RSA算法原理及CTF解题

原创

WHOAMIAnony 于 2021-03-12 20:42:21 发布

文章标签： 密码学

The RSA encryption algorithm is an asymmetric encryption algorithm. RSA is widely used in public key cryptography and electronic commerce. RSA was proposed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman. All three of them were working at the Massachusetts Institute of Technology. RSA is composed of the first three letters of their last names.

文章目录

RSA算法原理

什么是RSA

RSA加密

RSA解密

生成密钥对

1. 求N
2. 求欧拉函数φ(N)
3. 求E
4. 求D

CTF中的常见RSA题型

已知p、q、e，求d

已知p、q、e、密文c，求明文m

已知q、p、dq、dp、密文c，求明文m

已知e、n（非常大）、 dp 和密文c，求明文m

已知n（非常大）、e、d，求p、q

已知e、n、dp、密文c，求明文m

已知c1、c2、n、e1、e2，求明文m

n分解出多个不同的因子时 ，求明文m

已知 密文文件 和 公钥文件，求明文m

方法一：(key.pem 和 cipher.bin)

方法二：（flag.b64 和 key.pub）

已知私钥文件 private.pem 和密文文件 flag.enc，求明文m

提取私钥文件中的信息

解码公钥文件

利用 公钥文件 生成 私钥文件（已知公钥求私钥）

爆破攻击方法

低加密指数分解攻击（比如 e=2,e=3）

（1）e=2，可以把密文c开平方求解

（2）e=3，可进行小明文攻击

低解密指数攻击（e过大或过小）

多重基数的RSA低解密指数攻击

低加密指数广播攻击（n、c不同，e和m相同）

模不互素（存在两个或更多非常大的模数 n，且n1和n2不互质）

共模攻击（m，n相同、e，c不同，且e1 和 e2互质）

已知c，e，n（非常大），和 dp，dq，求解明文m

RSA算法原理

什么是RSA

RSA算法是现今使用最广泛的公钥密码算法，也是号称地球上最安全的加密算法。在了解RSA算法之前，先熟悉下几个术语。

根据密钥的使用方法，可以将密码分为对称加密和公钥加密：

- 对称密码：加密和解密使用同一种密钥的方式
- 公钥密码：加密和解密使用不同密码的方式，因此公钥密码通常也称为非对称密码。

RSA加密

RSA的加密过程可以使用一个通式来表达

$$密文 = 明文^{E} mod N$$

也就是说RSA加密是对明文的E次方后除以N后求余数的过程。

从通式可知，**只要知道E和N任何人都可以进行RSA加密了**，所以说E、N是RSA加密的密钥，也就是说**E和N的组合就是公钥，我们用(E,N)来表示公钥。**

$$公钥 = (E, N)$$

不过E和N不并不是随便什么数都可以的，它们都是经过严格的数学计算得出的，关于E和N拥有什么样的要求及其特性后面会讲到。顺便啰嗦一句E是加密（Encryption）的首字母，N是数字（Number）的首字母

RSA解密

RSA的解密同样可以使用一个通式来表达

$$明文 = 密文^{D} mod N$$

也就是说对密文进行D次方后除以N的余数就是明文，这就是RSA解密过程。知道D和N就能进行解密密文了，所以D和N的组合就是私钥

$$私钥 = (D, N)$$

从上述可以看出RSA的加密方式和解密方式是相同的，加密是求“E次方的mod N”，解密是求“D次方的mod N”
此处D是解密（Decryption）的首字母；N是数字（Number）的首字母。

总结一下

公钥	**（E，N）**
私钥	**（D，N）**
密钥对	**（E，D，N）**
加密	密文＝明文^E mod N
解密	明文＝密文^D mod N

生成密钥对

既然公钥是（E，N），私钥是（D，N）所以密钥对即为（E，D，N）但密钥对是怎样生成的？步骤如下：

1. 求N
2. 求L（L为计算过程的中间数）
3. 求E
4. 求D

1. 求N

准备两个质数p，q。这两个数不能太小，太小则会容易破解，将p乘以q就是N

- **N = p*q**

2. 求欧拉函数φ(N)

然后计算欧拉函数φ(N)或L：

- **φ(N) = (p-1) * (q-1)**

3. 求E

E 必须满足两个条件：E 是一个比1大比φ(N)小的数，E 和 φ(N) 的最大公约数为1，即 E 和 φ(N) 互质

用gcd(X,Y)来表示X，Y的最大公约数则E条件如下：

- **1 < E < φ(N)**
- **gcd(E,φ(N)) = 1**

之所以需要E和φ(N)的最大公约数为1是为了保证一定存在解密时需要使用的数D。现在我们已经求出了E和N也就是说我们已经生成了密钥对中的公钥了。

4. 求D

数D是由数E计算出来的。D、E和φ(N)之间必须满足以下关系：

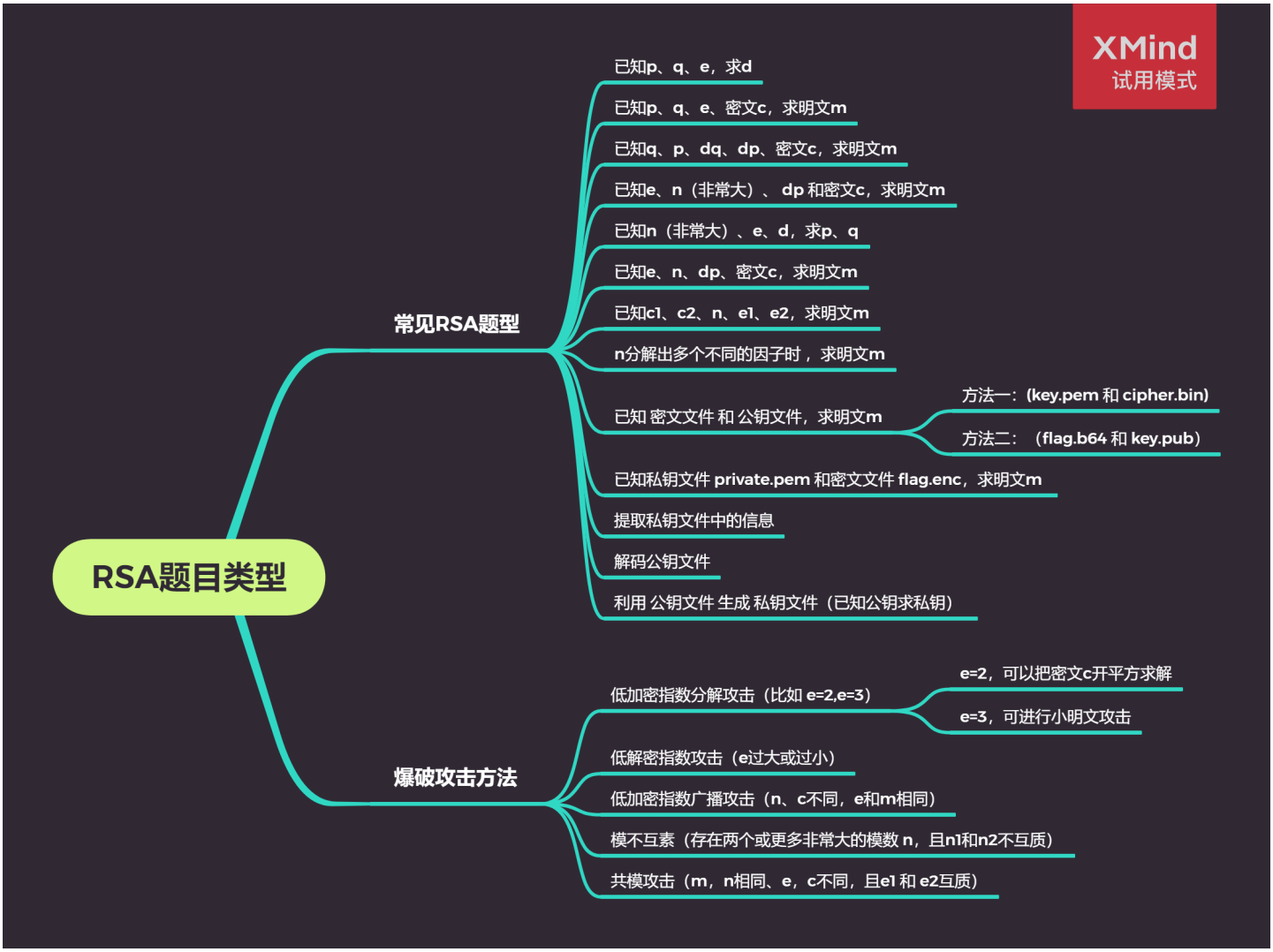
- **1 < D < φ(N)**
- **(E * D) mod φ(N) = 1**

只要D满足上述2个条件，则通过E和N进行加密的密文就可以用D和N进行解密。
简单地说条件2是为了保证密文解密后的数据就是明文。
现在私钥自然也已经生成了，密钥对也就自然生成了。

小结下：

求N	N＝ p * q；p，q为大质数
求φ(N)	φ(N)＝ (p－1，q－1)
求E	1 < E < φ(N)，gcd(E，φ(N))=1；E，φ(N)最大公约数为1（E和L互质）
求D	1 < D < φ(N)，(E＊D) mod φ(N) ＝ 1

CTF中的常见RSA题型



已知p = 17、q = 19。

计算N:

```
N=P*Q=323
```

欧拉函数，即L:

```
ϕ(N)=(p-1)*(q-1) = 144
```

计算公钥e:

```
要求1<e<ϕ(N),即1<e<20，并且e要与ϕ(N)互质，那么我们取最小值e=5

此时公钥=(E, N) = (5, 323)
```

计算私钥d:

```
要求1 < d < ϕ(N)并且(e*d) mod ϕ(N) = 1，即5*d%144=1，那么d=29

此时私钥=(D, N) = (29, 323)
```

d可以用python gmpy2库的`gmpy2.invert(e,ϕ(N))`求逆元快速算出来。

1. 公钥加密：

准备的明文m或M必须时小于N的数，因为加密或者解密都要mod N其结果必须小于N

假设明文m = 123，则：

```
密文 = 明文^e mod N
即
密文 = 123^5 % 323 = 225
```

2. 私钥解密：

```
明文=密文^D mod N
即
225^29 % 323=123
```

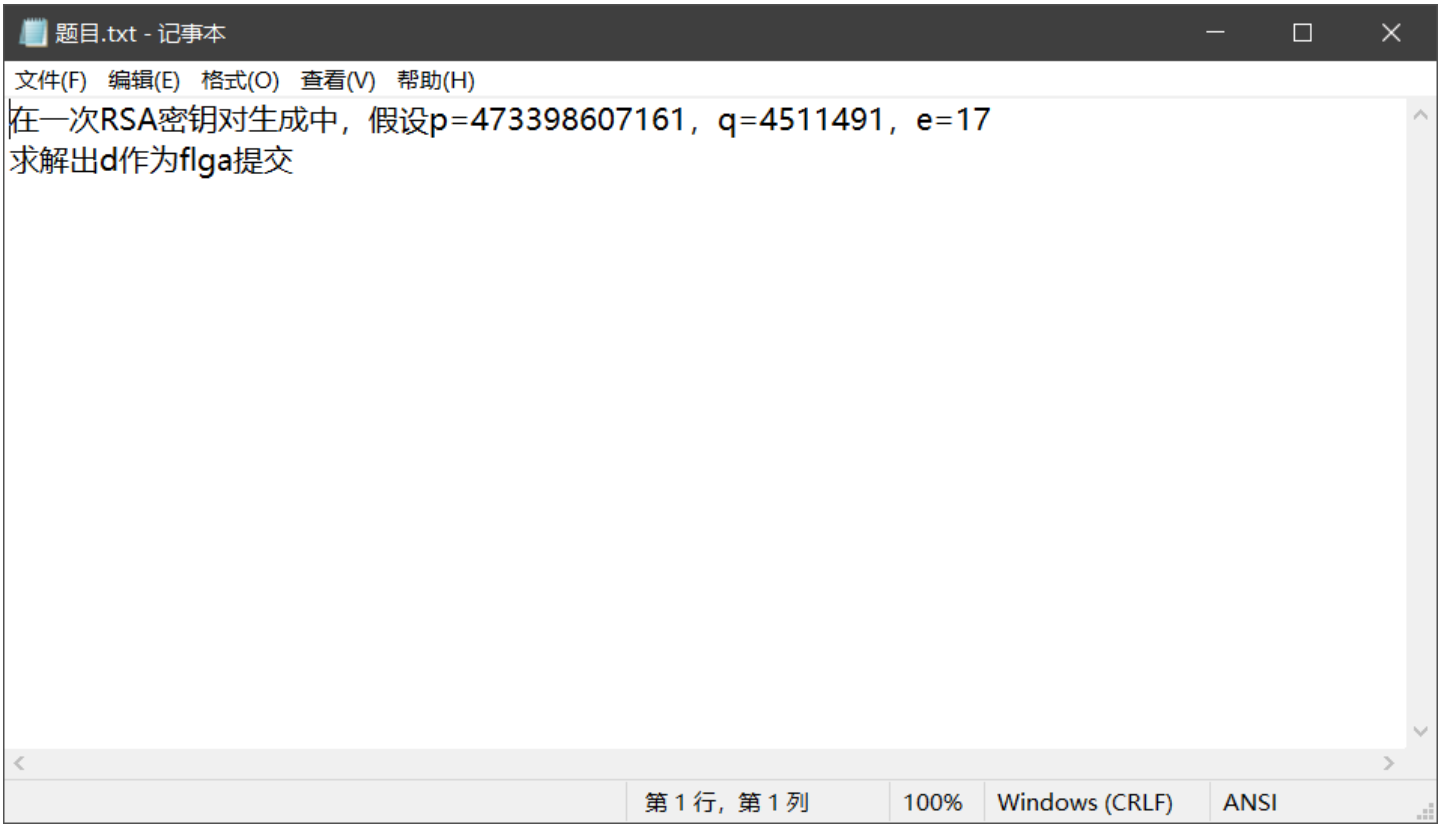
> 参数N和E是公开的但是D是私有的并且绝不能公开！P和Q在生成密钥后便不再需要了，但是必须销毁。
> 为了从公钥(N,E)得到D,需要试图分解N为它的两个素数因子。对于一个很大的模数N（512位或更大）
> 要想分解出它的P和Q是件非常困难的事。
> RSA 加密模式的所有安全性都依赖于大数分解(但是还没有数学上的证明)。

我们学习RSA不可能去手算，因此我们需要自己编写脚本！编写python脚本时我们需要**gmpy2**库。

> GMP（GNU Multiple Precision Arithmetic Library，即GNU高精度算术运算库），它是一个开源的高精度运算库，其中不但有普通的整数、实数、浮点数的高精度运算，还有随机数生成，尤其是提供了非常完备的数论中的运算接口，比如Miller-Rabin素数测试算法、大素数生成、欧几里德算法、求域中元素的逆、Jacobi符号、legendre符号等。
> gmpy2是Python的一个扩展库，是对GMP的封装，它的前身是gmpy，经过其作者的调整和封装，使得gmpy2的使用大大简化。

已知p、q、e，求d

[BUUCTF-Crypto]RSA



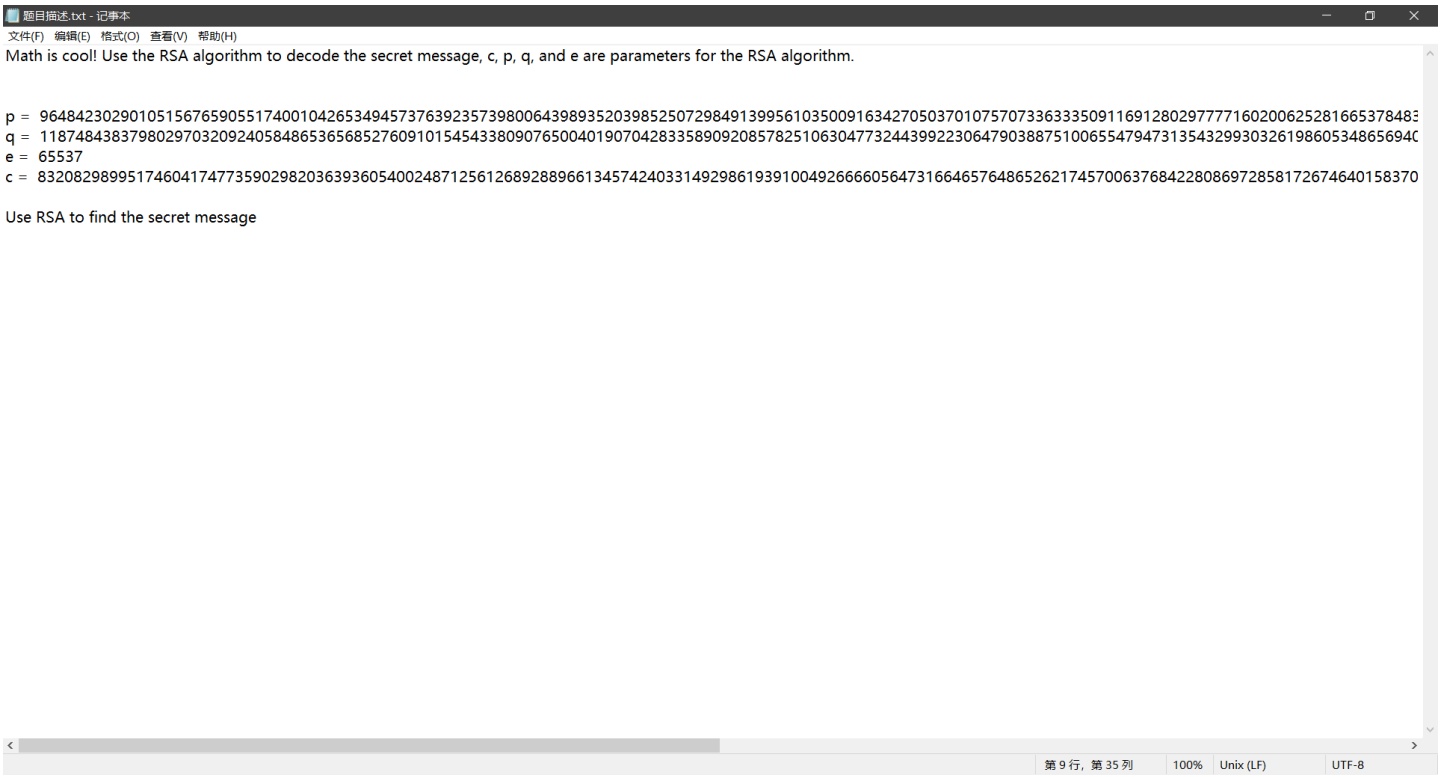
由题意可知：p，q，e，求d。直接上python脚本：

```
#coding=utf-8
import gmpy2
p = 473398607161
q = 4511491
e = 17
d = gmpy2.invert(e,(p-1)*(q-1))  # gmpy2.invert(e,φ(N))
print d
```

