RSA攻击持续总结

原创

Lan_Magnolia 于 2020-04-13 00:43:56 发布 2047 收藏 13
文章标签： 加密解密 密码学 算法 安全

本文链接：https://blog.csdn.net/cwr1499640048/article/details/105472586

RSA攻击持续总结

RSA算法描述

1、变量涉及
明文：m
密文：c
模数：n
大质数：p，q
欧拉函数值：r
密钥：d，e
2、算法流程

- 随机生成两个大质数p，q
- n=p*q
- r=(p-1)*(q-1)
- 求e：1<e<r 且 gcd（e，r）=1
- 求d：1<d<r 且 e*d mod r=1
- 加密过程：m<n c=pow(m,e,n) 即m的e次方 mod n
- 解密过程：m=pow(c,d,n)

RSA攻击

1、已知n、e或是p、q、e求d

- 对模数n的分解
  yafu factor()函数
  分解n网站
- 得到p、q，利用脚本

```
import gmpy2
p = 473398607161
q = 4511491
e = 17
print gmpy2.invert(e, (p-1)*(q-1))
```

2、已知c、d、n求m
直接利用m=pow(c,d,n)

3、已知c、e、n求m
即需要先求出密钥d，使用1中的方法，再使用2中的方法

4、已知私钥文件、c求m
题目中给出了私钥文件private.pem和flag.enc
可在kali或是Ubuntu使用openssl直接进行解密

```
openssl rsautl -decrypt -in flag.enc -inkey private.pem
```

5、已知公钥文件、c求m
题目中给出了public.pem和密文flag.enc
使用`openssl rsa -pubin -text -modulus -in warmup -in pubkey.pem`
返回公钥信息，即可以得到n、e，分解n得到p、q
使用rsatool生成私钥文件: private.pem

```
shell python rsatool.py -o private.pem -e 65537 -p XXX -q XXX
```
即接下来用生成的私钥文件解密flag文件

```
shell openssl rsautl -decrypt -in flag.enc -inkey private.pem
```

6、共模攻击
所谓共模攻击指的是两个不同的e1，e2和一个n对一个明文m进行加密，得到两份密文c1，c2，此时可以在不分解n的情况下还原出明文m的值。
$c1 = m^{e1} \bmod n$
$c2 = m^{e2} \bmod n$

此时不需要分解 n，不需要求解私钥，如果两个加密指数互素，就可以通过共模攻击在两个密文和公钥被嗅探到的情况下还原出明文 m 的值。
过程如下，首先两个加密指数互质，则：
$(e_1, e_2) = 1$ 即存在 s_2，使得：

$$s_1 e_1 + s_2 e_2 = 1$$
$$\because c_1 \equiv m^{e_1} \ mod \, n, c_2 \equiv m^{e_2} \ mod \, n$$
代入化简可以得出：
$$c_1^{s_1} c_2^{s_2} \equiv m \, mod \, n$$

即 $m^{e1s1} \bmod n = c1^{s1}$，$m^{e2s2} \bmod n = c2^{s2}$
$m^{e1s1+e2s2} \bmod n = c1^{s1}c2^{s2}$
$c1^{s1} * c2^{s2} = m$

```
# -*- coding: cp936 -*-
import time
import gmpy2
n =
e = [665213, 368273]
c = [...L, ...L]
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!'
```

```
import gmpy2 as gp
import libnum
def exgcd(a, b):
    if b==0:
        return 1, 0, a
    x2, y2, r = exgcd(b, a%b)
    x1 = y2
    y1 = x2-(a//b)*y2
    return x1, y1, r

n=gp.mpz()
c1=gp.mpz()
e1=gp.mpz()
c2=gp.mpz()
e2=gp.mpz()

r1, r2, t = exgcd(e1, e2)
m = gp.powmod(c1, r1, n) * gp.powmod(c2, r2, n) % n
print m
print hex(m)[2:]
print libnum.n2s(m)
```

7、低加密指数攻击

题目中给出了pubkey.pem和密文flag.enc

使用openssl命令提取其中的n和e，得到的e十分小。e是加密者自定义的，一般会越大越安全，即这里是低加密。

有一种情况是当e = 3时的小明文攻击

当e = 3时，如果明文过小，即m的三次方还是小于n，可以导致 $c = m^e$，e = 3 则此时直接对密文开三次方即可

```
import gmpy2
e =
n =
c =

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!'
```

