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

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


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)

#n = 173652311549263483644782768725584927759117606030023943537236034618984057402347150018201115486
#c = 69449671088154377354289412867841194031383197134557321559250559286465369625976729418058313121306
```

这里的明文经过了简单的乘法进行二次处理

.

.

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  173652311549263483644782768725584927759117606030023943537236034618984057402347150018201115486009149076170038066524923916867102562741566778871019971756922777296484560875349876167437246465982344660947795407294135838263551452779804790401570754536942505723166383481215712187597695337387215068111758669908519728384663075942262938369341166596852157756432854658953177558927544733320342344957959361836105695710164005353627626995176867816023020450485321314260352608789798921694410594676235230605692855705771992363098881558330137219979339604577846532620761355617698387041668103843096557889830733769418434671172560026459627378470
e is 65537
c is  694496710881543773542894128678411940313831971345573215592505592864653696259767294180583131213068933801491345208129640027286271044720726509975040165782816583601312284865683910085471996518868009737549119324912772559966038374682703180306602649798929885642021625020603506818096379745479215119107143364594624591491673263700711708519944289449566745554451748340400653660712148067868800042042228138053936851980716217509976389198864811793777795106989997526019001899583490454144756271830743390659202122666688563887702030400561445076308133708283860841475616225382569742049350991457854695163412750239364706872299536375332191267 6
```

.

.

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：

```
└─$ python2 solve.py --verbose  -i /home/wdnmd/桌面/1.txt
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:80
DEBUG: factor N: try Common factor between ciphertext and modulus attack
DEBUG: d = 0x233648b8421f05cf2307820eedcfae249e38abb155880d8f9f44622103ad8e9e1375f6abe86
ef6af69fb29958a3526c0a60923bd30630c66e624275d3dd7125698ca508003c3b3c8316be8c4e13340a9b2c
87f4e3649742a8b7e9f440bee7260c42195b5e601d08b083c559a5acd10cdc17e1cbac588704afc9dfff1a32
7c7eb1d692fde6db036ff44f36ac25e0b01ebe40043c5c1b78c4feb53ce1952fddfae3223b982cc9126e7fb2
c899e5bf8d91f3616e681158824c3a2816c7ee1f4908e1a002c5cc4f4b7e20557f0eebebe9a788f20ceac80e
f1f3bc760dc0e2880146d46820b8fac2cb52f68cf44b160bcc94b660a0a37e1ad95c81L
INFO:
S<%'??????A???sU??A?
?                    ?S???L????"?0???e+??▒?Q??
 N??O??□????}T3?%?K?Q_?Z?'d?c?1Tg?#H??+?.???m?g?"??/H?D?1C????
```


.

.

(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：

```python
# 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
```


.

.

原来是密文与模数不互素，难怪我在`http://www.factordb.com/index.php`网站中分解不出`N`。而且也不是像以前积累的2021年9月广州羊城杯的`BigRsa`一样有`两个N`的`模不互素`：

Search | Sequences | Report results | Factor tables | Status | Downloads | Login

173652311549263483644782768725584927759117606030023943537236034618984057402347150018201115486 Factorize!

Result:

status (?)	digits	number
C	617 (show)	1736523115...47<617> = 1736523115...47<617>

分解不出N

More information

Report factors

Format: Auto detect (slow)

Report

.

.

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):
    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
            print('p=',p)
            print('q=',q)
            return p, q
```

打印p、q然后常规RSA解密

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 = 11554435340107681330370795502680996038121623273618972020169179464072451867699676554698062310504620041746050325577385707942688948733593410643098193985972384444293540814818511250016374305146773653640988706825081349097170571741114927958118205259195337988833335692809310956197898691037504812880886043286467188254 3
p = 15029060827099243984405482330315426379419780356169578605686061517457518127716003222285953233545448691435785084934303617383896082018086759516962367036396373231590158794663957710720278031774852570940715332746360154801232194575939241684608918952215185113882137755142796015126077647425060526172348016708840814872 9
e = 65537
c =  69449671088154377354289412867841194031383197134557321559250559286465369625976729418058313121306893380149134520812964002728627104472072650997504016578281658360131228486568391008547199651886800973754911932491277255996603837468270318030660264979892988564202162502060350681809637974547921511910714336459462459149167326370071170851994428944956674555445174834040065366071214806786880004204222813805393685198071621750997638919886481179377779510698999752601900189958349045414475627183074339065920212266668856388770203040056144507630813370828386084147561622538256974204935099145785469516341275023936470687229953637533219126 76
n = 17365231154926348364478276872558492775911760603002394353723603461898405740234715001820111548600914907617003806652492391686710256274156677887101997175692277729648456087534987616743724646598234466094779540729413583826355145279804790401570754536942505723166383481215712187597695337387215068111758669908519728384663075942262938369341166596852157756432854658953177558927544733320342344957959361836105695710164005353627626995176867816023020450485321314260352608789798921694410594676235230605692855705771992363098881558330137219979339604577846532620761355617698387041668103843096557889830733769418434671172560026459627378 47
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)))
```

```
└─$ python 1.py
b'flag{M`\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00'
```

.

.

(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


```
00163743051467736533640988706825081349097170571741114927958118205259195337988833335692809310
95619789869103750481288088604328646718825 43L)
('q=', 15029060827099243984405482330315426379419780356169578605686061517457518127716003222 2
85953233545448691435785084934303617383896082018086759516962367036396373231590158794663957 71
07202780317748525709407153327463601548012321945759392416846089189522151851138821377551427 96
01512607764742506052617234801670884081487 29L)
DEBUG: d我在这里啊！！！=0x233648b8421f05cf2307820eedcfae249e38abb155880d8f9f44622103ad8e9e
1375f6abc860c5ef6af69fb29958a3526c0a60923bd30630c66e624275d3dd7125698ca508003c3b3c8316be8c4
e13340a9b2c3a387f4e3649742a8b7e9f440bee7260c42195b5e601d08b083c559a5acd10cdc17e1cbac588704a
fc9dfff1a324f37c7eb1d692fde6db036ff44f36ac25e0b01ebe40043c5c1b78c4feb53ce1952fddfae3223b982
cc9126e7fb2b97c899e5bf8d91f3616e681158824c3a2816c7ee1f4908e1a002c5cc4f4b7e20557f0eebebe9a78
8f20ceac80ea7af1f3bc760dc0e2880146d46820b8fac2cb52f68cf44b160bcc94b660a0a37e1ad95c81L
666
plain = 8395571914768579416654762333305535017137164972943391521259329928396505927214851604 0
38721361104183154589483487275308763686847067038977834754199610943609533852797817525000137 59
97922882066873113085787724087544609211118528854080008317479153365931620664925464256150880 65
13443981994150015387695427576272218347161581110192205737072101697927873587752287392581838 32
29872971509761250701141 22865
666
INFO: flag{Math_1s_1nterest1ng_hah}
```

.

.

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
```


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))).exportK

# 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

解毕！敬礼！