得到d等于125631357777427553，即flag为flag{125631357777427553}。

已知p、q、e、密文c，求明文m

[BUUCTF-Crypto]rsarsa



已知p、q、e、密文c，求明文m，根据公式`m=pow(c,d,n)`。写出脚本：

```
#coding=utf-8
import gmpy2
def Decrypt(c,e,p,q):
 L=(p-1)*(q-1)
 d=gmpy2.invert(e,L)
 n=p*q
 m=gmpy2.powmod(c,d,n)
 flag=str(m)
 print "flag{"+flag+"}"
if __name__ == '__main__':
 p =  964842302901051567659055174001042653494573763923573980064398935203985250729849139956103500916342705037010757073363335091169128029777716020062528166537848 3
 q =  118748438379802970320924058486536568527609101545433809076500401907042833589092085782510630477324439922306479038875100655479473135432993032619860534865694 07
 e =  65537
 c =  832082989951746041747735902982036393605400248712561268928896613457424033149298619391004926666056473166465764865262174570063768422808697285817267464015837058999417682141387422596893348407356335530538876418476511737762518202930872128856701803674068074067659236389731613758173927377478327627516901044238690190 34
 Decrypt(c,e,p,q)
```

注意：有时得到的明文需要转为十六进制后再转为字符串。

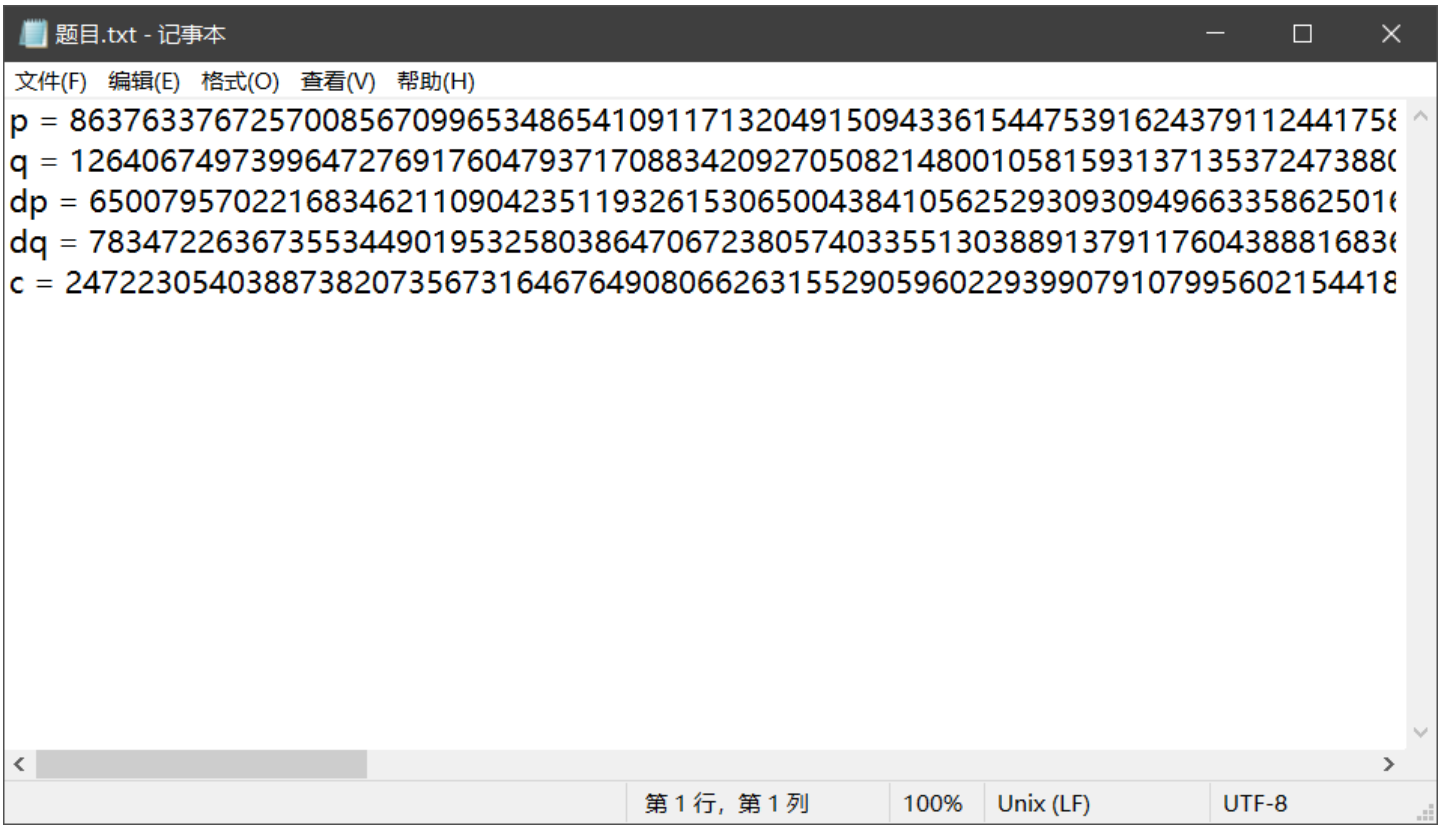
执行后得到flag为：flag{557744663355446657776887998 8}。

gmpy2.powmod(x,y,z) 这个函数就是表示x 的 y 次幂后除以z的余数即：

```
gmpy2.powmod(x,y,z) = x^y % z
```

已知q、p、dq、dp、密文c，求明文m

[BUUCTF-Crypto]RSA1



这道题并没有直接给公钥，但是泄露了dp和dq，应该是让我们根据这些条件求出明文m。dp和dq一开始并不知道这是个啥，看过WriteUp后才得知有如下关系：

```
dp=d mod(p-1)
dq=d mod(q-1)
即
dp=d%(p-1)
dq=d%(q-1)
```

但现在也还没有明白最后怎么推出来的，可以参考下面：

> https://www.cnblogs.com/P201521440001/p/11415504.html#rsa%E7%B3%BB%E5%88%97
> https://beiyuouo.github.io/blog/ctf-buuctf/

写出如下脚本：

```
#coding=utf-8
import gmpy2
p = 86376337672570085670996534865410911713204915094336154475391624379112441758856678063984117905240835534451581135022277452062053276909395040329946999020532 29
q = 12640674973996472769176047937170883420927050821480010581593137135372473880595613737337630629752577346147039284030082593490776630572584959954205336880228469
dp = 65007957022168346211090423511932615306500438410562529309309496633586250168818328407280660261502646930761093548740998413804548817160977783072681169105829 29
dq = 78347226367355344901953258038647067238057403355130388913791176043888168367455609809825679567351220196300217543876276751696804359958252753916081112055004 1
c = 24722305403887382073567316467649080662631552905960229399079107995602154418176056335800638887527614164073530437657085079676157350205351945222989351316076486573599576041978339872265925062764318536089007310270278526159678937431903862892400747915525118983959970607934142974736675784325993445942031372107342103852

I = gmpy2.invert(q,p)
mp = pow(c,dp,p)                #求幂取模运算，mp = c^dp % p
mq = pow(c,dq,q)                #求幂取模运算，mq = c^dp % q

m = (((mp-mq)*I)%p)*q+mq        #求明文公式

print(hex(m))           #转为十六进制
// 最后得到十六进制数后有时还要转化为字符串
```

注意：有时得到的明文需要转为十六进制后再转为字符串。

最后得到flag为：flag{W31c0m3_70_Ch1n470wn}。

已知e、n（非常大）、 dp 和密文c，求明文m

领航杯2019的一道题，EasyRSA：给了e、n、c 由于特别大，没法直接用质因数分解求得 q， p

```
phint = d % (p - 1)    其实 phint = dp

qhint = d % (q - 1)    其实 qhint = dq
```

phint和qhint也就是其他常见文章里的dp和dq，他俩加上e、n和密文c全部已知的话是可以实现任意密文c解密。

```
import gmpy2
import libnum
e=65537

n=169697521655091326276302669687488543303407016921254276195598364883502982347355714803530786149755803784673559523337553139355165137735521633929526563214902684525566048589668999562421070084105586579243442956519392973280079322457416609105100329695275982662705110048576745348022033873996782318808942523284311332246535449486612837776459850282076095266548166451555589151977450625691245874123787160498140406706650794800556448734707566029933872619395669588062965997829434601415820451509710312112186170912832841185737140292663312273273987242651703526467940687027896459808100055493763995351108200524724198468018091101865571621270

dp=178162577529102887026968525752110809032954301272870546778254691395153764262362176924644112218994867137499094640516445986741064682559131062261837911628429379409097029216526333474939300999933541308990379662432616803961828707819264649048853406280396041879087489043552939304738922871883524437064521518735808180 5

c=0x6c78dcee37830f3ec4ab4989d40fbb595060b3fbc395d52ad26defc13372c1a3948c5388f4e450e46e016c7803133d6881e5efc3b90a4789448097c94124590b1e7949f2524d7edccd61a27691c18d090ac1f54643b56314130604541758 1e3b263f4ad2816136a48b106f3058b08e2a810f4ae8ef25916cc110b41ac8158ce69ecbe20fc60c1ddb20154c6646bc5142aefe47abf053a8ac949d5bc057bb18b191ad08070fe9ec5d76b1fceae685514532448c1b388b2d38e7241ac19c296e95e4e021a3a4015d909a1d53a2eb7fa86f6329f4e6c937f958be576c58fab4d9c9126999c99bb28718efc41a6f5db52b47942a2ddf21639f020b5489699cf22b46

for i in range(1,65538):
    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)))
```

执行脚本即可得到flag。

已知n（非常大）、e、d，求p、q

由于n特别大，所以没法直接用质因数分解求得 q、p。

例题：题目给出了2个文件，一个是加密的脚本chall.py、另一个是加密脚本的输出内容output.txt。

先分析一下这个加密脚本chall.py：

```
from gmpy2 import invert
from md5 import md5
from secret import p,q

e = 65537
n = p*q
phi = (p-1)(q-1)
d = invert(e, phi)

print n,e,d
print "Flag: flag{%s}" % md5(str(p + q)).hexdigest()
```

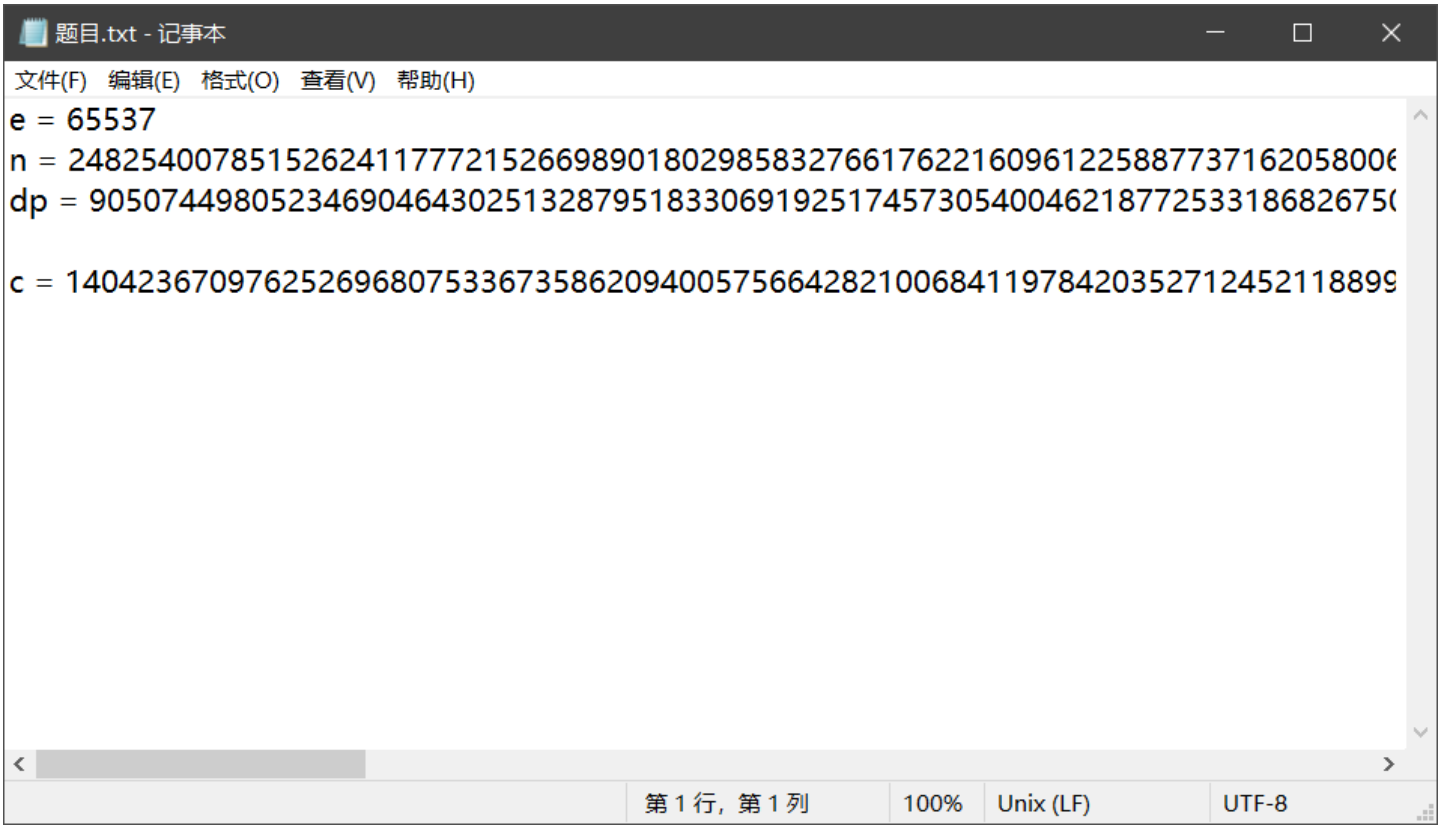
加密脚本真的是很简单啊，flag就是str(p+q)进行md5加密之后的得到的字符串，从output.txt中可以得到n,e,d。现在的关键问题就是求出p和q来，由于n太大了，所以我们不能直接用质因数分解求得 q、p。

我们写如下脚本求 q、p：

```
# 给出n,e,d, 求 q,p
import random
from md5 import md5

def gcd(a, b):
   if a < b:
     a, b = b, a
   while b != 0:
     temp = a % b
     a = b
     b = temp
   return a

def getpq(n,e,d):
 p = 1
 q = 1
 while p==1 and q==1:
  k = d * e - 1
  g = random.randint ( 0 , n )
  while p==1 and q==1 and k % 2 == 0:
   k /= 2
   y = pow(g,k,n)
   if y!=1 and gcd(y-1,n)>1:
    p = gcd(y-1,n)
    q = n/p
 return p,q

def main():
 n = 163525789633723061316424075415670455337666911771383756764919138975924589655440682968131227401265830820065562176162960095164132028336982688456344974789881288503732218535169732590868457258134248505486825038271911215486932887632436190332243226980759876675318632134682236541816580127548975881470274372292690982469698112261298833275980218597248369936263154766993846106808570474034314305257083906956228483153226367853982232074687541976435419585992101272613457709145146701990474350857144036414690162129583619939693045452140615601602677607864821633737844376418082926544893434876134461655429883826877295933848875162726906543094 e = 65537
 d = 945992837997366743013806852805943813909236862533907925328956057798530443506221312139823187583226489445831462957545555348575268564374326665463082995744200877525977631158565401485816534175754728411206188515800688147574055353282657626083943034396073852082236797552864432917266887769620874100764837004552053529804016167540777923930046668161549389269226554229025540867353385301166213495386943263255400823534086480337761035243814626452477071034527343972410708019018291828554742616656180371664408941407838947507210331543263819757818610657662672886902036621407745519455493072557602327492274111594121478960008916675447644945 3
 p,q = getpq(n,e,d)
        print p
        print q
        print "Flag: flag{%s}" %md5(str(p + q)).hexdigest()
if __name__ == '__main__':
 main()
```