还有一种情况是明文m的三次方比n大，但是不是足够大，这时可以设k，有：

$c = m^e - kn$

爆破k，如果c + kn能开三次根式，那么可以直接得到明文

```
# -*- coding: cp936 -*-
import gmpy2, time
e = 3
# 读入 n, 密文
n =
c =
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'
```

通用脚本：

```
import gmpy2

def de(c, e, n):
    k = 0
    while True:
        mm = c + n*k
        result, flag = gmpy2.iroot(mm, e)
        if True == flag:
            return result
        k += 1
c =
e =
n =
m = de(c, e, n)
print m
```

8、低加密指数广播攻击

这里加密指数比较低，并且使用了相同的加密指数给一个接受群发消息，这时可以根据广播攻击的得到明文。

这里取e = 3

$c = m^e \bmod n$

$c = m^e \bmod n$

$c = m^e \bmod n$

运用中国剩余定理，当e = 3时

$c = m^3 \bmod n\ n\ n$

```
# -*- 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 =
n = [...L,...L] #n是一个数组因为加密的是群消息
c = [...L,...L]
data = zip(c, n)
x, n = CRT(data)
m = gmpy2.iroot(gmpy2.mpz(x), e)[0]
print m
print '[!]All Done!'
```

9、低解密指数攻击

与低加密指数攻击相比，这里的e过大，直接使用RSAwienerHacker.py，求出d，接下来再求出m。

这个脚本在github上，脚本链接

这里给出上次发现的别人写的代码

```
# -*- 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 =
e =
c =
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!'
```

10、n的公约数

题目中给出了两个n1，n2一个共同的密钥e，两个密文c1，c2

这里可以运用两个n的公因子q，再通过q和n1求出p1，已知p1和q，可以求得d，即接下来已知c1，d1，n1可以求得明文。

```
from Crypto.Util.number import getPrime,bytes_to_long,long_to_bytes

c1=
n1=
n2=
e=

def egcd(a,b):
    if a == 0:
        return (b,0,1)
    else:
        g,y,x=egcd(b%a,a)
    return (g,x-(b//a)*y,y)
def modinv(a,m):
    g,x,y=egcd(a,m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x%m
def gcd(a,b):
    while a != 0:
        a,b=b%a,a
    return b
q=gcd(n1,n2)
p1=n1/q

d=modinv(e,(p1-1)*(q-1))
dec1=pow(c1,d,n1)
print long_to_bytes(dec1)
```

11、n可以分解为多个素数

参考文章

```
def egcd(a,b):
    if a == 0:
        return (b,0,1)
    else:
        g,y,x=egcd(b%a,a)
    return (g,x-(b//a)*y,y)
def modinv(a,m):
    g,x,y=egcd(a,m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x%m
e=
p=
q=
r=
n=
d=modinv(e,(p-1)*(q-1)*(r-1))
print(d)
c=
destr=hex(pow(c,d,n))
destr=destr[2:-1]
print(destr)
```

12、dp泄露攻击

```
import gmpy2
import libnum

e =
n =
dp =
c =

for i in range(1,e):
    if (dp*e-1)%i == 0:
        if n%(((dp*e-1)/i)+1)==0:
            p=((dp*e-1)/i)+1
            q=n/(((dp*e-1)/i)+1)
            phi = (p-1)*(q-1)
            d = gmpy2.invert(e,phi)%phi

print libnum.n2s(pow(c,d,n))
```

13、dp，dq泄露攻击

$$m1 \equiv c^{dq} \bmod q$$
$$m2 \equiv c^{dp} \bmod p$$
$$m \equiv (((m2-m1)*p-1 \bmod q)p+m1) \bmod n$$

```
import gmpy2
import libnum

p =
q =
dp =
dq =
c =

InvQ = gmpy2.invert(q,p)
mp = pow(c,dp,p)
mq = pow(c,dq,q)
m = (((mp-mq)*InvQ) % p)*q+mq
print libnum.n2s(m)
```

