

2021年9月绿城杯，CRYPTO的RSA-1

原创

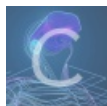
沐一·林  于 2021-09-30 12:20:30 发布  59  收藏

分类专栏: [密码学 CTF](#) 文章标签: [ctf](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: https://blog.csdn.net/xiao__1bai/article/details/120564257

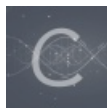
版权



[密码学](#) 同时被 2 个专栏收录 

51 篇文章 1 订阅

订阅专栏



[CTF](#)

167 篇文章 6 订阅

订阅专栏

2021年9月绿城杯，CRYPTO的RSA-1



打开文件，发现是很常规的 RSA 加密，就是明文进行了二次处理：

```
from Crypto.Util.number import *
import gmpy2
from flag import flag
assert flag[:5] == b'flag{'

m = bytes_to_long(flag)
p = getPrime(1024)
q = getPrime(1024)
n = p * q
print('n =', n)
e = 0x10001
M = 2021 * m * 1001 * p
c = pow(M, e, n)
print('c =', c)
```

这里的明文经过了简单的乘法进行二次处理

```
#n = 1736523115492634836447827687255849277591176060300239435372360346189840574023471500182011154860
#c = 69449671088154377354289412867841194031383197134557321559250559286465369625976729418058313121306
```

照例用工具 CTF-RSA-tool 工具，首先提取出来符合 CTF-RSA-tool 的格式的信息先：

```
N is 1736523115492634836447827687255849277591176060300239435372360346189840574023471500182011154860091490761700
3806652492391686710256274156677887101997175692277729648456087534987616743724646598234466094779540729413583826355
1452779804790401570754536942505723166383481215712187597695337387215068111758669908519728384663075942262938369341
1665968521577564328546589531775589275447333203423449579593618361056957101640053536276269951768678160230204504853
2131426035260878979892169441059467623523060569285570577199236309888155833013721997933960457784653262076135561769
838704166810384309655788983073376941843467117256002645962737847
e is 65537
c is 6944967108815437735428941286784119403138319713455732155925055928646536962597672941805831312130689338014913
4520812964002728627104472072650997504016578281658360131228486568391008547199651886800973754911932491277255996603
8374682703180306602649798929885642021625020603506818096379745479215119107143364594624591491673263700711708519944
2894495667455544517483404006536607121480678688000420422281380539368519807162175099763891988648117937777951069899
9752601900189958349045414475627183074339065920212266668856388770203040056144507630813370828386084147561622538256
97420493509914578546951634127502393647068722995363753321912676
```

运行脚本看一下回显，明文必然是错误的，因为 M 经过了二次简单乘法处理，所以我们要在脚本中找到对应的地方修改一下才行：

我们修改一下 CTF-RSA-tool 打印出 p、q

```
.
; # 密文与模数不互素
; def comfact_cn(N, c):
7   log.debug('factor N: try Common factor between ciphertext and modulus attack')
;   # Try an attack where the public key has a common factor with the ciphertext - sourcekris
;   if c:
)     commonfactor = libnum.gcd(N, c)
L     if commonfactor > 1:
?       q = commonfactor
;       p = N / q
;       print('p=', p)
;       print('q=', q)
;       return p, q
; |
; }
```

打印p、q然后常规RSA解密

CSDN @沐一一沐, 一沐沐一

然后用常规的脚本来计算试一下, 结果错误又出现了, 除数太大, 精度丢失了, 只有前缀flag了:

