# 2020年羊城杯Crypto的RRRRRRRSA

原创

沐一 · 林 于 2021-11-27 16:07:07 发布 104 收藏 2

分类专栏： CTF 密码学 文章标签： ctf



本文链接：https://blog.csdn.net/xiao__1bai/article/details/121576940



CTF 同时被 2 个专栏收录

167 篇文章 6 订阅

订阅专栏

密码学

51 篇文章 1 订阅

订阅专栏

## 2020年羊城杯Crypto的RRRRRRRSA

本来是在复现第七届“湖湘杯” Crypto 的 `signin`，但是查的资料都说和 2020 年羊城杯 Crypto 的 `RRRRRRRSA` 的原理是一样的，所以也顺便把这道题看了。

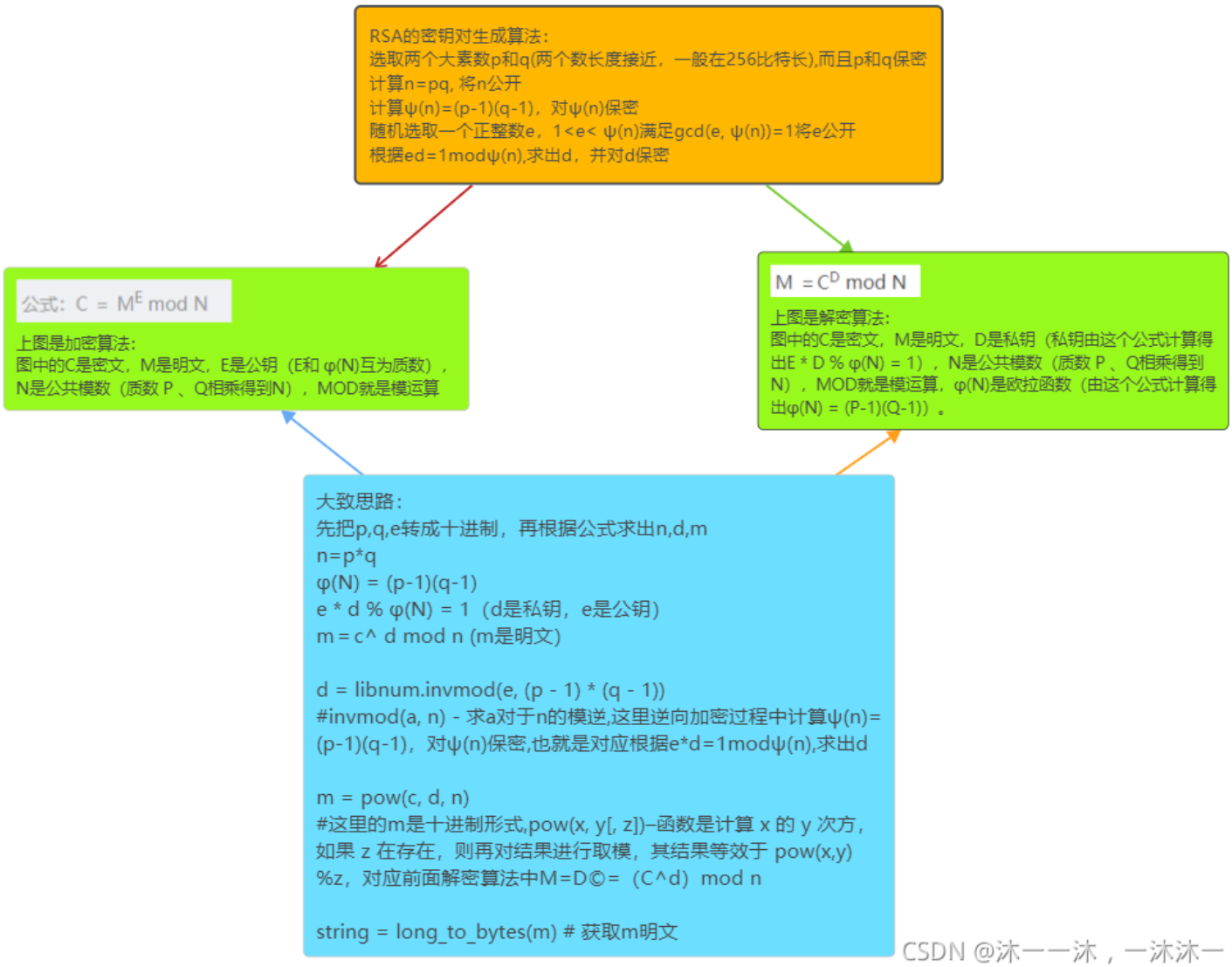
**打开附件，代码如下：**

```
import hashlib
import sympy
from Crypto.Util.number import *

flag = 'GWHT{************}'

flag1 = flag[:19].encode()
flag2 = flag[19:].encode()
assert(len(flag) == 38)

P1 = getPrime(1038)
P2 = sympy.nextprime(P1)
assert(P2 - P1 < 1000)

Q1 = getPrime(512)
Q2 = sympy.nextprime(Q1)

N1 = P1 * P1 * Q1
N2 = P2 * P2 * Q2

E1 = getPrime(1024)
E2 = sympy.nextprime(E1)

m1 = bytes_to_long(flag1)
m2 = bytes_to_long(flag2)

c1 = pow(m1, E1, N1)
c2 = pow(m2, E2, N2)


output = open('secret', 'w')
output.write('N1=' + str(N1) + '\n')
output.write('c1=' + str(c1) + '\n')
output.write('E1=' + str(E1) + '\n')
output.write('N2=' + str(N2) + '\n')
output.write('c2=' + str(c2) + '\n')
output.write('E2=' + str(E2) + '\n')
output.close()
```

