on C77
rho: x^2 + 1, starting 1000 iterations on C77
oml: starting B1 = 150K, B2 = gmp-ecm default on C77
ecm: 30/30 curves on C77, B1=2K, B2=gmp-ecm default
ecm: 74/74 curves on C77, B1=11K, B2=gmp-ecm default
ecm: 149/149 curves on C77, B1=50K, B2=gmp-ecm default, ETA: 0 sec

starting SIQS on c77: 99742889480132178464693625265991467727088330702125690789109469022100733238623

==== sieving in progress (1 thread): 36224 relations needed ====
==== Press ctrl-c to abort and save state ====
36228 rels found: 18589 full + 17639 from 189109 partial, (1789.47 rels/sec)

SIQS elapsed time = 117.6145 seconds.
Total factoring time = 138.0658 seconds

***factors found***

P39 = 296173636181072725338746212384476813557
P39 = 336771668019607304680919844592337860739

```

利用公约数分解N:

识别此类题目，通常会发现题目给了多个n，均不相同，并且都是2048bit，4096bit级别，无法正面硬杠，并且明文都没什么联系，e也一般取65537。可以直接gcd(n1,n2)求出一个因数。代码如下:

```

import gmpy2
n1 = 905101396540408448287008786482145553515900869604295302196563108909579534883095438312732385
n2 = 132259483961796038160620464187172147926685124136250915699975243642439959919610188941500592
p = gmpy2.gcd(n1, n2)
print 'gcd(n1, n2):\n', p
q1 = n1 / p
q2 = n2 / p
print 'q1 is:\n', q1
print 'q2 is:\n', q2

```

低加密指数攻击:

在RSA中e也称为加密指数。由于e是可以随意选取的，选取小一点的e可以缩短加密时间（比如3），但是选取不当的话，就会造成安全问题。（题目特征:e=3）
这种题目又具体分以下两种情况:

明文m极小(m的三次方小于N):

此时利用代码如下:



```
# -*- coding: cp936 -*- import gmpy2
e = 3
# 读入 n, 密文 n= 22885480907469109159947272333565375109310485067211461543881386718201442106967
c= 15685364647213619014219110070569189770745535885901269792039052046431067708991036961644224236
print 'n=', n
print 'c=', c
print '[+]Detecting m...'
result = gmpy2.iroot(c, 3)
print ' [-]The c has cubic root?', result[1]
if result[1]: print ' [-]The m is:', '{:x}'.format(result[0]).decode('hex')
print '[!]All Done!'
```

m的3次方比N大，但不够大：

此时利用代码如下：

```
# -*- coding: cp936 -*- import gmpy2, time
e = 3
# 读入 n, 密文 n = 1149769157472433877921577084641207350189713362139354389530747482761982827619
c = 5828813410620741112500628876643872258919868379601617907887884191584237969605489971465692568
i = 239000000 # i 应该是未知的。这里缩短一下距离，防止跑得太久 print 'n=', n
print 'c=', c
print '[!]Done!\n'
print '[+]Detecting m...'
s = time.clock()
while 1:
    m, b = gmpy2.iroot(c + i * n, 3)
    if b:
        print ' [-]m is: ' + '{:x}'.format(int(m)).decode('hex')
        break
    #print ' [-]i = %d\r' % i, i += 1
print '[!]Timer:', round(time.clock() - s, 2), 's'
```

低加密指数广播攻击

如果选取的加密指数较低，并且使用了相同的加密指数给一个接受者的群发送相同的信息，那么可以进行广播攻击得到明文。

这个识别起来比较简单，一般来说都是给了三组加密的参数和明密文，其中题目很明确地能告诉你这三组的明文都是一样的，并且e都取了一个较小的数字。利用代码如下：