14、已知n、r求p、q的算法

这里自己写了一个用二分法求二元一次方程组的代码

```
n=
r=
c1=n-r+1
print c1

l=c1/2
r=c1
p=(l+r)/2
y=p*(c1-p)

while l<r:
 p=(l+r)/2
 y=p*(c1-p)
        if y==n:
            print p
            break
        if y>n:
   print 'y>n'
   l=p
        else:
   print 'y<n'
   r=p
        print 'done'

q=c1-p

print q
```

15、已知n、e*d，求p，q算法

Algorithm

An RSA modulus N product of large distinct primes can be factored given (N, e, d) per:

1. Compute $f \leftarrow e\,d - 1$, and express f as $2^s\,t$ with t odd
2. Set $i \leftarrow s$ and $a \leftarrow 2$
3. Compute $b \leftarrow a^t \bmod N$, and if $b = 1$ then
   - set a to the next prime, and proceed at 3
4. If $i \neq 1$ then
   - compute $c \leftarrow b^2 \bmod N$, and if $c \neq 1$ then
     - set $b \leftarrow c$, decrease i, and proceed at 4
5. If $b = N - 1$ then
   - set a to the next prime, and proceed at 3
6. Compute and output $p \leftarrow \gcd(b - 1, N)$, and $q \leftarrow N/p$.

For standard RSA where N has 2 distinct factors, we have fully factored N. Otherwise, p or/and q won't be prime, just re-run the algorithm from step 2 replacing N by any still unfactored component, until all the prime factors of N have been pulled out.

2020MRCTF中的一道题目

```
def gen_q():
    p = Prime(1024)
    q = Prime(1024)
    assert (p < q)
    n = p * q
    print("Q_n = ", n)
    e = getRandomNBitInteger(53)
    F_n = (p - 1) * (q - 1)
    while gcd(e, F_n) != 1:
        e = getRandomNBitInteger(53)
    d = invert(e, F_n)
    print("Q_E_D = ", e * d)
    factor2 = 2021 * p - 2020 * q
    if factor2 < 0:
        factor2 = (-1) * factor2
    return sympy.nextprime(factor2)
```

参考一些大佬写的脚本

```
Q_n = 207142983381604497495453607436880188428772740545408520964594852839368023412713637661579761125250340043199380540349348808609569665850516844836625357806216733167748426147017264458706301091960166767251834128798704634322776299166691304940404037332955936553061041763679023524843675202629179431004676975405939257071621626166355335502627188087462545994562865784091878951710157969919101238045298255195192783889104831338133309025301604489729260960839902082432745485612382530027894749207307600011040480932956805930333278188212553008934234121922658144185461340155575792362194617803444691279876695651389303085251899448974217539475792362194617803444691279876695651389303085251899448974217539471007207922229813458611615685074281785540812771696289192925986874667257260233391895807558267175249361825951828633612277270333018303722110505829865349079433788509849907358382183253279830951353838317523342953346734839038932322519880529495048480206814859090290722115096853906798043283131037636820277321226632011267069973750105483164628658514228141923757222271397564684355502473185568857383410871187440614954007825377434970815806305575493281267578612370076828804844532619988098371750453882549810378930487368219105305036680682580260265867426844084457795549936840401911491393447716042842866284701228951665531068011963860031522828429893520 1
f, s, tem = Q_E_D-1, 0, 1
while f % 2 == 0:
    f = f // 2
    s += 1
i, a, t = s, 2, f
b = pow(a, t, Q_n)
while b == 1:
    a = sympy.nextprime(a)
    b = pow(a, t, Q_n)

while i != 1:
    c = pow(b, 2, Q_n)
    if c != 1:
        b = c
        i -= 1
    else:
        break
if b == Q_n-1:
    a = sympy.nextprime(a)
    b = pow(a, t, Q_n)
    while b == 1:
        a = sympy.nextprime(a)
        b = pow(a, t, Q_n)

p = gcd(b-1, Q_n)
q = Q_n//p
```