由于涉及了 `RSA` 的 `连分数低解密指数` 攻击，我并没有这方面的知识储备，所以下面话语中大部分参考自师傅 `前方是否可导？` 和 `mortal15`

> 对应的参考博客地址：
> https://blog.csdn.net/weixin_44110537/article/details/108560055
> https://blog.csdn.net/a5555678744/article/details/117701126

先放上以前的`RSA`图来回顾一下基本流程：




.

.

**wiener attack原理阐述：**

`在 RSA 中我们根据ed =1 mod ψ(n)有如下推导：`

ed =1 mod ψ(n) ⇒ ed =1 mod phi ⇒ ed -1 =k*phi* ⇒ *e/phi - k/d =1/(d*phi)

可以发现当 p,q 很大时，phi 和 n 是接近的，即phi(N)≈N，且phi(N)非常大。1/(d*phi) 很小，说明 e/phi 和 k/d 很接近，于是就有 `e/n` 和 `k/d` 很接近。

当 e 很大时，由于 e 和 N 都是`已知`的，通过对 e/n 进行`连分数展开`，然后对每一项求其渐进分数，通过遍历渐进分数 k/d 很有可能就被 e/n 的众多项渐进分数中的一项所`覆盖`，只要`检验`一下 ed mod phi(N) 看它是不是1就知道对不对了。
假设覆盖它的是 k1/d1，那么 `k1=k`；`d1=d`，得到两个数之后就可以用常规的 `RSA` 解题流程解题了。

`wiener attack` 就是这样一个依靠连分数进行的攻击方式，适用于非常接近某一值（比如1）时，求一个`比例关系`，通过该比例关系再来反推关键信息就简单很多，这种攻击对于解密指数 `d` 很小，即 `e` 和 `N` 很相近，位数将近一样时有很好的效果。
特别的：wiener attack并不是`针对于`解密指数的攻击方式，实际上，在低解密指数的情况下，任何`比例`非常`接近另外一个`已知比例的情况下都可以尝试用这个方法得到RSA中重要解密信息，不一定是 `d`，也有可能是 `p` 或者 `q` 。

.

.

**补充1：**

这里可能会有疑问,如果 `gcd(k,d)!=1` 那么对于最简的 k1/d1 来说是否应该存在t使得 tk1=k td1=d 呢? 但其实这里 gcd(k,d) 一定为 1 即 k，d 一定互质.证明也很简单.前面我们可以得到: ed-kphi=1 对于这么一个式子,在扩展欧几里得里有如果gcd(d,k)!=1 那么该该方程 `无解` .(不互质可以提出一个不为1公因子,除过去左边全是整数而右边却是真分数显然不可能)

.