已知e、n、dp、密文c，求明文m

[BUUCTF-Crypto]RSA2



这道题也是没有直接给公钥，但是泄露了dp，应该是让我们根据这些条件求出明文m。

编写脚本：

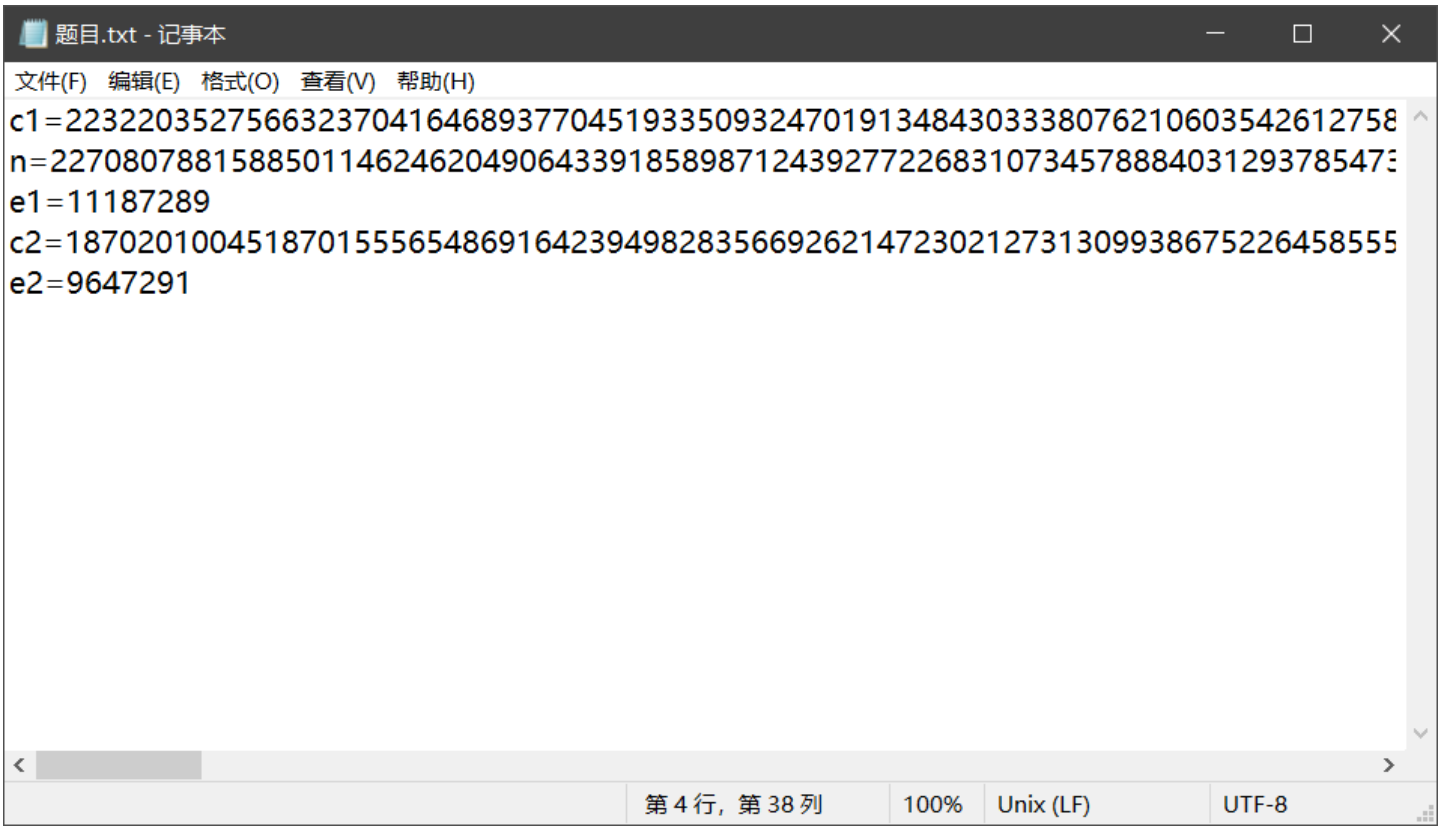
```
#coding=utf-8
import gmpy2 as gp

e = 65537
n = 248254007851526241177721526698901802985832766176221609612258877371620580060433101538328030305219918697643619
814200930679612109885533801335348445023751670478437073055544724280684733298051599167660303645183146161497485358633681492129668802402065797789905550489547645118787266601929429724133167768465309665906113
dp = 905074498052346904643025132879518330691925174573054004621877253318682675055421970943552016695528560364834446303196939207056642927148093290374440210503657

c = 140423670976252696807533673586209400575664282100684119784203527124521188996403826597436883766041879067494280957410201958935737360380801845453829293997433414188838725751796261702622028587211560353362847191060306578510511380965162133472698713063592621028959167072781482562673683090590521214218071160287665180751

for i in range(1, e):  # 在范围(1,e)之间进行遍历
    if (dp * e - 1) % i == 0:
        if n % (((dp * e - 1) // i) + 1) == 0:  # 存在p，使得n能被p整除
            p = ((dp * e - 1) // i) + 1
            q = n // (((dp * e - 1) // i) + 1)
            phi = (q - 1) * (p - 1)  # 欧拉定理
            d = gp.invert(e, phi)  # 求模逆
            m = pow(c, d, n)  # 快速求幂取模运算

print m   # 10进制明文
print '------------'
print hex(m)[2:]  # 16进制明文
print '------------'
print hex(m)[2:].decode('hex')  # 16进制转文本
```

注意：有时得到的明文需要转为十六进制后再转为字符串。

最后得到flag为：flag{wow_leaking_dp_breaks_rsa?_98924743502}。

已知c1、c2、n、e1、e2，求明文m

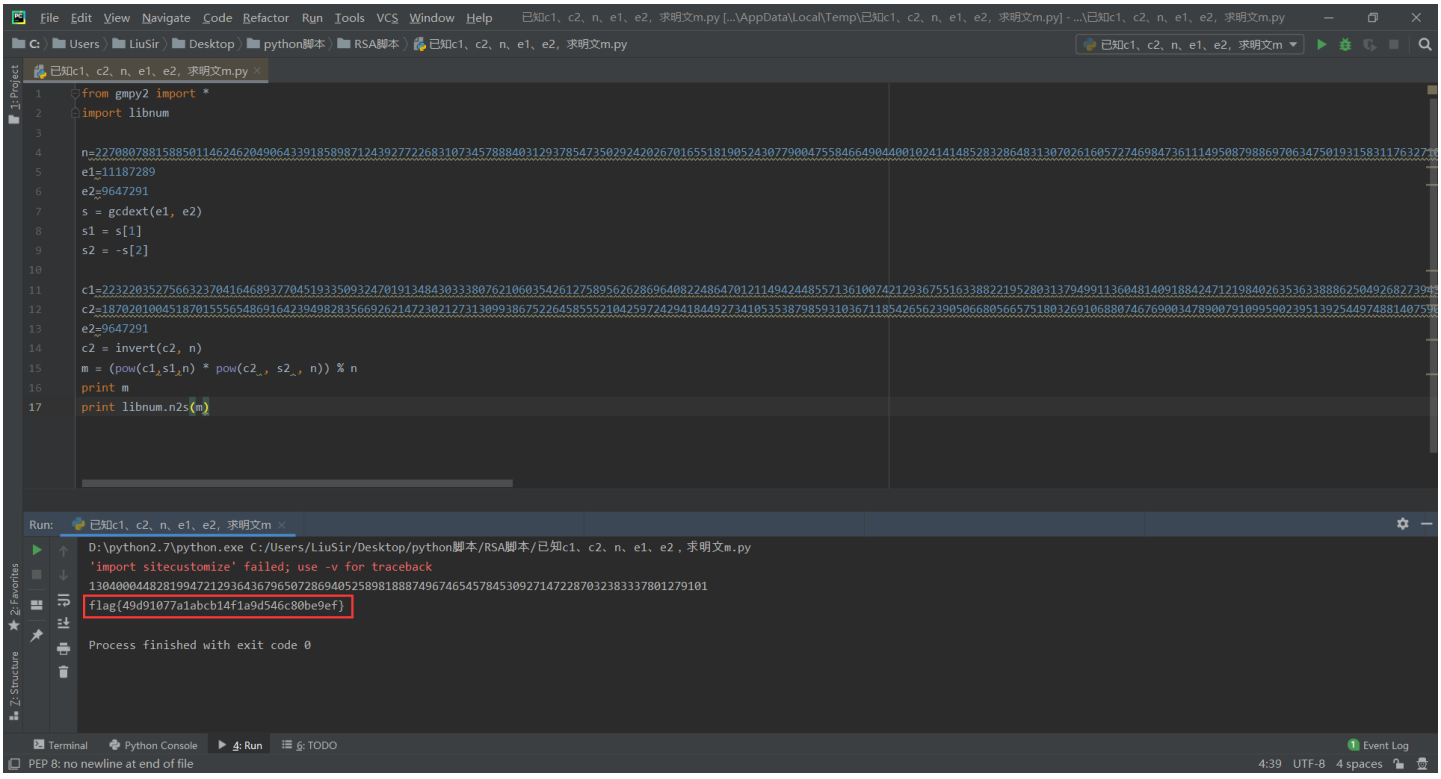
[BUUCTF-Crypto]RSA3



如上图，我们已知c1、c2、e1、e1、n，让我们求解明文m，我们编写如下脚本：

```
from gmpy2 import *
import libnum

n=227080788158850114624620490643391858987124392772268310734578884031293785473502924202670165518190524307790047558466490440010241414852832864831307026160572746984736111495087988697063475019315831176327107007872280164801276773
936499295304165986860273542164225659344590151619276136079028315428579778596125962823536793277733037270044072621972315863245991819835726224045903540845417880622621645101406058681224103880901744201477524085541297897609023008980462739090078528184740307706996476473630151021189567376739413542176926960449696953085064365731425655734875835070373569448480398643823392162666706735674888715089253111548 01
e1=11187289
e2=9647291
s = gcdext(e1, e2)
s1 = s[1]
s2 = -s[2]

c1=223220352756632370416468937704519335093247019134843033380762106035426127589562628696408224864701211494244855713610074212936755163388221952803137949911360481409188424712198402635363388862504926827394364100134366511617207258554848666900847887213495556620198790815011132229961233055330093259643777988927031615218528059568112195638833128963301562986216746843539195475581279209257068428089147621990110549558165349776752673950095753478203870734839284250665363614827748923709695207403042874565555089333727823275065690107725374975417643114290522162911989320926177926452539014789108015928782035648611189120454649598322566051361
c2=187020100451870155565486916423949828356692621472302127313099386752264585552104259724294184492734105353879859310367118542656239050668056657518032691068807467690034789007910995902395139254497488140759040174715855728484735564905654500626647064491284158347879619472662597897859629222387011340797204142284140661930714953046123410529874556159300235368238014992697733571860874527475008406404193650115544211830375056534612867327409837027408226711480456194976671845861236572856040618756539095678223289140653377977334446403515187754876498199782623636172657979828431796308887294072384966509877204287082171152579890078673316983971
e2=9647291
c2 = invert(c2, n)
m = (pow(c1,s1,n) * pow(c2 , s2 , n)) % n
print m
print libnum.n2s(m)
```

如下图，执行后得到flag：



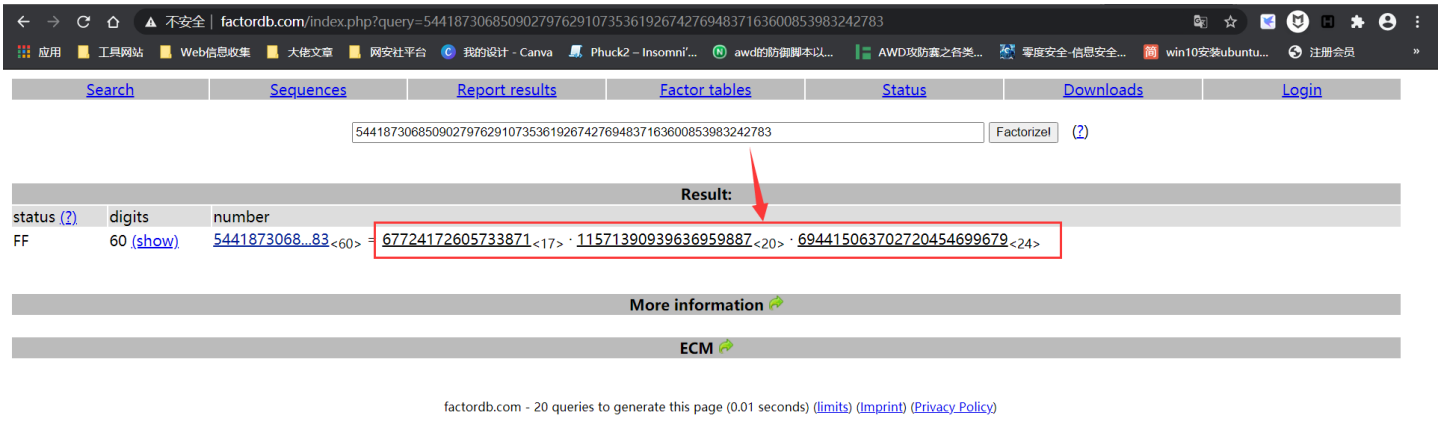
n分解出多个不同的因子时，求明文m

给出：

```
n= 544187306850902797629107353619267427694837163600853983242783
e= 39293
c= 439254895818320413408827022398053685867343267971712332011972
```

这里n不是很大，可以对其进行质因数分解。

我们对n在这个分解网站（http://www.factordb.com/index.php）进行质因数分解，得到了3个质因数：



写出如下脚本对密文c进行解密得到明文m：

```
import gmpy2
from Crypto.Util.number import long_to_bytes

n= 544187306850902797629107353619267427694837163600853983242783
e= 39293
c= 439254895818320413408827022398053685867343267971712332011972
p1 = 67724172605733871
p2 = 11571390939636959887
p3 = 694415063702720454699679
phi = (p1-1)*(p2-1)*(p3-1)
d = gmpy2.invert(e, phi)
m = pow(c, d, n)
print long_to_bytes(m)
```

执行脚本后，即可得到flag。

已知 密文文件 和 公钥文件，求明文m

要用到RsaCtfTool工具，教程：https://www.jianshu.com/p/c945b0f0de0a

密文文件可能是以下几种：

- flag.enc
- cipher.bin
- flag.b64
- …

公钥文件可能是以下几种：

- pubkey.pem
- key.pem
- key.pub
- pub.key
- …

方法一：(key.pem 和 cipher.bin)

攻防世界-wtc_rsa_bbq

下载附件，得到一个没有后缀的cry200文件，用winhex一看是一个zip，修改后缀为zip后解压，得到两个文件，key.pem公钥文件和cipher.bin密文文件：

应该是让我们先破解得到私钥，然后再用私钥破解明文，丢kali里面直接用 RsaCtfTool 进行破解明文即可（这一过程 RsaCtfTool 将自动求解私钥）：

```
python3 RsaCtfTool.py --publickey key.pem --uncipherfile cipher.bin
python3 RsaCtfTool.py --publickey 公钥文件 --uncipherfile 加密的文件
```

如上图得到flag。

方法二：（flag.b64 和 key.pub）

攻防世界-cr4-poor-rsa

下载题目附件，下载附件，同样得到一个没有后缀的文件，在kali上用file命令一看，发现是一个rar压缩包：

```
root@kali:~/桌面# file bf930316910b451c94c41ce8a9d851a8
bf930316910b451c94c41ce8a9d851a8: POSIX tar archive (GNU)
root@kali:~/桌面#
```

修改后缀为rar后解压，得到两个文件，flag.b64密文文件和 key.pub 公钥文件：

名称	修改日期	类型	大小
flag.b64	2016/12/11 17:08	B64 文件	1 KB
key.pub	2016/12/11 16:59	pub	1 KB

这个flag.b64的名字有点奇怪，会不会是里面的内容被base64加密了，所以我们先处理flag.b64，将flag.b64中的内容进行解 base64操作。使用 notepad++ 打开 flag.b64文件使用 插件中的 MIME Tools 中的 base64 decode 将文件内容解密，然后保存。

然后使用 RsaCtfTool 工具进行破解：

```
python3 RsaCtfTool.py --publickey key.pub --uncipherfile flag.b64
```

```
root@kali:~/RsaCtfTool# python3 RsaCtfTool.py --publickey /root/rsa/key.pub --uncipherfile /root/rsa/flag.b64
private argument is not set, the private key will not be displayed, even if recovered.
[*] Testing key /root/rsa/key.pub.
Can't load ecm because sage is not installed
Can't load ecm2 because sage is not installed
Can't load roca because sage is not installed
Can't load boneh_durfee because sage is not installed
Can't load qicheng because sage is not installed
Can't load binary_polinomial_factoring because sage is not installed
Can't load smallfraction because sage is not installed
[*] Performing noveltyprimes attack on /root/rsa/key.pub.
[*] Performing factordb attack on /root/rsa/key.pub.

Results for /root/rsa/key.pub:

Unciphered data :
HEX : 0x{c.hex()}
INT (big endian) : 1031458070139789855403559101256404874003859276715744454776...7088604549057047024077061970525055104470889482508265382 50
INT (little endian) : 1058048920308414954548777223739928508659495044479712523241614238320161223051470182905704952143926276316183744709183411 20
STR : b'\x00\x02\x9e8\xded\xa0\x96#H\x8au6L\xdb\xae\x84\x89:\x...0ALEXCTF{SMALL_PRIMES_ARE_BAD}\n'
root@kali:~/RsaCtfTool#
```