```

# -*- coding: cp936 -*- import gmpy2
import time
def CRT(items):
    N = reduce(lambda x, y: x * y, (i[1] for i in items))
    result = 0
    for a, n in items:
        m = N / n
        d, r, s = gmpy2.gcdext(n, m)
        if d != 1: raise Exception("Input not pairwise co-prime")
        result += a * s * m
    return result % N, N
# 读入 e, n, c e = 9
n = [142782424368849674771976671955176187834932417027468006479038058385550042422280158726561712
c = [850338684187843085736737099607007773503144264276776273196973468111237423423590721702204288
print '[+]Detecting m...'
data = zip(c, n)
x, n = CRT(data)
realnum = gmpy2.iroot(gmpy2.mpz(x), e)[0].digits()
print ' [-]m is: ' + '{:x}'.format(int(realnum)).decode('hex')
print '[!]All Done!'

```

低解密指数攻击

与低加密指数相同，低解密指数可以加快解密的过程，但是也带来了安全问题。一种基于连分数(一个数论当中的问题)的特殊攻击类型就可以危害 RSA 的安全。此时需要满足： $q < p < 2q$ 。如果满足上述条件，通过Wiener Attack可以在多项式时间中分解N。识别特征：E特别大。利用代码如下：




```

# -*- coding: cp936 -*- import gmpy2
import time
# 展开为连分数 def continuedFra(x, y):
    cF = []
    while y:
        cF += [x / y]
        x, y = y, x % y
    return cF
def Simplify(ctnf):
    numerator = 0
    denominator = 1
    for x in ctnf[::-1]:
        numerator, denominator = denominator, x * denominator + numerator
    return (numerator, denominator)
# 连分数化简 def calculateFrac(x, y):
    cF = continuedFra(x, y)
    cF = map(Simplify, (cF[0:i] for i in xrange(1, len(cF))))
    return cF
# 解韦达定理 def solve_pq(a, b, c):
    par = gmpy2.isqrt(b * b - 4 * a * c)
    return (-b + par) / (2 * a), (-b - par) / (2 * a)
def wienerAttack(e, n):
    for (d, k) in calculateFrac(e, n):
        if k == 0: continue
        if (e * d - 1) % k != 0: continue
        phi = (e * d - 1) / k
        p, q = solve_pq(1, n - phi + 1, n)
        if p * q == n:
            return abs(int(p)), abs(int(q))
    print 'not find!'
time.clock()
n = 1223860506325229217061311060769277932662809074575195569226664917788295923182258068254827980
e = 1185055248150302025739280842474351085176354818493653618031770715584195978815186297644595781
c = 9472193174575536616954091686751964873836697237500198884451530469300324470671555310791335185
p, q = wienerAttack(e, n)
print '[+]Found!'
print ' [-]p =', p
print ' [-]q =', q
print ' [-]n =', p*q
d = gmpy2.invert(e, (p-1)*(q-1))
print ' [-]d =', d
print ' [-]m is:' + '{:x}'.format(pow(c, d, n)).decode('hex')
print '\n[!]Timer:', round(time.clock(), 2), 's'
print '[!]All Done!'

```

共模攻击

如果在RSA的使用中使用了相同的模N对相同的明文m进行了加密，那么就可以在不分解n的情况下还原出明文m的值。

识别特征：若干次加密，e不同，N相同，m相同。

利用代码如下：



```

# -*- coding: cp936 -*- import time
import gmpy2
n = 1580527220137894614568969002445101991692165756930488951625385483564668843115437409680488251
e = [665213, 368273]
c = [166986176418882486646949801353321255317926925167880886827228320613931176095087652844732362
print '[+]Detecting m...'
time.clock()
c1 = c[0]
c2 = c[1]
e1 = e[0]
e2 = e[1]
s = gmpy2.gcdext(e1, e2)
s1 = s[1]
s2 = s[2]
# 求模反元素 if s1 < 0:
    s1 = -s1
    c1 = gmpy2.invert(c1, n)
elif s2 < 0:
    s2 = -s2
    c2 = gmpy2.invert(c2, n)
m = pow(c1, s1, n) * pow(c2, s2, n) % n
print '[-]m is:' + '{:x}'.format(int(m)).decode('hex')
print '\n[!]Timer:', round(time.clock(),2), 's'
print '[!]All Done!'

```

Rabin攻击

当 $e=2$ 时可以采取Rabin攻击。

利用代码如下：