.

**补充2：什么是连分数**

连分数（continued fraction）是特殊繁分数。如果a0，a1，a2，…an，…都是整数，则将分别称为无限连分数和有限连分数。可简记为a0 ，a1，a2，…，an，…和a0，a1，a2，…，an。一般一个有限连分数表示一个有理数，一个无限连分数表示一个无理数。如果a0，a1，a2，…，an，…都是实数，可将上述形式连分数分别叫无限连分数和有限连分数 。近代数学的计算需要，还可将连分数中的a0，a1 ，a2，…，an，…取成以x为变元的多项式。在近代计算数学中它常与某些微分方程式差分方程有关，与某些递推关系有关的函数构造的应用相联系。

`举例如：` 127/52 = 2 + 1/2+ 1/3+ 1/1+ 1/5

那连分数项目就是[2,2,3,1,5]

.

.

.

.

.

.

现在我们回过头来看回题目，题目先是把 `flag` 分成两部分（其实两部分加密方式是一样的），然后生成两个非常 `接近` 的大素数 `P1，P2` ，然后生成了两个 `相邻的较小` 的大素数 `Q1、Q2` ，用P1的平方和Q1相乘得到N1，N2则是用P2的平方和Q2相乘，再用两个 `相近` 巨大的加密指数E加密。

但是和普通的wiener attack `不同` 的是，e与N并没有 `相近` 到相除约 `为1` 的地步，相差还是很大的，也就是说解密指数 `d` 也许还是很大的。

有意思的是 `e` 和 `N` 的关系 `不符合` 利用条件，但是 `N1` 和 `N2` 的关系却适合，由于 `（P1/P2）**2` 接近于 `1` ，所以 `N1/N2` 的 `比例` 很相近于 `Q1/Q2` 。

.

.

**仿照 e/n 来展开：**

通过：N1/N2=(P1/P2)**2 * (Q1/Q2)显然我们可以知道的是N1/N2 <Q1/Q2

所以在 `Q1/Q2` 在区间(N1/N2,1)之间 (这是关键，不用P1/P2是因为平方不好求）

尝试对 `N1/N2` 进行 `连分数展开` 并求其各项渐进分数，其中某个连分数的 `分子` 可能就是Q1（这个可以依靠 `N1%Q1==0` 来验证）

.

.