原文链接

```
#coding=utf-8
from random import randint
import gmpy2
def oddR(r):
    while r%2==0:
        r=r//2
    return r

def bits(b):
    k=[]
    while b:
        if b%2!=0:
            k.append(1)
        else:
            k.append(0)
        b>>=1
    k.reverse()
    return k

def quickmod(a,b,n):      #a^b mod n 快速幂模n运算
    f=1
    k=bits(b)
    for i in range(len(k)):
        f=(f*f)%n
        if k[i]:
            f=(f*a)%n
    return f

def gcd(m,n):
    while(n!=0):
        m,n=n,m%n
    return m

def func(e_d,N):
    k=e_d-1
    r=oddR(k)           #求出k=2^t*r中的r

    while True:
        b=randint(2,N-1)    #获取区间(2,N-1)的一个随机数
        a=quickmod(b,r,N)
        if a==1:
            continue
        y=gcd(a-1,N)
        if a>1 and y>1:
            q=N//y
            return q
        else:
            r=r*2

def deciphering(e_d,n):    、
    p=func(e_d,n)
    q=n//p
    phi=n-(p+q)+1
    if p*q==n:
        print p
        print q
    else:
        print"error"
```

```
        print error

n =
e_d=
deciphering(e_d,n)
```

原文链接

16、已知(e, n, d)分解N

证明涉及二次剩余一些知识

```
from gmpy2 import next_prime, gcd
def Factorize(n, e, d):
    g = 2
    while True:
        k = e * d - 1
        while not k & 1:
            k //= 2
            p = int(gcd(pow(g, k, n) - 1, n)) % n
            if p > 1:
                return (p, n // p)
        g = int(next_prime(g))
if __name__ == "__main__":
    n =
    e =
    d =
    print(Factorize(n, e, d))
```

17、coppersmith算法

以下内容是看的是一位博主的文章， 原文章

明文高位已知

例2019强网杯

```
[+]Generating challenge 1

[+]n=0xa1888c641a05aeb81b3d1686317a86f104791fe1d570a5b11209f45d09ea401d255a70744e7a2d39520e359c23a9f1209ee47f496
dbd279e62ee1648b3a277ced8825298274322e0a7a86deea282676310a73b6bb946fc924c34ac6c8784ff559bf9a004c03fb167ef54aaea9
0ce587f2f3074b40d7f632022ec8fb12e659953L
[+]e=3
[+]m=random.getrandbits(512)
[+]c=pow(m,e,n)=0x93145ece45f416a11e5e9475518f165365456183c361500c2f78aff263028c90f20b7d97205f54e21f3bcc8a556b45
7889fde3719d0a0f9c9646f3f0d0a1e4bee0f259f023168fe8cc0511848c1635022fcc20b6088084585e2f8055a9d1b1d6bdb228087900bf
7c6d42298f8e45c451562c816e2303990834c94e580bf0cbd1L
[+]((m>>72)<<72)=0x9e67d3a220a3dcf6fc4742052621f543b8c78d5d9813e69272e65ac676672446e5c88887e8bfdfc92ec87ec74c163
50e6b539e3bd910b000000000000000000L
```

代码：（需要使用sage）

```
e = 0x3
b = 0x9e67d3a220a3dcf6fc4742052621f543b8c78d5d9813e69272e65ac676672446e5c88887e8bfdfc92ec87ec74c16350e6b539e3bd9
10b000000000000000000L
n = 0xa1888c641a05aeb81b3d1686317a86f104791fe1d570a5b11209f45d09ea401d255a70744e7a2d39520e359c23a9f1209ee47f496d
bd279e62ee1648b3a277ced8825298274322e0a7a86deea282676310a73b6bb946fc924c34ac6c8784ff559bf9a004c03fb167ef54aaea90
ce587f2f3074b40d7f632022ec8fb12e659953L
c=0x93145ece45f416a11e5e9475518f165365456183c361500c2f78aff263028c90f20b7d97205f54e21f3bcc8a556b457889fde3719d0a
0f9c9646f3f0d0a1e4bee0f259f023168fe8cc0511848c1635022fcc20b6088084585e2f8055a9d1b1d6bdb228087900bf7c6d42298f8e45
c451562c816e2303990834c94e580bf0cbd1L
kbits=72
PR.<x> = PolynomialRing(Zmod(n))
f = (x + b)^e-c
x0 = f.small_roots(X=2^kbits, beta=1)[0]
print "x: %s" %hex(int(x0))
```