已知私钥文件 private.pem 和密文文件 flag.enc，求明文m

这个简单了，直接用RsaCtfTool进行破解即可：

```
python3 RsaCtfTool.py --private private.pem --uncipherfile flag.enc
```

提取私钥文件中的信息

这个简单，直接用RsaCtfTool即可提取即可：

```
python3 RsaCtfTool.py --key private.pem --dumpkey
```

解码公钥文件

我们得到的公钥文件（pubkey.pem等）里面都是加密的，像这样：

光这样我们是看不到公钥里面的e和n的，我们可以用以下两种方法对公钥文件进行解码：

1. 用RsaCtfTool.py工具：

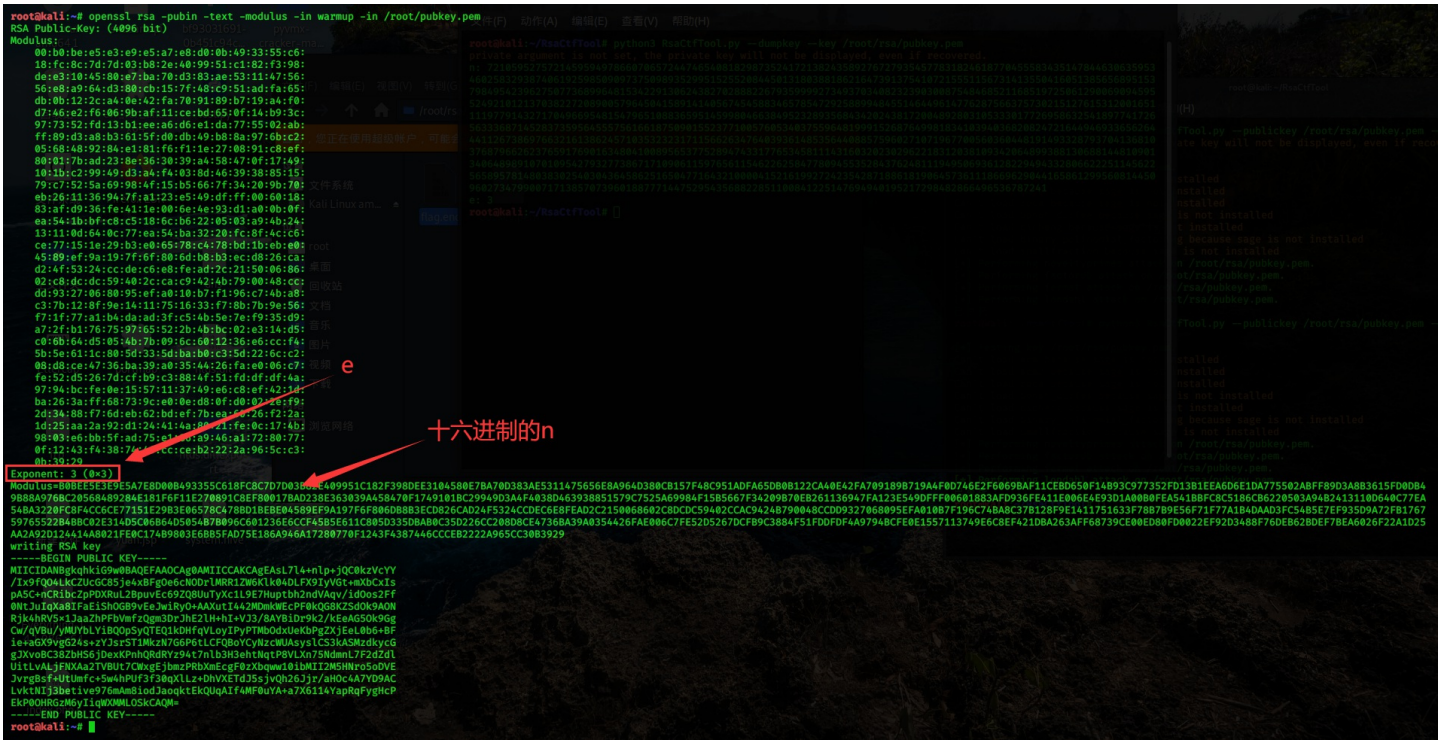
即把pubkey.pem、key.pub等格式的公钥文件转换为n，e

```
python3 RsaCtfTool.py --dumpkey --key 公钥文件
```

2. 利用 openssl 找出指数 e 和模数 n

openssl工具kali自带了。

```
openssl rsa -pubin -text -modulus -in warmup -in pubkey.pem
```



利用 公钥文件 生成 私钥文件（已知公钥求私钥）

这个简单，直接用RsaCtfTool生成即可：

```
python3 RsaCtfTool.py --publickey pubkey.pem --private > private.pem
python3 RsaCtfTool.py --publickey pub.key --private > private.key
......
```

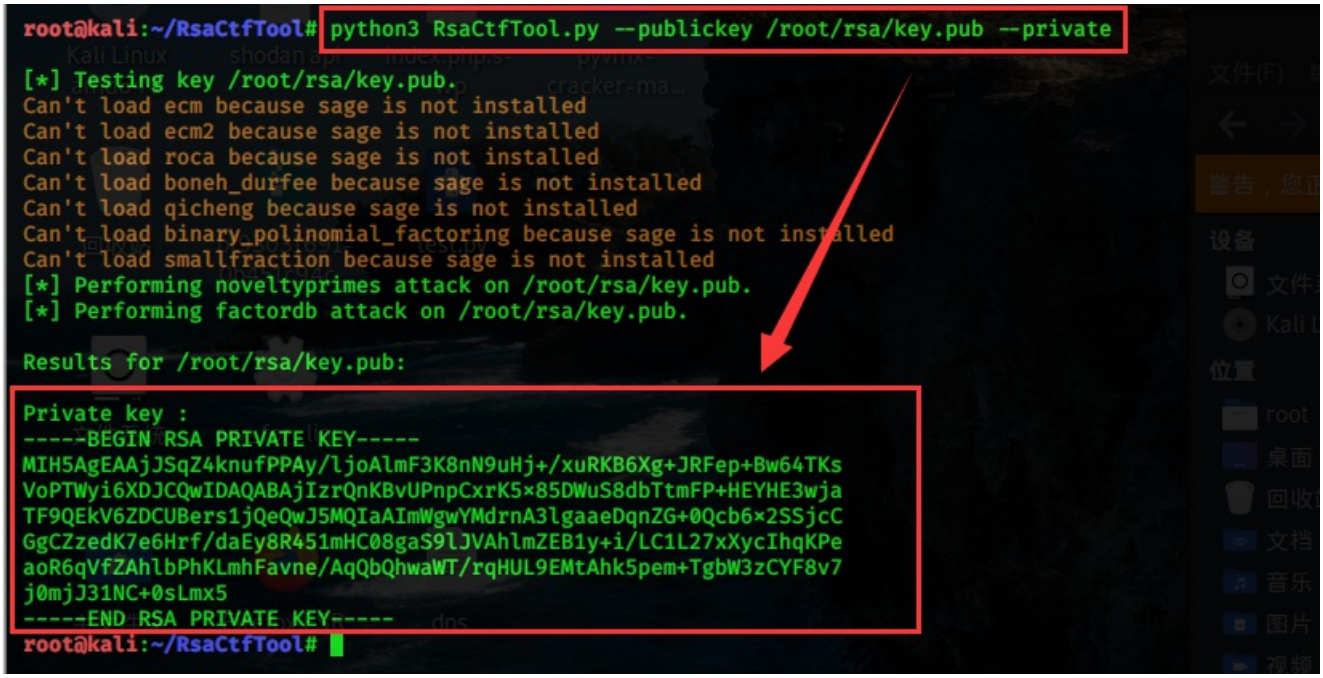
例题：还是上面方法二里的攻防世界-cr4-poor-rsa。

该题由于给出了公钥文件，所以我们可以先直接利用 公钥文件 生成 私钥文件，即从公钥求私钥，然后就可以用私钥直接对密文文件进行解密了。

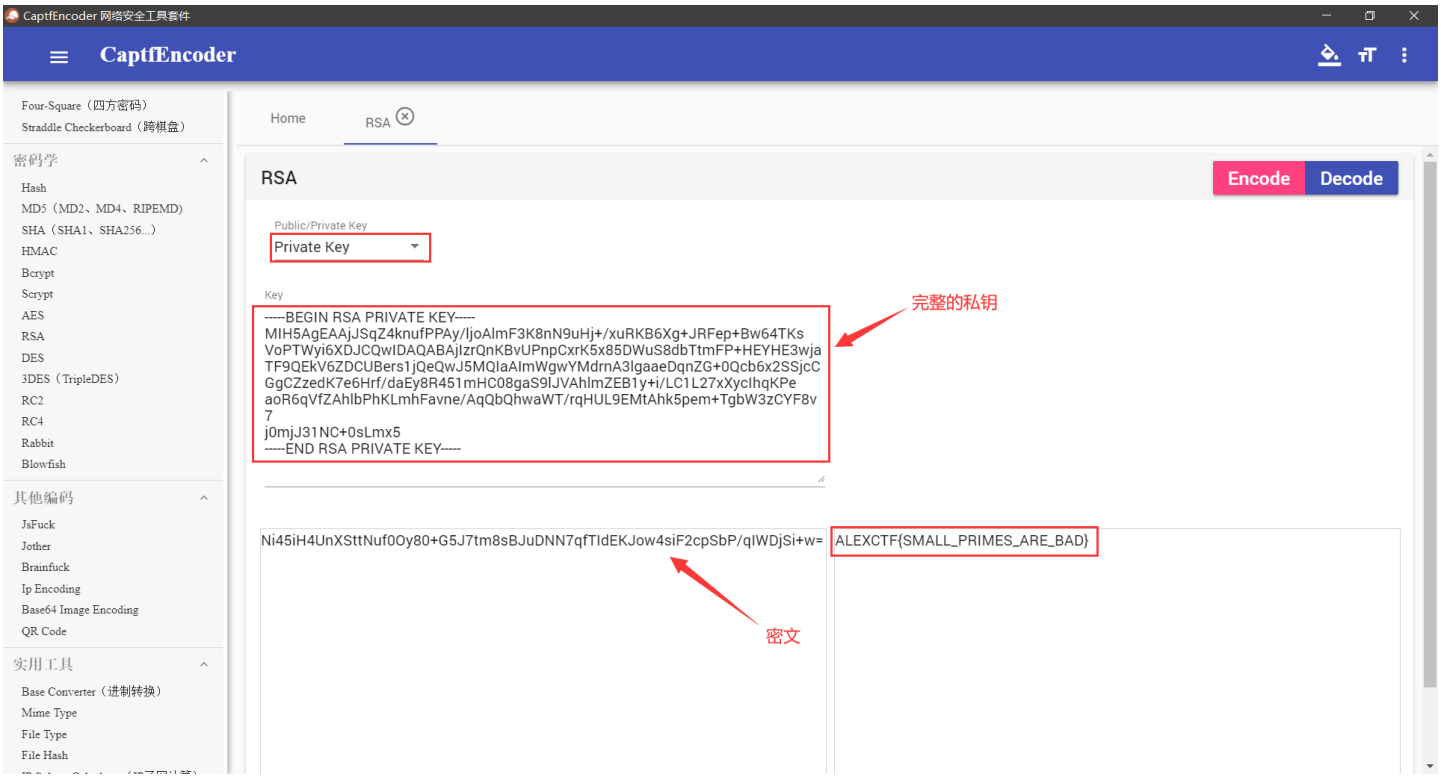
先从公钥生成私钥：

```
python3 RsaCtfTool.py --publickey key.pub --private
```

如下图得到了私钥：



然后将私钥和密文文件丢到CaptfEncoder工具里去解密，注意这里的文本要为未解密之前的base64编码：



爆破攻击方法

低加密指数分解攻击（比如 e=2,e=3）

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 e=2,e=3），但是选取不当的话，就会造成安全问题。

下面就是e选取的太小导致存在的安全问题：

（1）e=2，可以把密文c开平方求解

RSA加密，当e等于2时，相当于把明文m平方而已，得到的c也比n小很多。尝试把c开根号看能否得到明文。一般的python开根号方法精度较低，对大整数开出来的根号准确度低。

发现使用gmpy2库可以对大整数开根号。

例题：西湖论剑rsa

已知如下：

```
e=2

c=92179799413662202753778750958617109252070285517715206103872387348197592562230801756030321676580866698866613029629850463488651817405912513219666828485363315832435 29
```

让我们求明文m。

编写脚本进行开根号求解：

```
import gmpy2
import libnum
c = 92179799413662202753778750958617109252070285517715206103872387348197592562230801756030321676580866698866613029629850463488651817405912513219666828485363315832435 29
m = gmpy2.isqrt(c)
m = int(m)
m_text = libnum.n2s(m)  #将十六进制转为字符
print(m_text)
```

执行完后便可以得到flag为：flag1{Th1s_i5_wHat_You_ne3d_FirsT}。

（2）e=3，可进行小明文攻击

适用情况：e较小，一般为3。

当如果公钥e很小，明文m也不大的话，就会导致 `m^e=k*n+c` 中的k值很小甚至为0，爆破k或直接开三次方即可。

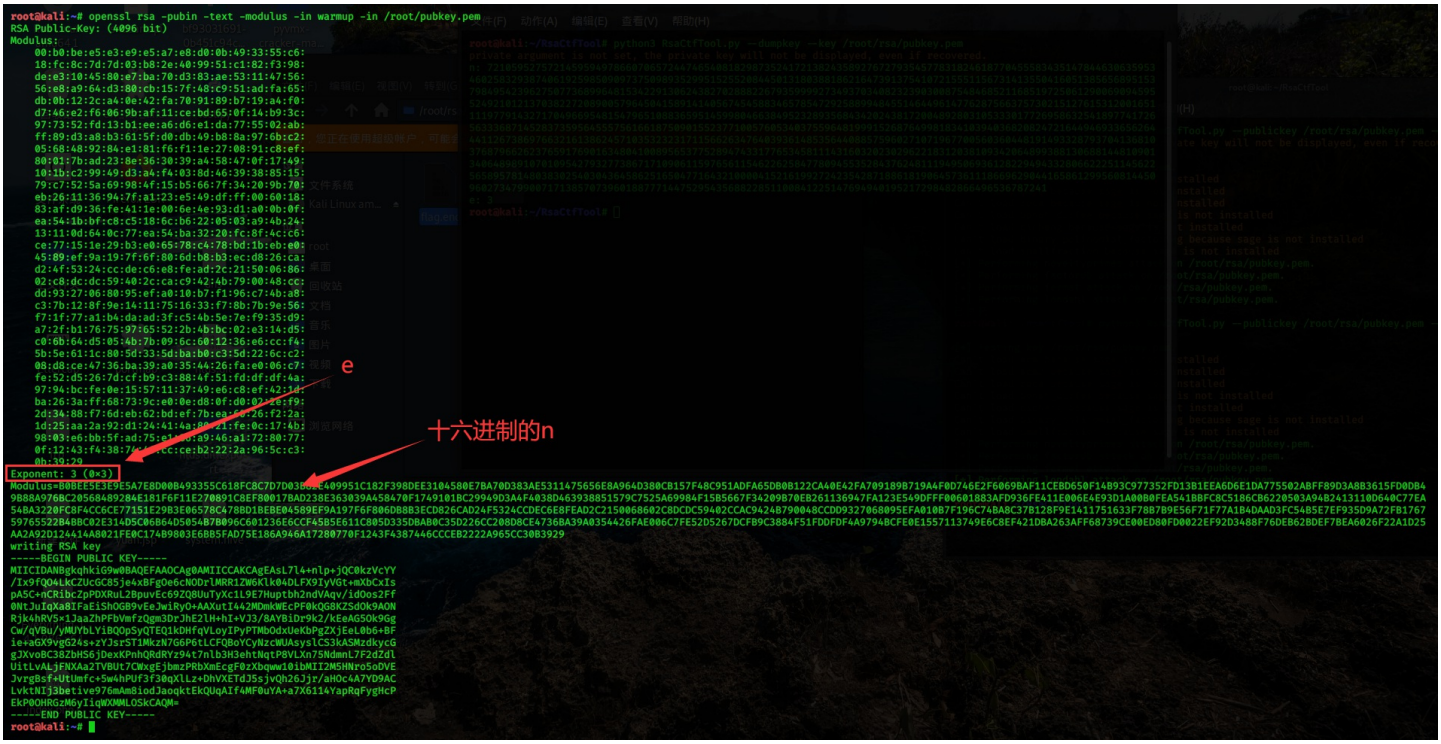
例题：Jarvis OJ -Crypto-Extremely hard RSA

下载附件，题目给出了两个文件， flag.enc 密文文件 和 pubkey.pem 公钥文件文件

当然这个题我们可以先用RsaCtfTool进行破解：

```
python3 RsaCtfTool.py --publickey pubkey.pem --uncipherfile  flag.enc
```