代码如下：

```
import gmpy2
N1=60143104944034567859993561862949071559877219267755259679749062284763163484947626697494729046430386559610613113754453726683312513915610558734802079868190554644983911078936369464590301246394586190666760362763580192139772729890492729488892169933099057105842090125200369295070365451134781912223048179092058016446222199742919885472867511334714233086339832790286482634562102936600597781342756061479024744312357407750731307860842457299116947352106025529309727703385914891200109853084742321655388368371397596144557614128458065859276522963419738435137978069417053712567764148183279165963454266011754149684758060746773409666706463583389316772088889398359242197165140562147489286818190852679930372669254697353483887004105934649944725189954685412228899457155711301864163839538810653626724347c1=55094296873556883585060020895253176070835143350249581136609315815308788255684072804968957510292559743192424646169207794748893753882418256401223641287546922358162629295622258913168323493447075410872354874300793298956869374606043622559405978242734950156459436487837698668489891733875650048466360950142617732135781244969524095348835624828008115829566644654403962285001724209210887446203934276651265377137788183939798543755386888532680013170540716731
```

```
665667026925131880050151757980340115499688123302521017629355454202405254009389038743796474746076549871309201816019663792820419019415419938927666668543656566523639748170970364455532870581889226949938079704455405411865632138947482122472553369352085604773657840258185416594159925417801951561518310289471664768096974274470521886845545083 2
E1=125932919717342481428108392434488550259190856475011752106073050593074410065655587870702051419898088541590032209854048032649625269856337901048406066968337289491951404384300466543616578679539808215698754491076340386697518948419895268049696498272031094236309803803729823608854215226233796069683774155739820423103
N2=601431049440345678599935618629490715598772192677552596797490622847631634849476266974947290464303865596106131137544537266833125139156105587348020798681956336474317328753921214586843318433067308894244186200693225782652363514075910293385198095389952498969051376423424356595729177141835433052437156643807877975620110063987303209809947479397915618856229499126982467017693214303259029120030416787744407040565978620935309810406968725228689211390412473625922572854239488709441370197451612115858459270192597095012375508189182721896064364139927593283188717651718441535274243479854627670281353765523024638613244081781838421393302449066067763590504829772567289102786879961061529710288786531235335597601677112702651714416230568739036699186942590435800170816713492320518707164935574345175791212 c2=3932844614015625757148418471386131972290586419755672073085277305914790228312325276765143027835795087262677834859689771132094244969327060377687030110288140530365155871908545428114239565205621724175165663181258054418043434984023691976543312238911686082759371159373238556232825575950935529866236150861153197238699523990851327323623985885458684584968686536078029035028713909214358703739680170435169273698595515293560198775885975942188667090773512013769803990016132739795175885287529144218885094627377173301150492232562224083828809794630982505109456668547950346193850237352098368429665897170092206942678823647657523618904010284841854763429021417516776743147500321605670109427589921141997934080271168498971013021592652638713853881953119981084147521814260669115292823636253418162220134 7
E2=125932919717342481428108392434488550259190856475011752106073050593074410065655587870702051419898088541590032209854048032649625269856337901048406066968337289491951404384300466543616578679539808215698754491076340386697518948419895268049696498272031094236309803803729823608854215226233796069683774155739820425393
def continuedFra(x, y):   #不断生成连分数的项,如127/52 = 2 + 1/2+ 1/3+ 1/1+ 1/5，生成[2,2,3,1,5]
    cF = []
    while y:
        cF += [x // y]
        x, y = y, x % y  #这里是连分数生成项的算法
    return cF

def Simplify(ctnf):    #对前面生成的连分数项化简
    numerator = 1
    denominator = 0
    for x in ctnf[::-1]:   #注意这里是倒叙遍历，从后面把连分数项合成总和。
        numerator, denominator = x * numerator + denominator, numerator
    return (numerator, denominator)   #把连分数分成和算出来的分母以元组的形式导出来，如Simplify(continuedFra(127,52))生成(127, 52)

def getit(c):
    cf=[]
    for i in range(1,len(c)):
        cf.append(Simplify(c[:i]))  #各个阶段的连分数的分子和分母
    return cf        #得到一串连分数，如：[(2, 1), (5, 2), (17, 7), (22, 9)]

def wienerAttack(e, n):    #低解密指数攻击，自己修改要碰撞的分子或分母
    cf=continuedFra(e,n)
    for (Q1,Q2) in getit(cf):   #遍历得到的连分数，令分子分母分别是Q1，Q2，因为前面我们说了N1/N2=(p1/p2)**2 (q1/q2)
        if Q1 == 0:
            continue
        if N1%Q1==0 and Q1!=1:  #满足这个条件就找到Q1了，其实这里的Q2也是对的。
            return (Q1,Q2)
    print('没找到能覆盖的分子/分母')

Q1,Q2=wienerAttack(N1,N2)    #找出一个Q1,其实这里也可以找出Q2的，但处于p2=sympy.nextprime(p1)的限制，只能用p1求出p2，不能用p2求出p1

from Crypto.Util.number import *
P1=gmpy2.iroot(N1//Q1,2)[0]
P2=gmpy2.next_prime(P1)    #p1对了，p2也会对
```

```
phi1=P1*(P1-1)*(Q1-1)     #求出phi1，也就是ψ(n1)
phi2=P2*(P2-1)*(Q2-1)     #求出phi2，也就是ψ(n2)
d1=gmpy2.invert(E1,phi1)    #逆模求d
d2=gmpy2.invert(E2,phi2)    #逆模求d
m1=long_to_bytes(gmpy2.powmod(c1,d1,N1))  #普通解密算法，但是要用将Long型数转为字节型数据
m2=long_to_bytes(gmpy2.powmod(c2,d2,N2))  #普通解密算法，但是要用将Long型数转为字节型数据
print((m1+m2))       #拼接flag字符。
```

结果：


.

.

.

解毕！敬礼！