CTF-Crypto-RSA整理

原创

OceanSec 于 2021-10-28 21:32:07 发布 1681 收藏 7

分类专栏： # CTF 文章标签： ctf rsa crypto

本文链接：https://blog.csdn.net/q20010619/article/details/121023558

CTF 专栏收录该内容

66 篇文章 29 订阅

订阅专栏

rsa基本参数

```
N：大整数N，我们称之为模数（modulus）
p 和 q ：大整数N的两个因子（factor）
e 和 d：互为模反数的两个指数（exponent）
c 和 m：分别是密文和明文

{N,e}称为公钥,{N,d}称为私钥
```

加密过程：

c=m^e mod n

```
c=pow(m,e,n)
```

解密过程：

m=c^d mod n

```
m=pow(c,d,n)
```

求解私钥d：

```
d = gmpy2.invert(e, (p-1)*(q-1))
```

一般来说，n，e是公开的，但是由于n一般是两个大素数的乘积，所以我们很难求解出d，所以RSA加密就是利用现代无法快速实现大素数的分解，所存在的一种安全的非对称加密

应用流程

1. 选取两个较大的互不相等的质数p和q，计算`n = p * q`。
2. 计算`phi = (p-1) * (q-1)`。
3. 选取任意e，使得e满足`1<e<phi`且`gcd(e , phi) == 1`。
4. 计算e关于 phi 的模逆元d， 即d满足`(e * d)% phi ==1`。
5. 加解密：`c = (m ^ e) % n`， `m = (c ^ d) % n`。其中m为明文，c为密文，(n,e)为公钥对，d为私钥，要求`0 <= m < n`

工具

RsaCtfTool

CTF-RSA-tool

安装

```
安装之前必须先安装这四个库(PyCrypto,GMPY2,SymPy,requests)

git clone https://github.com/Ganapati/RsaCtfTool.git
cd RsaCtfTool  //进入这个目录
安装python第三方库
pip install -r requirements.txt
```

用法一：已知公钥(自动求私钥) –publickey，密文 —-uncipherfile。
将文件解压复制到RsaCtfTool里

```
python RsaCtfTool.py --publickey 公钥文件 --uncipherfile 加密的文件
```

用法二：已知公钥求私钥

```
python RsaCtfTool.py --publickey 公钥文件 --private
```

用法三：密钥格式转换——把PEM格式的公钥转换为n，e

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

用法四：密钥格式转换——把n,e转换为PEM格式

```
python RsaCtfTool.py --createpub -n 782837482376192871287312987398172312837182 -e 65537
```

rsatool

安装：

```
git clone https://github.com/ius/rsatool.git
cd rsatool  //进入这个目录
python setup.py install
```

提供模数和私有指数，PEM输出到key.pem：

```
python rsatool.py -f PEM -o key.pem -n 13826123222358393307 -d 9793706120266356337
```

提供两个素数，DER输出到key.der：

```
python rsatool.py -f DER -o key.der -p 4184799299 -q 3303891593
```

项目地址:https://github.com/ius/rsatool

openssl

1. 生成PKCS#1私钥

```
openssl genrsa -out rsa_prikey.pem 1024
    -out 指定生成文件，此文件包含公钥和私钥两部分，所以即可以加密，也可以解密
    1024 生成密钥的长度(生成私钥为PKCS#1)
```

2.把RSA私钥转换成PKCS8格式

```
openssl pkcs8 -topk8 -inform PEM -in rsa_prikey.pem -outform PEM -nocrypt -out prikey.pem
```

3. 根据私钥生成公钥

```
openssl rsa -in rsa_prikey.pem -pubout -out pubkey.pem
    -in 指定输入的密钥文件
    -out 指定提取生成公钥的文件(PEM公钥格式)
```

4. 提取PEM RSAPublicKey格式公钥

```
openssl rsa -in key.pem -RSAPublicKey_out -out pubkey.pem
    -in 指定输入的密钥文件
    -out 指定提取生成公钥的文件(PEM RSAPublicKey格式)
```

5. 公钥加密文件

```
openssl rsautl -encrypt -in input.file -inkey pubkey.pem -pubin -out output.file
    -in 指定被加密的文件
    -inkey 指定加密公钥文件
    -pubin 表面是用纯公钥文件加密
    -out 指定加密后的文件
```