但是破解失败了，所以换一种思路，先用openssl或RsaCtfTool解码公钥：



从中发现e=3，很小，所以很可能存在小名文攻击。

可以假设，k为0，然后将c直接开三次方就可以得到明文 m 了。注意：密文c要从 flag.enc 密文文件中以十六进制方式来读取，因为文件中含有很多不可打印字符。

编写脚本：

```
import gmpy2,binascii,libnum,time
n = 0xB0BEE5E3E9E5A7E8D00B493355C618FC8C7D7D03B82E409951C182F398DEE3104580E7BA70D383AE5311475656E8A964D380CB157F
48C951ADFA65DB0B122CA40E42FA709189B719A4F0D746E2F6069BAF11CEBD650F14B93C977352FD13B1EEA6D6E1DA775502ABFF89D3A8B3
615FD0DB49B88A976BC20568489284E181F6F11E270891C8EF80017BAD238E363039A458470F1749101BC29949D3A4F4038D463938851579
C7525A69984F15B5667F34209B70EB261136947FA123E549DFFF00601883AFD936FE411E006E4E93D1A00B0FEA541BBFC8C5186CB6220503
A94B2413110D640C77EA54BA3220FC8F4CC6CE77151E29B3E06578C478BD1BEBE04589EF9A197F6F806DB8B3ECD826CAD24F5324CCDEC6E8
FEAD2C2150068602C8DCDC59402CCAC9424B790048CCDD9327068095EFA010B7F196C74BA8C37B128F9E1411751633F78B7B9E56F71F77A1
B4DAAD3FC54B5E7EF935D9A72FB176759765522B4BBC02E314D5C06B64D5054B7B096C601236E6CCF45B5E611C805D335DBAB0C35D226CC2
08D8CE4736BA39A0354426FAE006C7FE52D5267DCFB9C3884F51FDDFDF4A9794BCFE0E1557113749E6C8EF421DBA263AFF68739CE00ED80F
D0022EF92D3488F76DEB62BDEF7BEA6026F22A1D25AA2A92D124414A8021FE0C174B9803E6BB5FAD75E186A946A17280770F1243F4387446
CCCEB2222A965CC30B3929     # 十六进制的n

e = 3
res = 0
c = int(open('flag.enc','rb').read().encode('hex'),16)
print time.asctime
for i in xrange(200000000):
    if gmpy2.iroot(c+n*i,3)[1] == 1:
        res = gmpy2.iroot(c+n*i,3)[0]
        print i,res
        print libnum,n2s(res)
        print time.asctime()
        break
```

破电脑跑了快半个小时，期间以为卡了，还好跑出来了。

低解密指数攻击（e过大或过小）

适用情况：e过大或过小，一般e过大时使用。

在RSA中d也称为解密指数，当d比较小的时候，e也就显得特别大了。在e过大或过小的情况下，可使用算法从e中快速推断出d的值，进而求出m。（具体我也没看懂）

一般我们看到e非常大，就应该下意识的想到是使用低解密指数攻击，也叫RSA维纳攻击(RSA wiener attack)。

例题：我不是个人

题目给出如下：

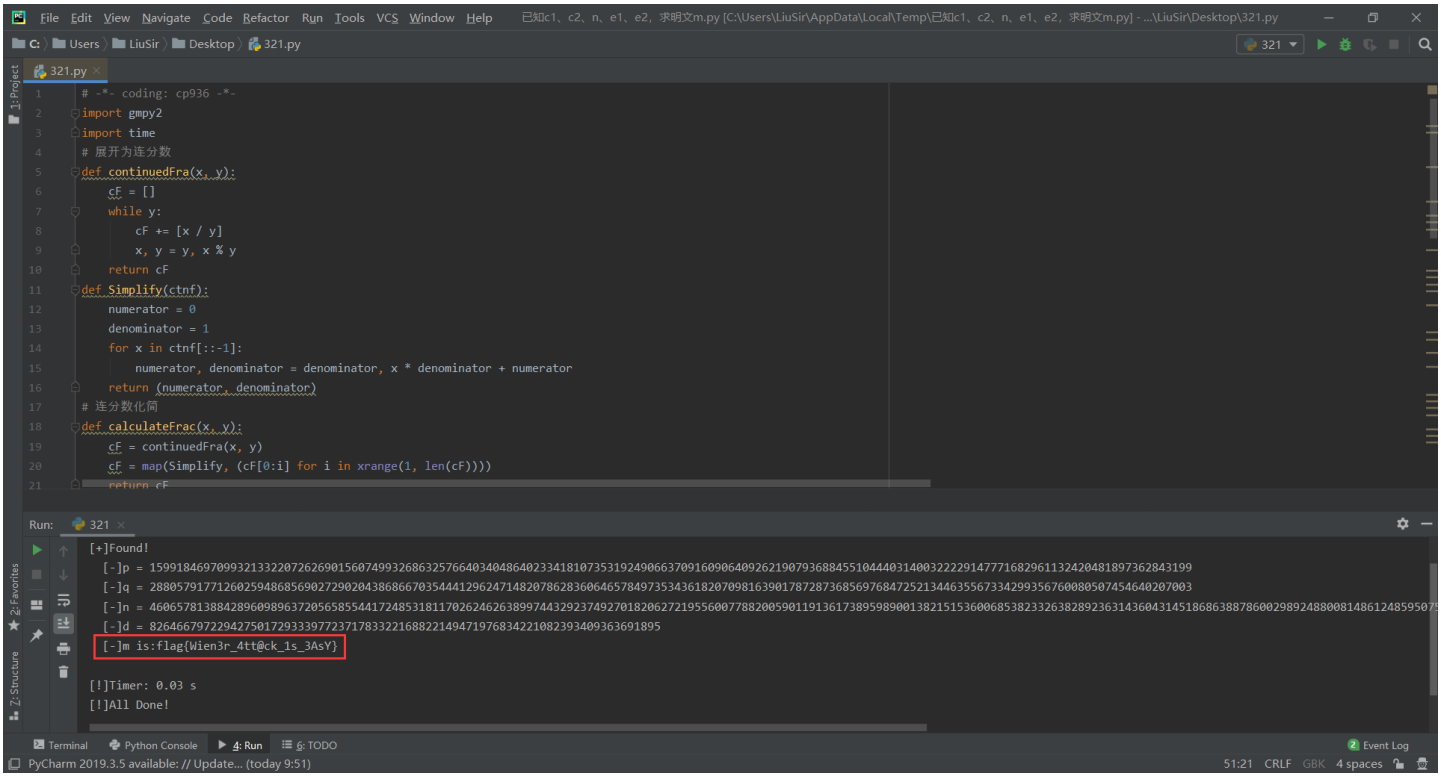
```
n = 46065781388428960989637205658554417248531811702624626389974432923749270182062721955600778820059011913617389598900138215153600685382332638289236314360431451868638878600298924880081486124859507532627709964533869497709745916853089877600729369572810197606942397169652423775522718706141820284991147912479399072259 7

e = 35461110244130757205657218182792589919834535022875373093108939327546391654445662689424541509610783446577840953237318712531855461472259930179152891621283936812106603554100880826153450058602365276771227162578520428096468800468032830012484968047710530251937737009257810782711682139182621097232037761496754782761 9

c = 38230991316229399651823567590692301060044620412191737764632384680546256228451518238842965221394711848337832459443844446889468362154188214840736744657885858943810177675871991111466653158257191139605699916347308294995664530280816850482740530602254559123759121106338359220242637775919026933563326069449424391192
```

让我们求明文。

编写如下脚本解密：

```
# -*- 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 = 46065781388428960989637205658554417248531811702624626389974432923749270182062721955600778820059011913617389598900138215153600685382332638289236314360431451868638878600298924880081486124859507532627709964533869497709745916853089877600729369572810197606942397169652423775522718706141820284991147912479399072259 7

e = 35461110244130757205657218182792589919834535022875373093108939327546391654445662689424541509610783446577840953237318712531855461472259930179152891621283936812106603554100880826153450058602365276771227162578520428096468800468032830012484968047710530251937737009257810782711682139182621097232037761496754782761 9

c = 38230991316229399651823567590692301060044620412191737764632384680546256228451518238842965221394711848337832459443844446889468362154188214840736744657885858943810177675871991111466653158257191139605699916347308294995664530280816850482740530602254559123759121106338359220242637775919026933563326069449424391192

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!'
```

如下图，执行后即可得到flag：

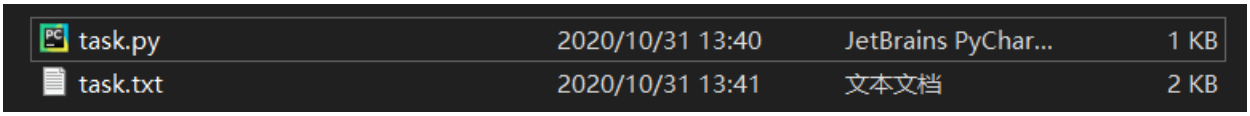


多重基数的RSA低解密指数攻击

[2020祥云杯]simplersa

这里与之前的低解密指数攻击不同，这是多重基数的，即 `phi = (p - 1) * (q - 1) * (r - 1)`。

下载附件后解压得到：



teask.py内容如下：

```
from Crypto.Util.number import *
import gmpy2

p, q, r = [getPrime(512) for i in range(3)]
n = p * q * r
phi = (p - 1) * (q - 1) * (r - 1)
d = getPrime(256)
e = gmpy2.invert(d , phi)

flag = b"flag{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}"

c = pow(bytes_to_long(flag), e, n)

print(e, n)
print(c)
```

task.txt：

```
107229542594413650703993867710144248121351940812514823388044284920635337968198930500057038709315223626320339572697469295981931541078118009421620910006953079140749551088264078192056473221432789809994479271425362204787315263043806015164460178684368374625640792570970216356514100435623887940638556658670422614853786381171729896660731474773755172437951667537663477145588397606900713421898243517016064784854941228912898207064783277444634506248937409267316961883670167 9   // e

182722199269284917924406983427381656571427650530524610343596288746152038170973992722305523995396518245125219476893570262805658703417380060582742404328167318360647873618992737774557537990887645648501683241680602925497276961739356023849432607894084229515302928539449178371238499012510077459647706448228082940785601483523171178899006667653441441474106775956410233161466671379707381124509951213052860046409949273467168908499003607786004223845490896084159510712293317 3   // n

107992917411082049405935541505910422990526876308915777137465793264671101748870153646068731964836254956331312526806972241214802388562696264091585231729791642172581807781423729280721895257411114191815839119062136250886284293294578305918195261431728911640587874175891335169790528999365110596816919321124214499143471555295234079154532327006576352986501032619282433468441321235770827525909620250904283808115005572765044388743825396460741494424587790400258099786630045 2   // c
```

一看e这么大，下意识就想到了低解密指数攻击（维纳攻击），但这里与之前的不同，这是多重基数的，即 `phi = (p - 1) * (q - 1) * (r - 1)`。

在网上找到一个解密脚本：

```
#coding:utf-8
from Cryptodome.Util.number import long_to_bytes
e = 10722954259441365070399386771014424812135194081251482338804428492063533796819893050005703870931522362632033957269746929598193154107811800942162091000695307914074955108826407819205647322143278980999447927142536220478731526304380601516446017868436837462564079257097021635651410043562388794063855665867042261485378638117172989666073147477375517243795166753766347714558839760690071342189824351701606478485494122891289820706478327744463450624893740926731696188367016791
n = 18272219926928491792440698342738165657142765053052461034359628874615203817097399272230552399539651824512521947689357026280565870341738006058274240432816731836064787361899273777455753799088764564850168324168060292549727696173935602384943260789408422951530292853944917837123849901251007745964770644822808294078560148352317117889900666765344144147410677595641023316146667137970738112450995121305286004640994927346716890849900360778600422384549089608415951071229339173
c = 10799291741108204940593554150591042299052687630891577713746579326467110174887015364606873196483625495633131252680697224121480238856269626409158523172979164217258180778142372928072189525741111419181583911906213625088628429324578305918195261431728911640587874175891335169790528999365110596816919321124214499143471555295234079154532327006576352986501032619282433468441321235770827525909620250904283808115005572765044388743825396460741494424587790400258099786630045 2
#连分数展开算法
def lf(x,y):
 arr=[]
 while y:
  arr+=[x//y]
  x,y=y,x%y
 return arr
#渐进分数算法
def jj(k):
 x=0
 y=1
 for i in k[::-1]:
  x,y=y,x+i*y
 return (y,x)
data=lf(e,n)

for x in range(1,len(data)+1):
 data1=data[:x]
 d = jj(data1)[1]
 m = pow(c,d,n)
 flag = long_to_bytes(m)
 if b'flag{' in flag:
  print(flag)
  break
```

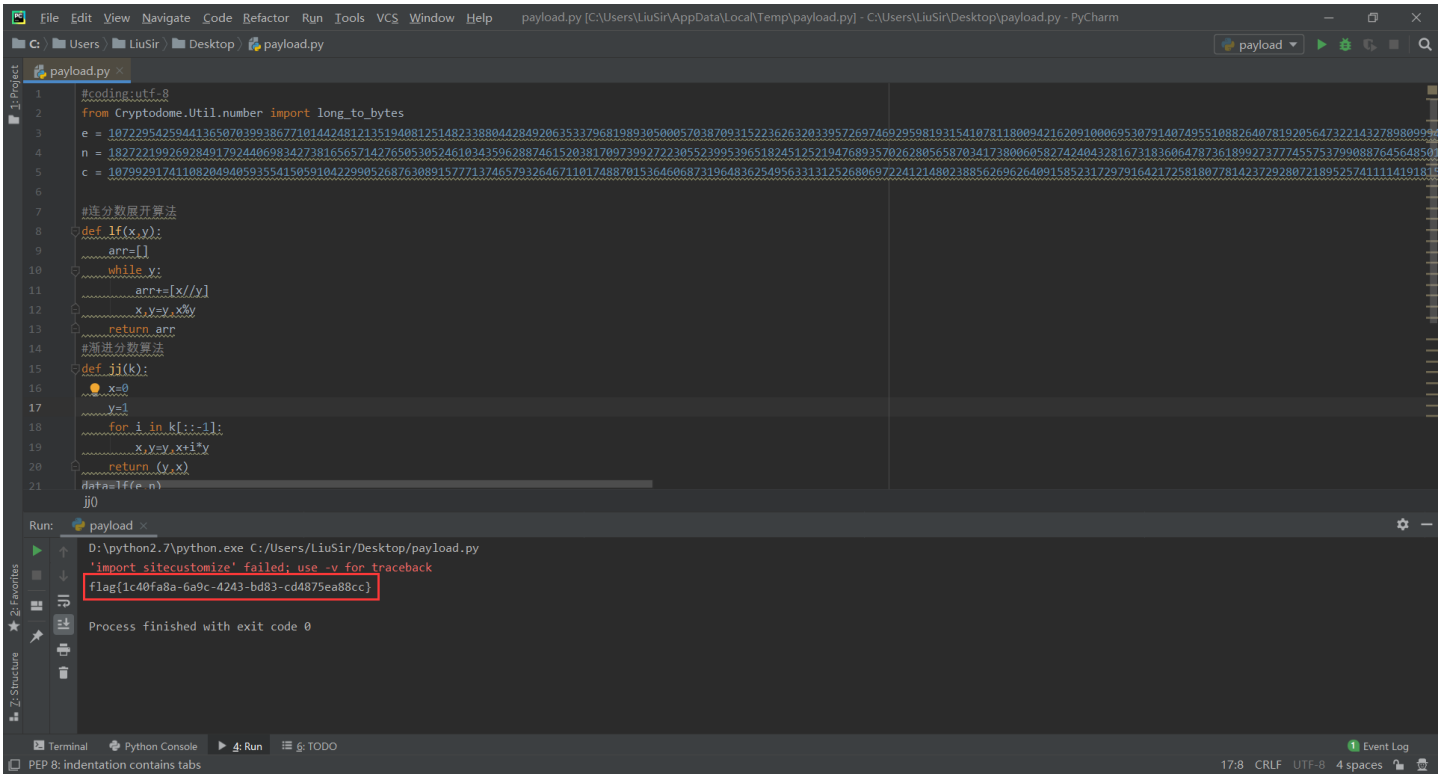
解密脚本2：

```
#coding:utf-8
import gmpy2
from Crypto.Util.number import *

def transform(x,y):       #使用辗转相处将分数 x/y 转为连分数的形式
    res=[]
    while y:
        res.append(x//y)
        x,y=y,x%y
    return res

def continued_fraction(sub_res):
    numerator,denominator=1,0
    for i in sub_res[::-1]:      #从sublist的后面往前循环
        denominator,numerator=numerator,i*numerator+denominator
```

```
        denominator,numerator=numerator,i*numerator+denominator
    return denominator,numerator    #得到渐进分数的分母和分子，并返回


#求解每个渐进分数
def sub_fraction(x,y):
    res=transform(x,y)
    res=list(map(continued_fraction,(res[0:i] for i in range(1,len(res)))))  #将连分数的结果逐一截取以求渐进分数
    return res

#以上是获得e/n的连分数

def get_pq(a,b,c):      #由p+q和pq的值通过维达定理来求解p和q
    par=gmpy2.isqrt(b*b-4*a*c)   #由上述可得，开根号一定是整数，因为有解
    x1,x2=(-b+par)//(2*a),(-b-par)//(2*a)
    return x1,x2

def wienerAttack(e,n):
    for (d,k) in sub_fraction(e,n):  #用一个for循环来注意试探e/n的连续函数的渐进分数，直到找到一个满足条件的渐进分数
        #if k==0:                     #可能会出现连分数的第一个为0的情况，排除
            #continue
        #if (e*d-1)%k!=0:             #ed=1 (\pmod φ(n)) 因此如果找到了d的话，(ed-1)会整除φ(n),也就是存在k使得(e*d-1)
//k=φ(n)
            #continue
        if 250<=d.bit_length()<=256:
            print(d)
            global c
            print(long_to_bytes(pow(c,d,n)))
        else:
            continue
        phi=(e*d-1)//k               #这个结果就是 φ(n)
        px,qy=get_pq(1,n-phi+1,n)
        if px*qy==n:
            p,q=abs(int(px)),abs(int(qy))     #可能会得到两个负数，负负得正未尝不会出现
            d=gmpy2.invert(e,(p-1)*(q-1))     #求ed=1 (\pmod  φ(n))的结果，也就是e关于 φ(n)的乘法逆元d
            return d
    print("该方法不适用")


e=10722954259441365070399386771014424812135194081251482338804428492063533796819893050005703870931522362632033957
2697469295981931541078118009421620910006953079140749551088264078192056473221432789809994479271425362204787315263
0438060151644601786843683746256407925709702163565141004356238879406385566586704226148537863811717298966607314747
7375517243795166753766347714558839760690071342189824351701606478485494122891289820706478327744463450624893740926
73169618836701679
n=18272219926928491792440698342738165657142765053052461034359628874615203817097399272230552399539651824512521947
6893570262805658703417380060582742404328167318360647873618992737774557537990887645648501683241680602925497276961
7393560238494326078940842295153029285394491783712384990125100774596477064482280829407856014835231711788990066676
5344144147410677595641023316146667137970738112450995121305286004640994927346716890849900360778600422384549089608
41595107122933173
c=10799291741108204940593554150591042299052687630891577713746579326467110174887015364606873196483625495633131252
6806972241214802388562696264091585231729791642172581807781423729280721895257411114191815839119062136250886284293
2945783059181952614317289116405878741758913351697905289993651105968169193211242144991434715552952340791545323270
0657635298650103261928243346844132123577082752590962025090428380811500557276504438874382539646074149442458779040
02580997866300452
d = wienerAttack(e,n)
print("d=",d)
```