p的高位已知，低位未知
例2019强网杯

```
[+]Generating challenge 2
[+]n=0x241ac918f708fff645d3d6e24315e5bb045c02e788c2b7f74b2b83484ce9c0285b6c54d99e2a601a386237d666db2805175e7cc86
a733a97aeaab63486133103e30c1dca09741819026bd3ea8d08746d1d38df63c025c1793bdc7e38d194a30b492aadf9e31a6c1240a65db49
e061b48f1f2ae949ac9e7e0992ed24f9c01578dL
[+]e=65537
[+]m=random.getrandbits(512)
[+]c=pow(m,e,n)=0x1922e7151c779d6bb554cba6a05858415e74739c36df0bcf169e49ef0e566a4353c51a306036970005f2321d1d104f
91a673f40944e830619ed683d8f84eaf26e7a93c4abe1dbd7ca3babf3f4959def0e3d87f7818d54633a790fc74e9fed3c5b5456c21e3f425
240f6217b0b14516cb59aa0ce74b83ca17d8cc4a0fbc829fb8L
[+]((p>>128)<<128)=0x2c1e75652df018588875c7ab60472abf26a234bc1bfc1b685888fb5ded29ab5b93f5105c1e9b46912368e626777
a87320000000000000000000000000000000L
```

代码：（需要使用sage）

```
n = 0x241ac918f708fff645d3d6e24315e5bb045c02e788c2b7f74b2b83484ce9c0285b6c54d99e2a601a386237d666db2805175e7cc86a
733a97aeaab63486133103e30c1dca09741819026bd3ea8d08746d1d38df63c025c1793bdc7e38d194a30b492aadf9e31a6c1240a65db49e
061b48f1f2ae949ac9e7e0992ed24f9c01578dL
p_fake = 0x2c1e75652df018588875c7ab60472abf26a234bc1bfc1b685888fb5ded29ab5b93f5105c1e9b46912368e626777a873200000
00000000000000000000000000L

pbits = 1024
kbits = 130
pbar = p_fake & (2^pbits-2^kbits)
print "upper %d bits (of %d bits) is given" % (pbits-kbits, pbits)

PR.<x> = PolynomialRing(Zmod(n))
f = x + pbar

x0 = f.small_roots(X=2^kbits, beta=0.4)[0]  # find root < 2^kbits with factor >= n^0.3
print hex(int(x0 + pbar))
```

已知低位的密钥d和N
Partial Key Exposure Attack(部分私钥暴露攻击)
2019强网杯

```
[+]Generating challenge 3
[+]n=0x51fb3416aa0d71a430157d7c9853602a758e15462e7c08827b04cd3220c427bbb8199ed4f5393dae43f013b68732a685defc17497
f0912c886fa780dfacdfbb1461197d95a92a7a74ade874127a61411e14a901382ed3fb9d62c040c0dbaa374b5a4df06481a26da3fca27142
9ff10a4fc973b1c82553e3c1dd4f2f37dc24b3bL
[+]e=3
[+]m=random.getrandbits(512)
[+]c=pow(m,e,n)=0x3d7e16fd8b0b1afdb4e12594c3d8590f1175800ef07bb275d7a8ad983d0d5d5fd5c6f81efa40f5d10c48bb200f805e
679d633ee584748e5feef003e0921dea736ba91eef72f3d591d3a54cd59fd36f61140fdd3fb2e2c028b684e50cbeae4a1f386f6ab35359d4
6a29996c0f7d9a4a189f1096150496746f064c3cc41cf111b0L
[+]d=invmod(e,(p-1)*(q-1))
[+]d&((1<<512)-1)=0x17c4b18f1290b6a0886eaa7bf426485a3994c5b71186fe84d5138e18de7e060db57f9580381a917fdfd171bfd159
825a7d1e2800e2774f5e4449d17e6723749bL
[-]long_to_bytes(m).encode('hex')=
d = 0x36a7780f1c08f66d7563a8fdbae2401c4e5eb8d97452b056fcadde216b2d6fd27abbbf38a37b7e742d4ab7cf04cc6f03e9fd64dbaa
060c85af51a55ea733fd2017c4b18f1290b6a0886eaa7bf426485a3994c5b71186fe84d5138e18de7e060db57f9580381a917fdfd171bfd1
59825a7d1e2800e2774f5e4449d17e6723749bL
```