6. 私钥解密文件

```
openssl rsautl -decrypt -in input.file -inkey key.pem -out output.file
    -in 指定需要解密的文件
    -inkey 指定私钥文件
    -out 指定解密后的文件
```

ras 的用法如下：

```
openssl rsa [-inform PEM|NET|DER] [-outform PEM|NET|DER] [-in filename] [-passin arg] [-out filename] [-passout
arg]
       [-sgckey] [-des] [-des3] [-idea] [-text] [-noout] [-modulus] [-check] [-pubin] [-pubout] [-engine id]</pr
e>

常用选项：

-in filename：指明私钥文件
-out filename：指明将提取出的公钥保存至指定文件中
-pubin:根据公钥提取出私钥
-pubout：根据私钥提取出公钥
```

python库

1. libnum
2. gmpy2
3. pycryptodemo

sagemath

下载地址：https://mirrors.tuna.tsinghua.edu.cn/sagemath/linux/64bit/index.html

安装

```
tar xvf sage-8.0-Ubuntu_16.04-x86_64.tar.bz2
cd SageMath
```

分离整数

- www.factordb.com：N比特位数768或者更高，在线网站会存储一些已分解成功的N
- 命令行分解，factordb-pycli，借用 factordb 数据库
- yafu：q、p差值太大或者太小，p+1/p-1光滑，适用Fermat或Pollard rho法分解

相关攻击

基础RSA加密脚本

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

msg = 'flag is :testflag'
hex_msg=int(msg.encode("hex"),16)
print(hex_msg)
p=getPrime(100)
q=getPrime(100)
n=p*q
e=0x10001
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
print("d=",hex(d))
c=pow(hex_msg,e,n)
print("e=",hex(e))
print("n=",hex(n))
print("c=",hex(c))
```

以下部分感谢 jerry 师傅整理：http://panaceasec.cn/

基础RSA解密脚本

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
import binascii
import gmpy2
n=0x80b32f2ce68da974f25310a23144977d76732fa78fa29fdcbf
#这边我用yafu分解了n
p=78090079033426965944329795684 3
q=10345265594079935077348184088 29
e=0x10001
c=0x534280240c65bb1104ce3000bc8181363806e7173418d15762


phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
m=pow(c,d,n)
print(hex(m))
print(binascii.unhexlify(hex(m)[2:].strip("L")))
```

有n、e、c

```
import gmpy2
import libnum

n= 790599770834333691619771594722575631090081194757552884397747588248878368574243360325186518620885907002419802001585428557621222621564456328977574444225141580629965010373233 79
e= 65537
c= 315735919869150018576402634669391642063072477484651483959788107202150949707070020437219910557890845188315406526528422586327528997995926456407090743854001678292132431679568 1
p1 = 2514358789
q=107283086870331422422630427208638208443839610981393914768563788464392025680580601753303238899632937208742631742549284667038295373889873573840568779384826 83
p2 = 2930880917
d = gmpy2.invert(e,(p1-1)*(q-1)*(p2-1))
# print(d)
m = pow(c,d,n)
print(libnum.n2s(m))
```

0x03 p和q相差过大或过小

利用条件

因为n=p*q
其中若p和q的值相差较小，或者较大，都会造成n更容易分解的结果
例如出题如下

```
p=getPrime(512)
q=gmpy2.next_prime(p)
n=p*q
```

因为p和q十分接近，所以可以使用yafu直接分解

yafu分解

使用

```
factor(*)
```

括号中为要分解的数

在线网站分解

http://factordb.com/
通过在此类网站上查询n，如果可以分解或者之前分解成功过，那么可以直接得到p和q

0x04 公约数分解n

利用条件

当题目给的多对公钥n是公用了一个素数因子的时候，可以尝试公约数分解
出题一般如下

```
p1=getPrime(512)
p2=getPrime(512)
q=getPrime(512)
n1=p1*q
n2=p2*q
```

所以当题目给了多个n，并且发现n无法分解，可以尝试是否有公约数。

欧几里得辗转相除法

求公约数可以使用欧几里得辗转相除法，实现python脚本如下

```
def gcd(a, b):   #求最大公约数
    if a < b:
        a, b = b, a
    while b != 0:
        temp = a % b
        a = b
        b = temp
    return a
```