（这两个脚本也可以用来解正常的维纳攻击，但之前那个脚本不能用来解多重基数的RSA维纳攻击）

执行后得到flag：



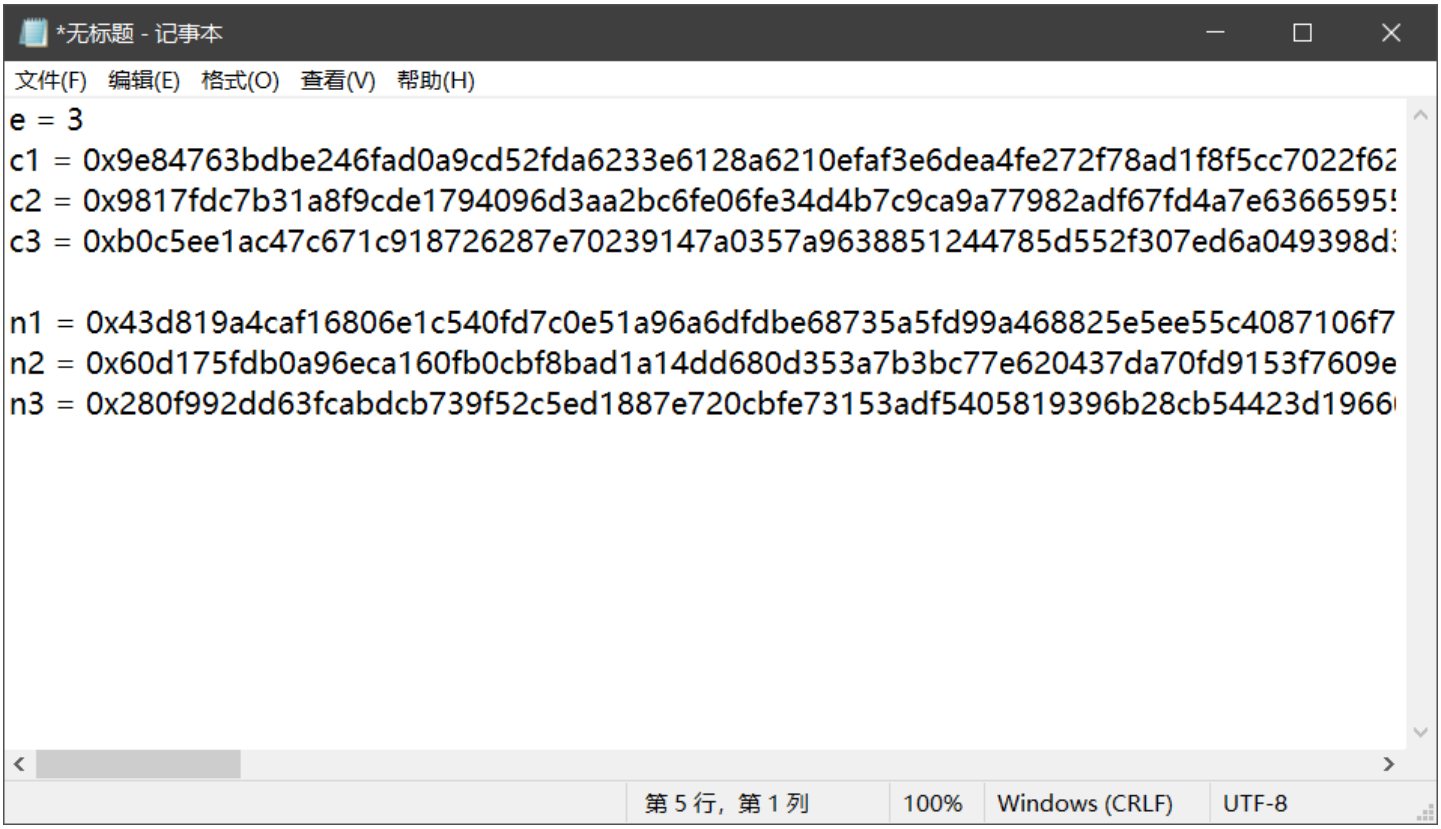
低加密指数广播攻击（n、c不同，e和m相同）

使用条件：模数n、密文c不同，明文m、加密指数e相同

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

例题1：Jarvis OJ -2018强网杯nextrsa-Level9

题目给出如下：



即相同的加密指数e，但是模数n和密文c不同，很明显使用低加密指数广播攻击。

解题脚本1，如下，执行即可得到明文m：

```python
#!/usr/bin/python
#coding:utf-8

import random
from gmpy2 import invert, iroot
import libnum

def broadcast(n1, n2 ,n3, c1, c2, c3):
    n = [n1, n2, n3]
    C = [c1, c2, c3]
    N = 1
    for i in n:
        N *= i

    Ni = []
    for i in n:
        Ni.append(N / i)

    T = []
    for i in xrange(3):
        T.append(long(invert(Ni[i], n[i])))

    X = 0
    for i in xrange(3):
        X += C[i] * Ni[i] * T[i]

    m3 = X % N
    m = iroot(m3, 3)
    return m[0]
```

```
def main():
    e = 3

    c1 = 0x9e84763bdbe246fad0a9cd52fda6233e6128a6210efaf3e6dea4fe272f78ad1f8f5cc7022f62f4f542341128e42d6fd10e67c
5f96edbd243917c0151289f7228e44019b8c65a541d7306b398465e26b69cab36cc61e4ac094832b4299bbaf4630b722a0fb4f1997053be9
7e926f94afb55a0bb6ef00ab694e2f595d9eb8ca96c49f5cbbe194529f68a1aaf6f5151484b471285ba8fc8cd30b55612f35a74dc68e255c
363579a80d27ce5090873ac719ba59f2492c91fd28bcce26b6a02bae005cbbd2a4cfe5b93442be8664d2313d412e7e09f545c64b7b74bbc4
08b6e574d0d300135cba8d6c1d73737d59baca9992ede644d856eb4cfcda562a75743e4b491
    c2 = 0x9817fdc7b31a8f9cde1794096d3aa2bc6fe06fe34d4b7c9ca9a77982adf67fd4a7e636659553f4168a16757dc3a75e54ff850
b9a94a5270f4f75502c7055a3a389df2ea6b00784a4e78e66901b427253c0f343f127e0ff162a349bb14eb4c1453fc6daace19bba4940d77
c435686ef3b59f732072cde2e148d1a64f9682b3f1ceb9a000d87e180a1f9eb20c59dbebc13ddb2e07b64db89217f40369aeec878a45d999
09ab2a3e4cdb74aa68890c941315ae289d6667200c53f9a32c8a64bfc74e62898ac03c460f945a13f11ee28860a3cd07526c30aa92eb8944
2a76549fe4ed8a43d14fdeeb350e90443a3a586db719f8610eb5d4a8f5bd1e481b5ef6e96ef
    c3 = 0xb0c5ee1ac47c671c918726287e70239147a0357a9638851244785d552f307ed6a049398d3e6f8ed373b3696cfbd0bce1ba88d
152f48d4cea82cd5dafd50b9843e3fa2155ec7dd4c996edde630987806202e45821ad6622935393cd996968fc5e251aa3539ed593fe893b1
5d21ecbe6893eba7fe77b9be935ca0aeaf2ec53df7c7086349eb12792aefb7d34c31c18f3cd7fb68e8a432652ef76096096e1a5d7ace90a2
82facf2d2760e6b5d98f0c70b23a6db654d10085be9dcc670625646a153b52c6c710efe8eb876289870bdd69cb7b45813e4fcfce815d1918
38926e9d60dd58be73565cff0e10f4e80122e077a5ee720caedc1617bf6a0bb072bbd2dab0

    n1 = 0x43d819a4caf16806e1c540fd7c0e51a96a6dfdbe68735a5fd99a468825e5ee55c4087106f7d1f91e10d50df1f2082f0f32bb8
2f398134b0b8758353bdabc5ba2817f4e6e0786e176686b2e75a7c47d073f346d6adb2684a9d28b658dddc75b3c5d10a22a3e85c6c12549d
0ce7577e79a068405d3904f3f6b9cc408c4cd8595bf67fe672474e0b94dc99072caaa4f866fc6c3feddc74f10d6a0fb31864f52adef71649
684f1a72c910ec5ca7909cc10aef85d43a57ec91f096a2d4794299e967fcd5add6e9cfb5baf7751387e24b93dbc1f37315ce573dc063ecdd
d4ae6fb9127307cfc80a037e7ff5c40a5f7590c8b2f5bd06dd392fbc51e5d059cffbcb85555L
    n2 = 0x60d175fdb0a96eca160fb0cbf8bad1a14dd680d353a7b3bc77e620437da70fd9153f7609efde652b825c4ae7f25decf14a3c8
240ea8c5892003f1430cc88b0ded9dae12ebffc6b23632ac530ac4ae23fbffb7cfe431ff3d802f5a54ab76257a86aeec1cf47d482fec970f
c27c5b376fbf2cf993270bba9b78174395de3346d4e221d1eafdb8eecc8edb953d1ccaa5fc250aed83b3a458f9e9d947c4b01a6e72ce4fee
37e77faaf5597d780ad5f0a7623edb08ce76264f72c3ff17afc932f5812b10692bcc941a18b6f3904ca31d038baf3fc1968d1cc0588a656d
0c53cd5c89cedba8a5230956af2170554d27f524c2027adce84fd4d0e018dc88ca4d5d26867L
    n3 = 0x280f992dd63fcabdcb739f52c5ed1887e720cbfe73153adf5405819396b28cb54423d196600cce76c8554cd963281fc4b153e
3b257e96d091e5d99567dd1fa9ace52511ace4da407f5269e71b1b13822316d751e788dc935d63916075530d7fb89cbec9b02c01aef19c39
b4ecaa1f7fe2faf990aa938eb89730eda30558e669da5459ed96f1463a983443187359c07fba8e97024452087b410c9ac1e39ed1c74f380f
d29ebdd28618d60c36e6973fc87c066cae05e9e270b5ac25ea5ca0bac5948de0263d8cc89d91c4b574202e71811d0ddf1ed23c1bc35f3a04
2aac6a0bdf32d37dede3536f70c257aafb4cfbe3370cd7b4187c023c35671de3888a1ed1303L
    m = broadcast(n1, n2 ,n3, c1, c2, c3)
    print m    # 输出明文m
    print libnum.n2s(m)   # 输出明文数字转化为字符串后的结果

if __name__=="__main__":
    main()
```

（题目给的是十六进制，我们可以先将其转换为十进制在解密，或者解密后再用libnum.n2s(m)将其转换为字符串）

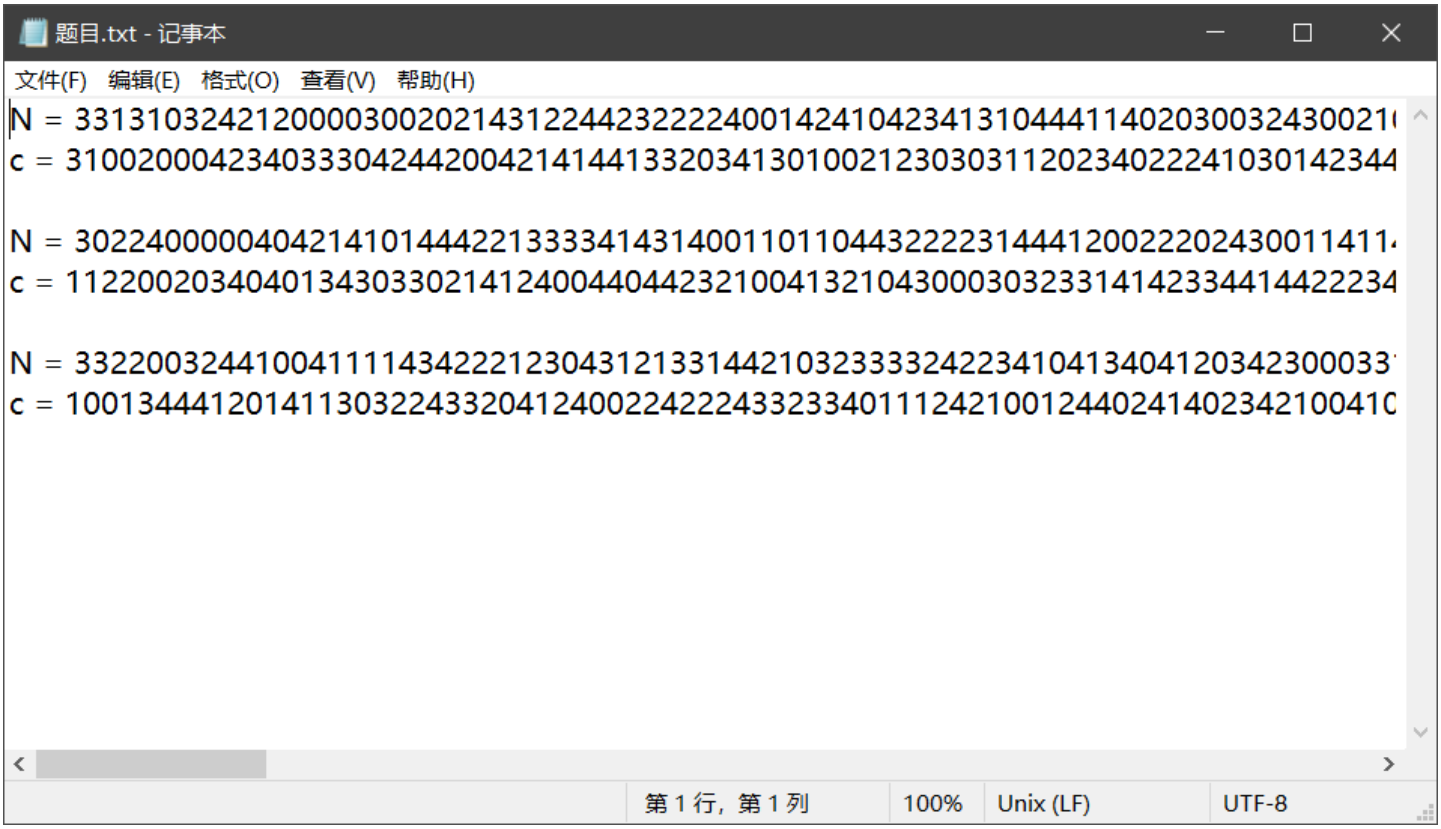
得到的明文m是一大串数字。该脚本即使不提供确切的e值也可以求出结果m，因为他自动爆破合适的e，这在下面的题目中很有用。