```
#!/usr/bin/python # coding=utf-8 # 适合e=2 import gmpy
import string
from Crypto.PublicKey import RSA
# 读取公钥参数 with open('./tmp/pubkey.pem', 'r') as f:
    key = RSA.importKey(f)
    N = key.n
    e = key.e
p = 275127860351348928173285174381581152299
q = 319576316814478949870590164193048041239
with open('./tmp/flag.enc', 'r') as f:
    cipher = f.read().encode('hex')
    cipher = string.atoi(cipher, base=16)
    # print cipher # 计算yp和yq yp = gmpy.invert(p,q)
yq = gmpy.invert(q,p)
# 计算mp和mq mp = pow(cipher, (p + 1) / 4, p)
mq = pow(cipher, (q + 1) / 4, q)
# 计算a,b,c,d a = (yp * p * mq + yq * q * mp) % N
b = N - int(a)
c = (yp * p * mq - yq * q * mp) % N
d = N - int(c)
for i in (a,b,c,d):
    s = '%x' % i
    if len(s) % 2 != 0:
        s = '0' + s
    print s.decode('hex')
```

e=1

当e=1时，攻击代码可以转化为：

```
#!/usr/bin/env python3 #coding:utf-8 import binascii
import gmpy2
N_hex=0x180be86dc898a3c3a710e52b31de460f8f350610bf63e6b2203c08fddad44601d96eb454a34dab7684589bc
e_hex=0x1
c_hex=0x4963654354467b66616c6c735f61706172745f736f5f656173696c795f616e645f7265617373656d626c656
c_hex = gmpy2.mpz(c_hex)
N_hex = gmpy2.mpz(N_hex)
i = 0
while i<10:
    m_hex = hex(c_hex + gmpy2.mpz(hex(i))*N_hex)
    print(m_hex[2:])
    try:
        print(binascii.a2b_hex(m_hex[2:]).decode("utf8"))
    except binascii.Error as e:
        print("位数非偶数，跳过...")
    i += 1
```



以上便是常见的CTF种分解N的办法。在实际应用中，我总结出了以下的判断顺序：

1.先网站在线查询

2.使用Yafu

3.如果所给为多个N，使用公因数攻击

4.当E为1，2或特别大时，使用Rabin攻击或低解密指数攻击

5.当E为3时，如果只给出一组明密文。使用低加密指数攻击，如果给出多组明密文，使用低加密指数广播攻击

6.如果所给为多次加密，使用同个N，使用共模攻击

超级大杀器

上面有一堆让人头大的算法，比如分解一个大整数可能就有十来种算法，当然不是每个算法都能成功分解我们题目给的大整数的，如果我们挨个尝试，不仅浪费精力还浪费时间，等你找到正确的算法，已经与一血，二血，三血无缘了。所以我们需要一个自动化的工具。

<https://github.com/3summer/CTF-RSA-tool> (<https://github.com/3summer/CTF-RSA-tool>)

该项目便是一个超级大杀器，它能根据题目给的参数类型，自动判断应该采用哪种攻击方法，并尝试得到私钥或者明文，从而帮助CTFer快速拿到flag或解决其中的RSA考点。EG:

```
root@kali: ~/桌面/CTF-RSA-tool-master
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
root@kali:~/桌面/CTF-RSA-tool-master# python solve.py --verbose -k pubkey.pem -
-decrypt flag.enc
DEBUG: factor N: try past ctf primes
DEBUG: factor N: try Gimmicky Primes method
DEBUG: factor N: try Wiener's attack
DEBUG: Starting new HTTP connection (1): www.factordb.com
DEBUG: http://www.factordb.com:80 "GET /index.php?query=184958528317971463827449
839591526689799653355547671707738065461546785649065519 HTTP/1.1" 200 1094
DEBUG: http://www.factordb.com:80 "GET /index.php?id=1100000001281853538 HTTP/1.
1" 200 970
DEBUG: http://www.factordb.com:80 "GET /index.php?id=1100000001281853539 HTTP/1.
1" 200 969
DEBUG: d = 0x2d44796a60024c8c3aaaae053c235b6e0ad4f822f311b4535162b79904e02469L
INFO: 000
      nN=00makerCTF{3asy_RS4}
root@kali:~/桌面/CTF-RSA-tool-master#
```