代码：（需要使用sage）

```
def partial_p(p0, kbits, n):
    PR.<x> = PolynomialRing(Zmod(n))
    nbits = n.nbits()
    f = 2^kbits*x + p0
    f = f.monic()
    roots = f.small_roots(X=2^(nbits//2-kbits), beta=0.3)  # find root < 2^(nbits//2-kbits) with factor >= n^0.3
    if roots:
        x0 = roots[0]
        p = gcd(2^kbits*x0 + p0, n)
        return ZZ(p)


def find_p(d0, kbits, e, n):
    X = var('X')


    for k in xrange(1, e+1):
        results = solve_mod([e*d0*X - k*X*(n-X+1) + k*n == X], 2^kbits)
        for x in results:
            p0 = ZZ(x[0])
            p = partial_p(p0, kbits, n)
            if p:
                return p


if __name__ == '__main__':
    # n = 0x51fb3416aa0d71a430157d7c9853602a758e15462e7c08827b04cd3220c427bbb8199ed4f5393dae43f013b68732a685defc
17497f0912c886fa780dfacdfbb1461197d95a92a7a74ade874127a61411e14a901382ed3fb9d62c040c0dbaa374b5a4df06481a26da3fca
271429ff10a4fc973b1c82553e3c1dd4f2f37dc24b3bL
    e = 3
    # d = 0x17c4b18f1290b6a0886eaa7bf426485a3994c5b71186fe84d5138e18de7e060db57f9580381a917fdfd171bfd159825a7d1e
2800e2774f5e4449d17e6723749bL

    n = 5756920104899347505234918724475216975416515457578276000385177781376704895305183928852813712167099988430984981576599961634630379247151863908069716676764495704658238578572110237028880603818795603250576153278971600952213145021701062933800024193603618520503881439120584823236400634921383631780690303251519440773 9
    nbits = n.nbits()
    kbits = floor(nbits*0.5)
    print "kbits : ", kbits
    d0 = 1244848677959253796774387650148978357579294769878147704641867595620534030329181934099194560059806799908134954814673426128260540575360296026444649631806619
    print "lower %d bits (of %d bits) is given" % (kbits, nbits)

    p = find_p(d0, kbits, e, n)
    print "found p: %d" % p
    q = n//p
    # print d
    print inverse_mod(e, (p-1))
```

18、Wiener Attack

利用了连分数的最佳逼近性

当d< 1/3 n $^{1/4}$时

```python
def rational_to_quotients(x, y):  # calculate the series of continued fraction
    a = x // y
    quotients = [a]
    while a * y != x:
        x, y = y, x - a * y
        a = x // y
        quotients.append(a)
    return quotients

def convergents_from_quotients(quotients):  # calculate the convergent series of continued fraction
    convergents = [(quotients[0], 1)]
    for i in range(2, len(quotients) + 1):
        quotients_partion = quotients[0:i]
        denom = quotients_partion[-1]  # 分母
        num = 1
        for _ in range(-2, -len(quotients_partion), -1):
            num, denom = denom, quotients_partion[_] * denom + num
        num += denom * quotients_partion[0]
        convergents.append((num, denom))
    return convergents

def WienerAttack(e, n):
    quotients = rational_to_quotients(e, n)
    convergents = convergents_from_quotients(quotients)
    for (k, d) in convergents:
        if k and not (e * d - 1) % k:
            phi = (e * d - 1) // k
            # check if (x^2 - coef * x + n = 0) has integer roots
            coef = n - phi + 1
            delta = coef * coef - 4 * n
            if delta > 0 and iroot(delta, 2)[1] == True:
                print('d = ' + str(d))
                return d

cipher = encrypt(n, e, flag)
d = WienerAttack(e, n)
decrypt(n, d, cipher)
```