解题脚本2：使用的是中国剩余定理解的题，代码确实简洁。

```
# -*- coding: UTF-8 -*-
import gmpy2
import libnum

def CRT(data):
 sum = 0
 m = 1
 for n in data:
  m = m*n[0]
 for n,c in data:
  m1 = m/n
  mr = gmpy2.invert(m1,n)
  sum = sum+mr*m1*c
 return sum%m

c1 = 0x9e84763bdbe246fad0a9cd52fda6233e6128a6210efaf3e6dea4fe272f78ad1f8f5cc7022f62f4f542341128e42d6fd10e67c5f96
edbd243917c0151289f7228e44019b8c65a541d7306b398465e26b69cab36cc61e4ac094832b4299bbaf4630b722a0fb4f1997053be97e92
6f94afb55a0bb6ef00ab694e2f595d9eb8ca96c49f5cbbe194529f68a1aaf6f5151484b471285ba8fc8cd30b55612f35a74dc68e255c3635
79a80d27ce5090873ac719ba59f2492c91fd28bcce26b6a02bae005cbbd2a4cfe5b93442be8664d2313d412e7e09f545c64b7b74bbc408b6
e574d0d300135cba8d6c1d73737d59baca9992ede644d856eb4cfcda562a75743e4b491
c2 = 0x9817fdc7b31a8f9cde1794096d3aa2bc6fe06fe34d4b7c9ca9a77982adf67fd4a7e636659553f4168a16757dc3a75e54ff850b9a9
4a5270f4f75502c7055a3a389df2ea6b00784a4e78e66901b427253c0f343f127e0ff162a349bb14eb4c1453fc6daace19bba4940d77c435
686ef3b59f732072cde2e148d1a64f9682b3f1ceb9a000d87e180a1f9eb20c59dbebc13ddb2e07b64db89217f40369aeec878a45d99909ab
2a3e4cdb74aa68890c941315ae289d6667200c53f9a32c8a64bfc74e62898ac03c460f945a13f11ee28860a3cd07526c30aa92eb89442a76
549fe4ed8a43d14fdeeb350e90443a3a586db719f8610eb5d4a8f5bd1e481b5ef6e96ef
c3 = 0xb0c5ee1ac47c671c918726287e70239147a0357a9638851244785d552f307ed6a049398d3e6f8ed373b3696cfbd0bce1ba88d152f
48d4cea82cd5dafd50b9843e3fa2155ec7dd4c996edde630987806202e45821ad6622935393cd996968fc5e251aa3539ed593fe893b15d21
ecbe6893eba7fe77b9be935ca0aeaf2ec53df7c7086349eb12792aefb7d34c31c18f3cd7fb68e8a432652ef76096096e1a5d7ace90a282fa
cf2d2760e6b5d98f0c70b23a6db654d10085be9dcc670625646a153b52c6c710efe8eb876289870bdd69cb7b45813e4fcfce815d19183892
6e9d60dd58be73565cff0e10f4e80122e077a5ee720caedc1617bf6a0bb072bbd2dab0

n1 = 0x43d819a4caf16806e1c540fd7c0e51a96a6dfdbe68735a5fd99a468825e5ee55c4087106f7d1f91e10d50df1f2082f0f32bb82f39
8134b0b8758353bdabc5ba2817f4e6e0786e176686b2e75a7c47d073f346d6adb2684a9d28b658dddc75b3c5d10a22a3e85c6c12549d0ce7
577e79a068405d3904f3f6b9cc408c4cd8595bf67fe672474e0b94dc99072caaa4f866fc6c3feddc74f10d6a0fb31864f52adef71649684f
1a72c910ec5ca7909cc10aef85d43a57ec91f096a2d4794299e967fcd5add6e9cfb5baf7751387e24b93dbc1f37315ce573dc063ecddd4ae
6fb9127307cfc80a037e7ff5c40a5f7590c8b2f5bd06dd392fbc51e5d059cffbcb85555L
n2 = 0x60d175fdb0a96eca160fb0cbf8bad1a14dd680d353a7b3bc77e620437da70fd9153f7609efde652b825c4ae7f25decf14a3c8240e
a8c5892003f1430cc88b0ded9dae12ebffc6b23632ac530ac4ae23fbffb7cfe431ff3d802f5a54ab76257a86aeec1cf47d482fec970fc27c
5b376fbf2cf993270bba9b78174395de3346d4e221d1eafdb8eecc8edb953d1ccaa5fc250aed83b3a458f9e9d947c4b01a6e72ce4fee37e7
7faaf5597d780ad5f0a7623edb08ce76264f72c3ff17afc932f5812b10692bcc941a18b6f3904ca31d038baf3fc1968d1cc0588a656d0c53
cd5c89cedba8a5230956af2170554d27f524c2027adce84fd4d0e018dc88ca4d5d26867L
n3 = 0x280f992dd63fcabdcb739f52c5ed1887e720cbfe73153adf5405819396b28cb54423d196600cce76c8554cd963281fc4b153e3b25
7e96d091e5d99567dd1fa9ace52511ace4da407f5269e71b1b13822316d751e788dc935d63916075530d7fb89cbec9b02c01aef19c39b4ec
aa1f7fe2faf990aa938eb89730eda30558e669da5459ed96f1463a983443187359c07fba8e97024452087b410c9ac1e39ed1c74f380fd29e
bdd28618d60c36e6973fc87c066cae05e9e270b5ac25ea5ca0bac5948de0263d8cc89d91c4b574202e71811d0ddf1ed23c1bc35f3a042aac
6a0bdf32d37dede3536f70c257aafb4cfbe3370cd7b4187c023c35671de3888a1ed1303L
e = 3

n = [n1,n2,n3]
c = [c1,c2,c3]
data = zip(n,c)
m_e = CRT(data)
m = gmpy2.iroot(m_e,e)[0]
print m    # 输出明文数字
print libnum.n2s(m)   # 输出明文数字转为的字符串
```

例题2：[BUUCTF-Crypto]RSA4

题目给出如下：



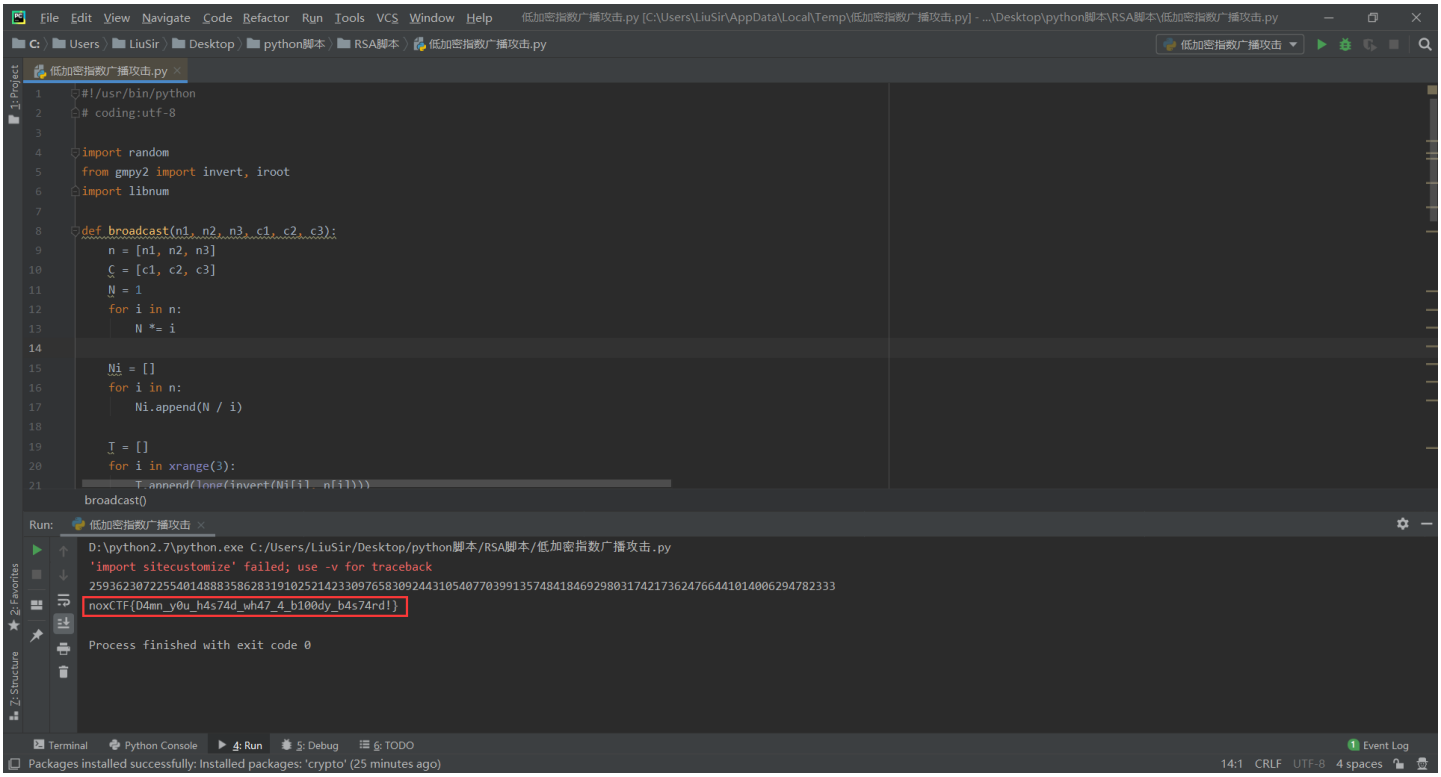
还是模数n和密文c不同，注意，注意这次没有给出e的值。解题的话其实直接爆破e即可，理论上讲e不可能特别大。

编写如下脚本（还是上面那个脚本，因为该脚本为我们自动爆破了e），此题中的n、c均是以5进制表示（只有0到4的数），要先用`int("*****",5)`转换为十进制才能计算：

```
#!/usr/bin/python
# coding:utf-8

import random
from gmpy2 import invert, iroot
import libnum

def broadcast(n1, n2, n3, c1, c2, c3):
    n = [n1, n2, n3]
    C = [c1, c2, c3]
    N = 1
    for i in n:
        N *= i

    Ni = []
    for i in n:
        Ni.append(N / i)

    T = []
    for i in xrange(3):
        T.append(long(invert(Ni[i], n[i])))

    X = 0
    for i in xrange(3):
        X += C[i] * Ni[i] * T[i]

    m3 = X % N
    m = iroot(m3, 3)
```

```
    return m[0]


def main():
    e = 3

    n1 = int('331310324212000030020214312244232222400142410423413104441140203003243002104333214202031202212403400220031202142322434104143104244241214204444443323000244130122022422310201104411044030113302323014101331214303223312402430402404413033243132101010422240133122211400434023222214231402403403200012221023341333340042343122302113410211022123241303024431330001303404020104442443120130000334110042432010203401440404010003442001223042211442001413004',5)
    c1 = int('310020004234033304244200421414413320341301002123030311202340222410301423440312412440240244110200112141140201224032402232131204213012303204422003300004011434102141321223311243242010014140422411342304322201241112402132203101131221223004022003120002110230023341143201404311340311134230140231412201333333142402423134333211302102413111114244300324401233400340443142234004012241113230002422344204412404110210231002220031232143430301220323010422431',5)
    n2 = int('302240000040421410144422133334143140011011044322223144412002220243001141141114123223331331304421113021231204322233120121444434210041232214144413244434424302311222143224402302432102242132244032010020113224011121043232143221203424243134044314022212024343100042342002432331144300214212414033414120004344211330224020301223033334324244031204240122301242232011303211220044222411134403012132420311110302442344021122101224411230002203344140143044114',5)
    c2 = int('112200203404013430330214124004404423210041321043000303233141423344144222343401042200334033203124030011440014210112103234440312134032123400444344144233020130110134042102220302002413321102022414130443041144240310121020100310104334204234412411424420321211112232031121330310333414423433343322024400121200333330432223421433344122023012440013041401423202210124024431040013414313121123433424113113414422043330422002314144111134142044333404112240344',5)
    n3 = int('332200324410041111434222123043121331442103233332422341041340412034230003314420311333101344231212130200312041044324431141033004333110021013020140020011222012300020041342040004002220210223122111314112124333211132230332124022423141214031303144444134403024420111423244424030030003340213032121303213343020401304243330001314023030121034113334404440421242240113103203013341231330004332040302440011324004130324034323430143102401440130242321424020323',5)
    c3 = int('100134441201411303224332041240022422243323340111242100124402414023421004103311314413032420110021013230404033111204213044222222003244022442433222422444414043342130111111330022213203030324422101133032212042042243101434342203204121042113212104212423330331134311311111414320001124000211131212223434000340331204040104302143311203133432432212330411234001403013202143210113021124113442241344231201304214121200310221130032140404301212433201324043124232',5)
    m = broadcast(n1, n2, n3, c1, c2, c3)
    print m  # 输出明文m
    print libnum.n2s(m)   # 数字型（不论是十六进制还是十进制）与字符串之间的转换

if __name__ == "__main__":
    main()
```