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

q = 115544353401076813303707955026809960381216232736189720201691794640724518676996765546980623105046200417460503
2557738570794268894873359341064309819398597238444429354081481851125001637430514677365364098870682508134909717057
17411149279581182052591953379888333356928093109561978986910375048128808860432864671882543
p = 150290608270992439844054823303154263794197803561695786056860615174575181277160032222859532335454486914357850
8493430361738389608201808675951696236703639637323159015879466395771072027803177485257094071533274636015480123219
45759392416846089189522151851138821377551427960151260776474250605261723480167088408148729
e = 65537
c = 69449671088154377354289412867841194031383197134557321559250559286465369625976729418058313121306893380149134
5208129640027286271044720726509975040165782816583601312284865683910085471996518868009737549119324912772559966038
3746827031803066026497989298856420216250206035068180963797454792151191071433645946245914916732637007117085199442
8944956674555445174834040065366071214806786880004204222813805393685198071621750997638919886481179377779510698999
7526019001899583490454144756271830743390659202122666688563887702030400561445076308133708283860841475616225382569
7420493509914578546951634127502393647068722995363753321912676

n = 173652311549263483644782768725584927759117606030023943537236034618984057402347150018201115486009149076170038
0665249239168671025627415667788710199717569227772964845608753498761674372464659823446609477954072941358382635514
5277980479040157075453694250572316638348121571218759769533738721506811175866990851972838466307594226293836934116
6596852157756432854658953177558927544733320342344957959361836105695710164005353627626995176867816023020450485321
3142603526087897989216944105946762352306056928557057719923630988815583301372199793396045778465326207613556176983
8704166810384309655788983073376941843467117256002645962737847

d = libnum.invmod(e, (p - 1) * (q - 1)) #invmod(a, n) - 求a对于n的模逆, 这里逆向加密过程中计算ψ(n)=(p-1)(q-1), 对ψ(n)
)保密, 也就是对应根据e*d=1modψ(n), 求出d
#print(hex(d))
m = pow(c, d, n) # 这里的m是十进制形式, pow(x, y[, z])--函数是计算 x 的 y 次方, 如果 z 在存在, 则再对结果进行取模, 其结
果等效于 pow(x,y) %z, 对应前面解密算法中M=D(C)= (C^d) mod n
print(long_to_bytes(m/(2021*1001*p)))
```


总结:

1:

(这里积累第一个经验)

这里就属于 CTF-RSA-tool 工具的进阶理解了, 首先根据回显 `DEBUG: factor N: try Common factor between ciphertext and modulus attack` 我在文件 `factor_N.py` 找到了对应的部分:

```
# 密文与模数不互素
def comfact_cn(N, c):
    log.debug('factor N: try Common factor between ciphertext and modulus attack')
    # Try an attack where the public key has a common factor with the ciphertext - sourcekris
    if c:
        commonfactor = libnum.gcd(N, c)
        if commonfactor > 1:
            q = commonfactor
            p = N / q
            return p, q
```

CSDN @沐一一沐, 一沐沐一

2:

(这里积累第二个经验)

然后一开始我的思路是用在线大数除法工具解决精度丢失的问题, 找不到。于是进一步修改 CTF-RSA-tool 工具, 终于在 `RSAutils.py` 中找到输出结果部分修改一下, 成功输出 `flag` :

```
# 分解得到p q, 或用户输入了p和q, 计算d
if self.p and self.q:
    self.d = libnum.invmod(self.e, (self.p - 1) * (self.q - 1))
    log.debug('d我在这里啊!!!=' + hex(self.d))
else:
    log.error('can not factor N, please offer p and q or d')
    return

# --private 是否需要打印私钥
if self.args.private:
    log.info('\np=%d\nq=%d\nd=%d\n' % (self.p, self.q, self.d))
    log.info('private key:\n' + RSA.construct((long(self.n), long(self.e),
                                             long(self.d), long(self.p), long(self.q))).export())

# 不需要解密, 直接返回
if not self.c:
    return
self.plain = pow(self.c, self.d, self.n)

# 打印解密出来的明文
print '666'
print 'plain =', self.plain
print '666'
log.info(libnum.n2s(self.plain/(2021*1001*self.q)))

# d泄露攻击, 根据过期的(N, e1, d1), 和一个新的e2, 返回d2
def d_leak(N, e1, d1, e2):
```

标记1

标记2和对应修改

CSDN @沐一一沐, 一沐沐一

解毕! 敬礼!