19、Boneh and Durfee attack

当 d 较小时，满足d < N^{0.292} 时，我们可以利用该攻击，比 Wiener's Attack 要强一些。

例题

2019强网杯

```
[+]Generating challenge 6
[+]n=0xbadd260d14ea665b62e7d2e634f20a6382ac369cd44017305b69cf3a2694667ee651acded7085e0757d169b090f29f3f86fec2557
46674ffa8a6a3e1c9e1861003eb39f82cf74d84cc18e345f60865f998b33fc182a1a4ffa71f5ae48a1b5cb4c5f154b0997dc9b001e441815
ce59c6c825f064fdca678858758dc2cebbc4d27L
[+]d=random.getrandbits(1024*0.270)
[+]e=invmod(d,phin)
[+]hex(e)=0x11722b54dd6f3ad9ce81da6f6ecb0acaf2cbc3885841d08b32abc0672d1a7293f9856db8f9407dc05f6f373a2d9246752a7c
c7b1b6923f1827adfaeefc811e6e5989cce9f00897cfc1fc57987cce4862b5343bc8e91ddf2bd9e23aea9316a69f28f407cfe324d546a7dd
e13eb0bd052f694aefe8ec0f5298800277dbab4a33bbL
[+]m=random.getrandbits(512)
[+]c=pow(m,e,n)=0xe3505f41ec936cf6bd8ae344bfec85746dc7d87a5943b3a7136482dd7b980f68f52c887585d1c7ca099310c4da2f70
d4d5345d3641428797030177da6cc0d41e7b28d0abce694157c611697df8d0add3d900c00f778ac3428f341f47ecc4d868c6c5de0724b0c3
403296d84f26736aa66f7905d498fa1862ca59e97f8f866cL
[-]long_to_bytes(m).encode('hex')=
```

攻击脚本

20、e与r不互素时，除去公因数之后，有限域开根
题目中即gcd(e,r) != 1
例如题目中给出了如下的条件，其中tmp为公因数

```
n1=
e1=
c1=
n2=
e2=
c2=
assert pow(flag,e1,n1)==c1
assert pow(flag,e2,n2)==c2
assert gcd(e1,(p1-1)*(q1-1))==tmp
assert gcd(e2,(p2-1)*(q2-1))==tmp
```

例题和解题步骤参考这篇博客 链接
脚本

```
n1=
e1=
c1=
n2=
e2=
c2=
from libnum import *
import gmpy2
p=gcd(n1,n2)
q1=n1/p
q2=n2/p
assert(p*q1==n1)
assert(p*q2==n2)
f1=(p-1)*(q1-1)
f2=(p-1)*(q2-1)
tmp=14

e1=e1/tmp
e2=e2/tmp
bd1=invmod(e1,f1)
bd2=invmod(e2,f2)

m1=pow(c1,bd1,n1)
m2=pow(c2,bd2,n2)
m3=m1%p
m2=m2%q2
m1=m1%q1

m=solve_crt([m1,m2,m3], [q1,q2,p])
print m
n=q1*q2
f=(q1-1)*(q2-1)
m=m%n
d=invmod(7,f)
m=pow(m,d,n)
m = gmpy2.iroot(m, 2)[0]
flag = binascii.unhexlify(hex(m)[2:])
print flag
```

```
'''print('\n' + cut_off_line +
      'gcd(e, phi) != 1\n')
n, e, _, p, q = init_RSA(256, 5, coprime=False)
flag = 'flag{te3t_f1@g_666}'
print(bytes_to_long(flag.encode()))
cipher = encrypt(n, e, flag)'''
n =
e =
cipher =
phi = (p-1) * (q-1)
fac = GCD(e, phi)
new_e = e//fac
d = inverse(new_e, phi)
tmp_FLAG = pow(cipher, d, n)

if iroot(tmp_FLAG, fac)[1]:
    FLAG = long_to_bytes(int(iroot(tmp_FLAG, fac)[0]))
    print(FLAG)
else:
    print('root on GF')
    print('c1 = ' + str(tmp_FLAG % p))
    print('c2 = ' + str(tmp_FLAG % q))
    print('p = ' + str(p))
    print('q = ' + str(q))
    print('e = ' + str(fac))
```