如下得到flag：

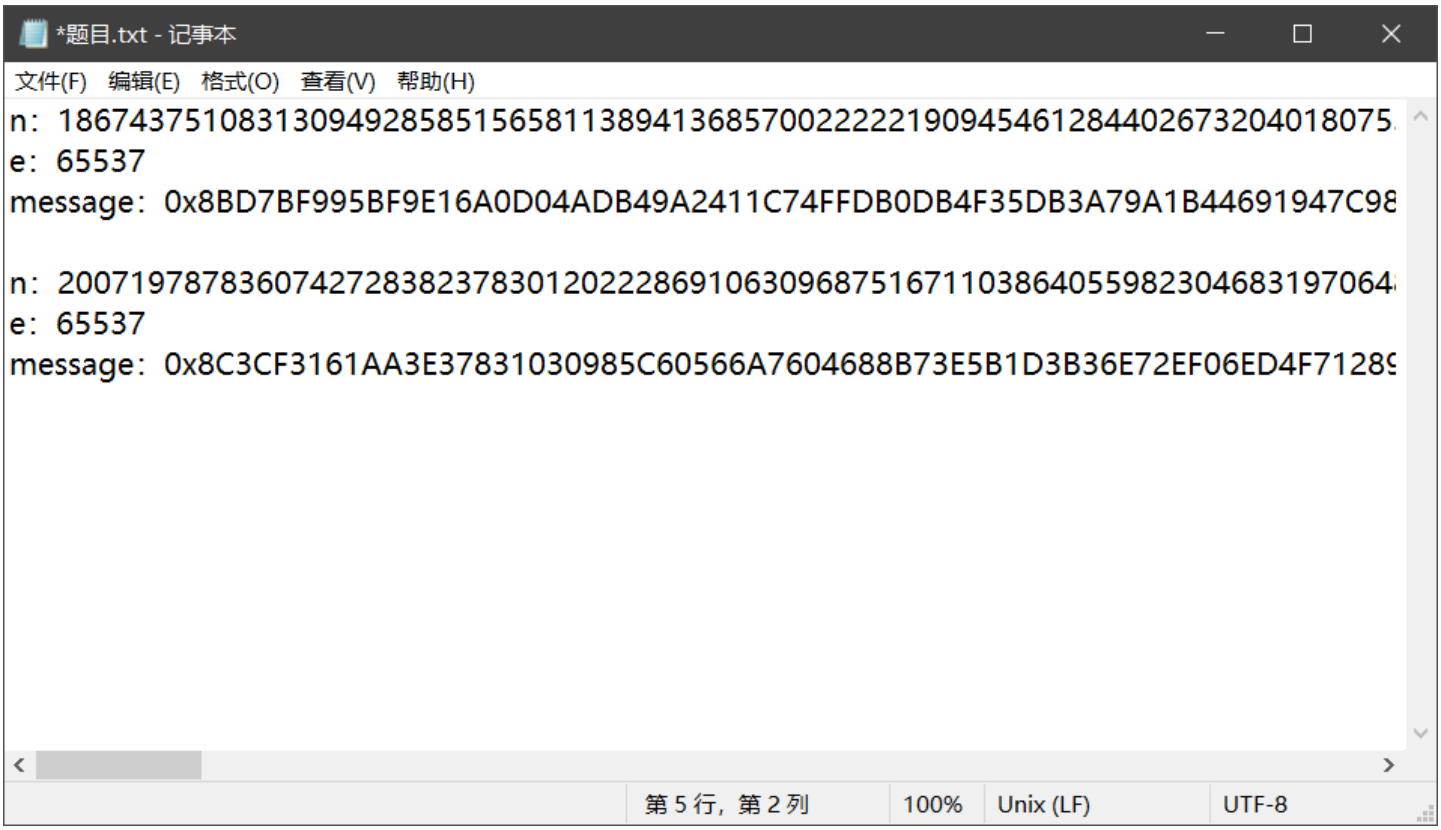


模不互素（存在两个或更多非常大的模数 n，且n1和n2不互质）

适用条件：存在两个或更多非常大的模数 n，且n1和n2不互质，即gcd(N1,N2)!=1，二者有公因数。

例题：

题目给出如下：



让我们求出明文。

由于 不能直接 分解 n，只能先找出 n1，n2 的公因数作为 q，再拿n1，n2除以 q 得到 p1 和p2

编写脚本：

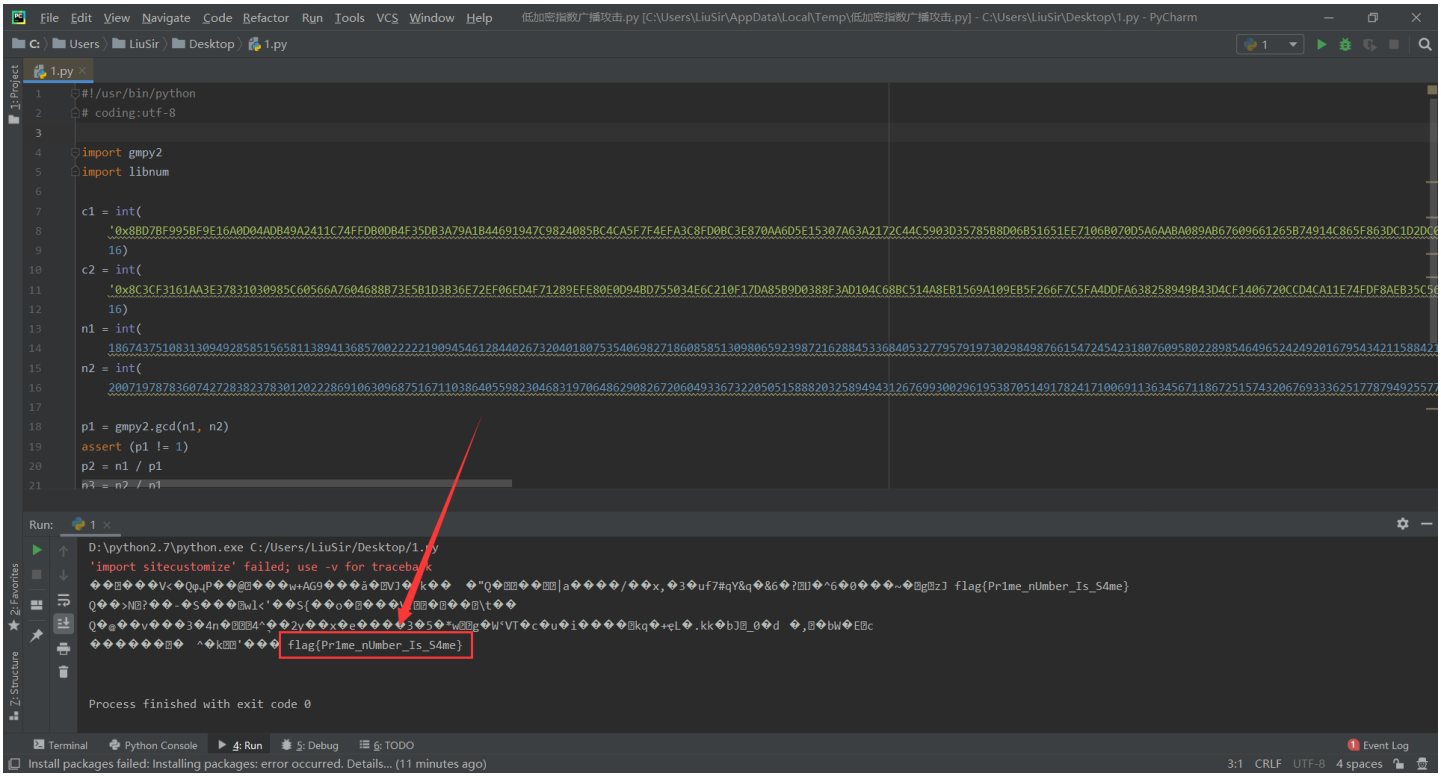
判定 x 和 y 是否互素：

```python
#判断两个数是否互素

def gcd(a, b):  # 判断来两个数是否互素,辗转相除法
    if (b == 0):
        return a
    else:
        return gcd(b, a % b)


def main():
    x = 17  # x,y的值根据需要修改即可
    y = 65537
    if gcd(x, y) == 1:  # 如果两个数的最大公约数是1，那么两数互素。
        print(str(x) + " " + str(y) + "  两个数互素")
    else:
        print(str(x) + " " + str(y) + " 两个数不互素")


if __name__ == "__main__":
    main()
```

解密脚本：（题目给的c1和c2是十六进制，我们可以先将其转换成十进制再解密）

```python
#!/usr/bin/python
# coding:utf-8

import gmpy2
import libnum

c1 = int(
    '0x8BD7BF995BF9E16A0D04ADB49A2411C74FFDB0DB4F35DB3A79A1B44691947C9824085BC4CA5F7F4EFA3C8FD0BC3E870AA6D5E1530
7A63A2172C44C5903D35785B8D06B51651EE7106B070D5A6AABA089AB67609661265B74914C865F863DC1D2DC08CE0B026107A74EC3FDC62
666B50110B9D15A243EAAD6F53646929A3369285404868E42DD0BBE92D956018E3C0B36EF5E9516E433228CFDD06D6E662EC0A9A31061EA1
1F61CA17EABF43D2D4977FC9D6FC53AB6DC01509401B8D9A46B59A9ADAA97D54CC50C27445E4C21B893510620EC3566AD6E8727FA147437B
207505217E6F2DF009E2286C8354D281374D7802D08A2062FE48DBF135BBCAB120EBF84',
    16)
c2 = int(
    '0x8C3CF3161AA3E37831030985C60566A7604688B73E5B1D3B36E72EF06ED4F71289EFE80E0D94BD755034E6C210F17DA85B9D0388F
3AD104C68BC514A8EB1569A109EB5F266F7C5FA4DDFA638258949B43D4CF1406720CCD4CA11E74FDF8AEB35C56A79781C87157FC42135733
29C5B0FF411F8A4F34580AA103DB9FD403C0D409FA11860A7C4595FDC49DC2CF94E5112B772E5DEC8F17E24B10A7FD7A95DCB87BE5E27C32
FC931574A7847BC506A61EFE9DB3D3F612143845FE80D7B3EA548B886A67A29CBDB2775B1F91178B6DA763F1A6ECFF46592E4C7FFAAB6C9F
EF29D9CB9E035A3D98ECFFB26BA2EEAA56D1CD096E6A2CF9A58086CAD7718DDA5CB0C1B',
    16)
n1 = int(
    186743751083130949285851565811389413685700222221909454612844026732040180753540698271860858513098065923987216
288453368405327795791973029849876615472454231807609580228985464965242492016795434211588421034964528619321831443433
331592510615432206679661241561634229102396212705531130761389858385017792293068515535138050058726361159189313758870
708003711296496548004793832636078992866149115453883484010146248683416979269684197112659302912316105354447631916609
095873601039087467195861855933867945320660341121646617237488740454702251292985183856835611226238599244356006735011
8624442290740294392946469444865207441210588886717886735772 7)
n2 = int(
    200719787836074272838237830120222869106309687516711038640559823046831970648629082672060493367322050515888203258
949431267699300296195387051491782417100691136345671186725157432067693336251778794925577033591785283424895851567
136235306543195007385081468312234877328248350056979327044270466753927149226835843764492035946415407945578718815
814072280966424177446112615571015730501632859199717112148562430313548459455648371096574945239022964444637487236
391096124380125900847718653777954090005869927329715945983552726097890791470618526644721153953445048226446519574963
078949984673093470383494704719007760507695781522033491289 51)

p1 = gmpy2.gcd(n1, n2)
assert (p1 != 1)
p2 = n1 / p1
p3 = n2 / p1
e = 0x10001
d1 = gmpy2.invert(e, (p1 - 1) * (p2 - 1))
d2 = gmpy2.invert(e, (p1 - 1) * (p3 - 1))
m1 = pow(c1, d1, n1)
m2 = pow(c2, d2, n2)
print libnum.n2s(m1) + libnum.n2s(m2)
```

执行后即可得到flag：



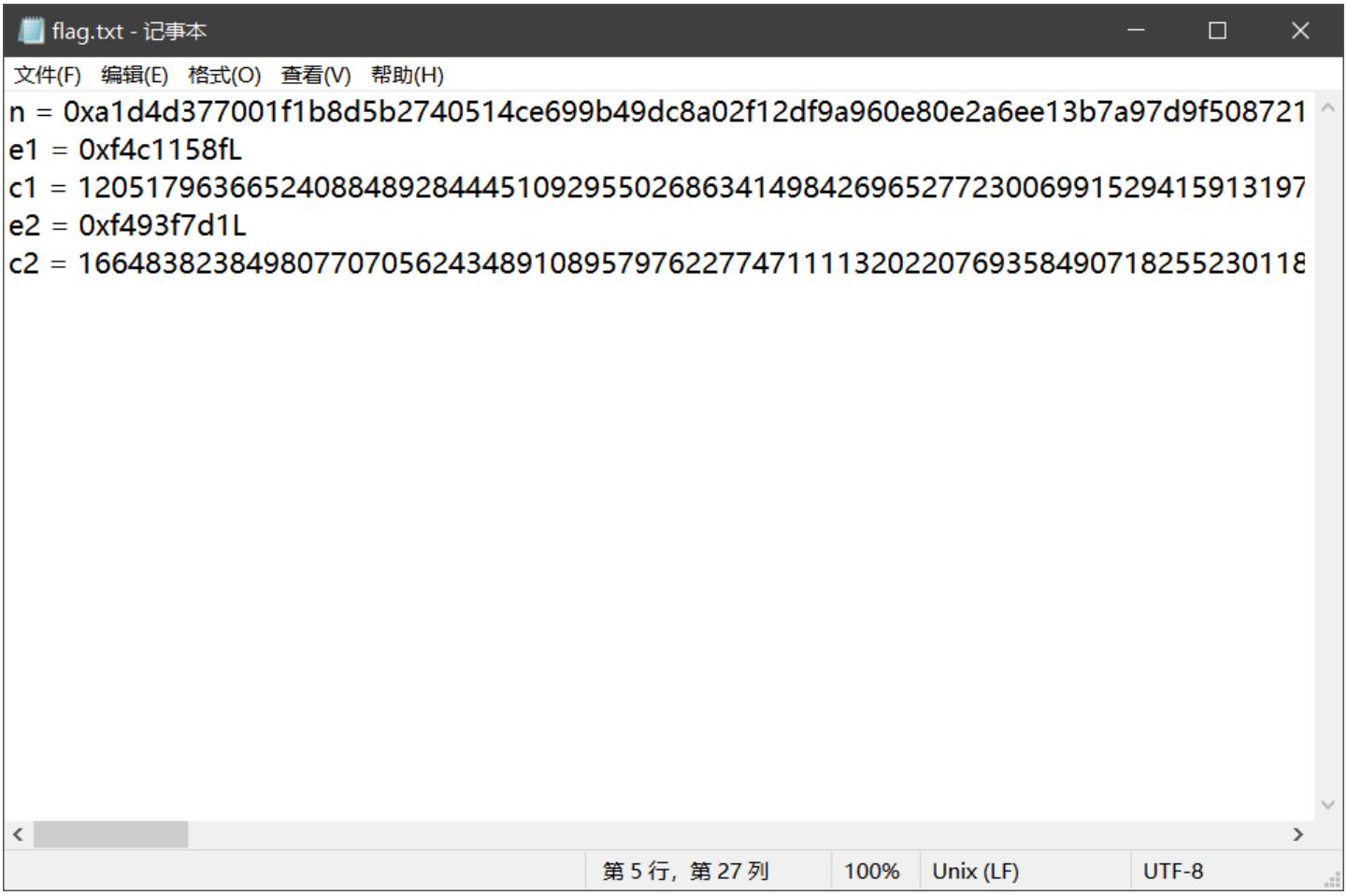
共模攻击（m，n相同、e，c不同，且e1 和 e2互质）

适用情况：明文m、模数n相同，公钥指数e、密文c不同，gcd(e1,e2)==1，也就是e1和e2互质。

对同一明文的多次加密使用相同的模数和不同的公钥指数可能导致共模攻击。

例题：

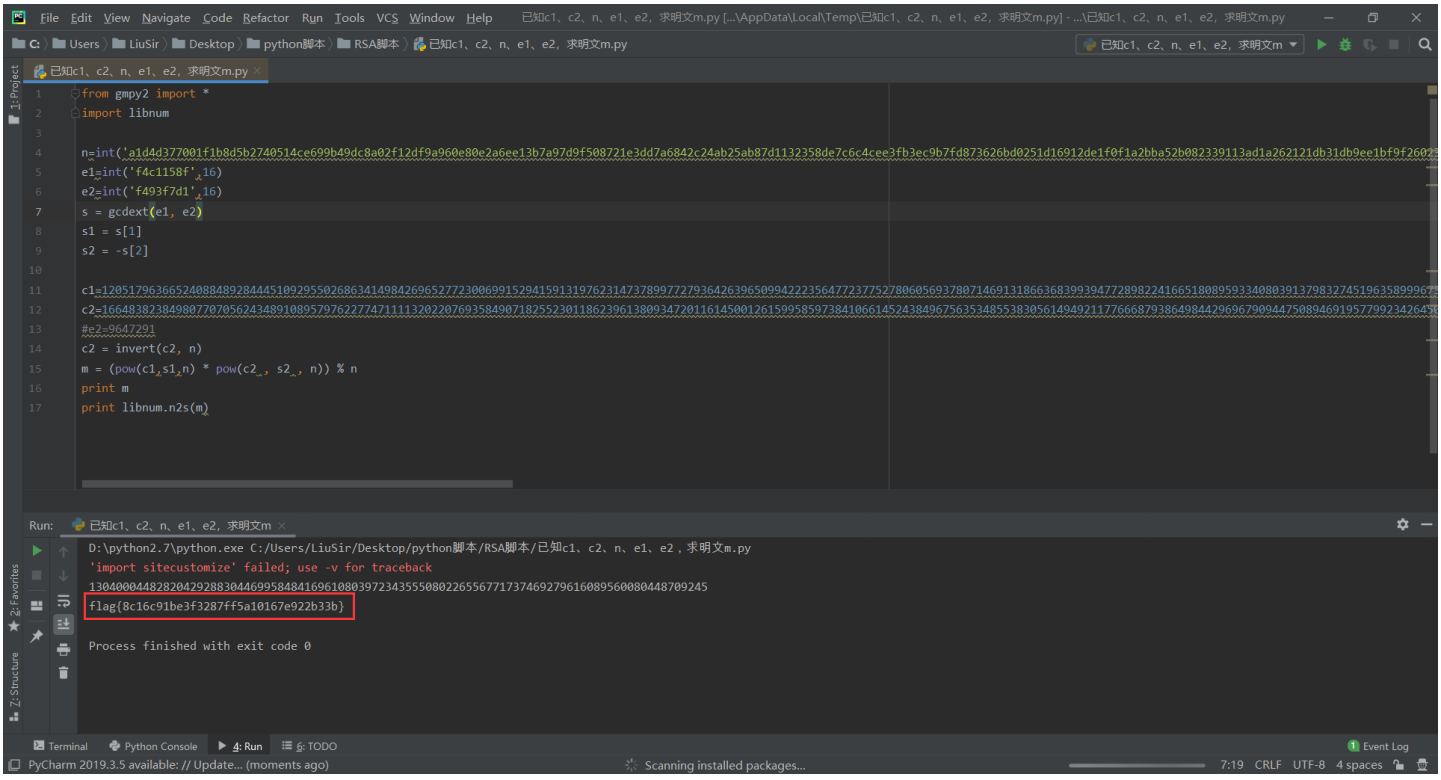
题目给出如下：



可以发现，明文m、模数n相同，公钥指数e、密文c不同，且e1 和 e2互质，为共模攻击，直接上脚本：

```
from gmpy2 import *
import libnum

n=int('a1d4d377001f1b8d5b2740514ce699b49dc8a02f12df9a960e80e2a6ee13b7a97d9f508721e3dd7a6842c24ab25ab87d1132358de
7c6c4cee3fb3ec9b7fd873626bd0251d16912de1f0f1a2bba52b082339113ad1a262121db31db9ee1bf9f26023182acce8f84612bfeb0758
03cf610f27b7b16147f7d29cc3fd463df7ea31ca860d59aae5506479c76206603de54044e7b778e21082c4c4da795d39dc2b9c0589e577a7
73133c89fa8e3a4bd047b8e7d6da0d9a0d8a3c1a3607ce983deb350e1c649725cccb0e9d756fc3107dd4352aa18c45a65bab7772a4c5aef7
020a1e67e6085cc125d9fc042d96489a08d885f448ece8f7f254067dfff0c4e72a63557',16)
e1=int('f4c1158f',16)
e2=int('f493f7d1',16)
s = gcdext(e1, e2)
s1 = s[1]
s2 = -s[2]

c1=120517963665240884892844451092955026863414984269652772300699152941591319762314737899772793642639650994222356472737527806056937807146913186636839939472898224166518089593340803913798327451963589996734323497943301819051718
709807518655868569656941242449109980876397661605271517459716669684900920279597477446629607627693769738733623143693170696779851882404994923673483971528314806130892416509854017091137325195201225617407959645788145876202882024723106204183257094755002924708009138560347432552090905489132135154932987521239299578509008290614398700799670928805692609756924823628055245227290288940649158862576448537833423
c2=166483823849807707056243489108957976227747111132022076935849071825523011862396138093472011614500126159958597384106614524384967563534855383056149492117766687938649844296967909447508946919577992342645085300840268946112285136989633474023291098381096216097704069257005209833878114510748384703700446786340992020034809259032675087440061954552340253250608172238138589850747208721241681429439264676946767177135035590071128743817500054063714001099629435083494971511484460648460965314450374161749139159230503322428434039261331658173102726338843582637785167702885155929595983215176249952636313180194516350199933780820807438121279 5
#e2=9647291
c2 = invert(c2, n)
m = (pow(c1,s1,n) * pow(c2 , s2 , n)) % n
print m
print libnum.n2s(m)
```

（注意进制转换，上面的n、e1、e2为十六进制，c1、c2为十进制，我们要把他们转化为相同的进制）



如上图得到flag。