用例

```
def gcd(a, b):   #求最大公约数
    if a < b:
        a, b = b, a
    while b != 0:
        temp = a % b
        a = b
        b = temp
    return a

n1=0x6c9fb4bf11344e4c818be178e3d3db352797099f929e4ba8fa86d9c4ce3d8f71e3daa8c795b67dc2dabe1e1608836904386c364ecec
759c27eaa83eb93710003d4cc848e558f7b11372405c5787b60eca627372767455a5fcf30cb6c157ca5a6267d63ffa16fe49e7433136a479
45de2219f46a35f2b6a58196057c602e72a0b
n2=0x46733cc071bdee0d178fb32836a6b0a2f145a681df47d31ea9d9fc5b5fa0cc7ddbcd34531aefeace9840fc890f7a111f73593c9a418
86b9a6f91cde3e6f9c71821a8ad877de51f78094599209746e80635c5625459ad7ba14f926b74875c8980a9436d6bbd54e1d9da72ae20038
3516098c04e24f58b23b4a8142cef0c931a55
print(gcd(n1,n2))
```

使用欧几里得辗转相除得到共有的因子，然后n1和n2除以这个因子，即可得到另一个素数因子。

0x05 模数分解

场景

已知e,d,n求p,q
例如

```
('d=', '0x455e1c421b78f536ec24e4a797b5be78df09d8d9e3b7f4e2244138a7583e810adf6ad056bb59a91300c9ead5ed77ea6bafdebf
7ab2d9ec200127901083c7ffca45e83f2c934358366a2b6207b96a0eae6df0476060c063c281512834a42350a3b56bc09f5cec1a6975257d
7f12a58f6389060e49b41f05e88ea2b30b395f6391')
('e=', '0x10001')
('n=', '0x71ee0f4883690893ab503e97e25e6308d4c1e0a050cbea7b9c040f7a5b5b484afcecc8a9b3cc6bf089a1e83281562df217caab
7220e3dfc14399139ce437af2f131f9345675e4d848cfab5827818eeab7834374be4a0513f81f3df125a932c2bb4c24c834d798bcc80f9c4
a8770b01f8e54620b72a4f0491edd391e635d48e71L')
```

模数分解

私钥d的获取是通过

```
d = gmpy2.invert(e, (p-1)*(q-1))
```

分解p,q python实现如下

```
import random
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
```

完整用例

```
import random
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

n=0x71ee0f4883690893ab503e97e25e6308d4c1e0a050cbea7b9c040f7a5b5b484afcecc8a9b3cc6bf089a1e83281562df217caab7220e3
dfc14399139ce437af2f131f9345675e4d848cfab5827818eeab7834374be4a0513f81f3df125a932c2bb4c24c834d798bcc80f9c4a8770b
01f8e54620b72a4f0491edd391e635d48e71
e=0x10001
d=0x455e1c421b78f536ec24e4a797b5be78df09d8d9e3b7f4e2244138a7583e810adf6ad056bb59a91300c9ead5ed77ea6bafdebf7ab2d9
ec200127901083c7ffca45e83f2c934358366a2b6207b96a0eae6df0476060c063c281512834a42350a3b56bc09f5cec1a6975257d7f12a5
8f6389060e49b41f05e88ea2b30b395f6391
p,q=getpq(n,e,d)
print("p=",p)
print("q=",q)
print(p*q==n)
```

0x06 dp&dq泄露

介绍

首先了解一下什么是dp、dq

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

这种参数是为了让解密的时候更快速产生的

场景

假设题目仅给出p，q，dp，dq，c，即不给公钥e

```
('p=', '0xf85d730bbf09033a75379e58a8465f8048b8516f8105ce2879ce774241305b6eb4ea506b61eb7e376d4fcd425c76e80cb748eb
faf3a852b5cf3119f028cc5971L')
('q=', '0xc1f34b4f826f91c5d68c5751c9af830bc770467a68699991be6e847c29c13170110ccd5e855710950abab2694b6ac730141152
758acbeca0c5a51889cbe84d57L')
('dp=', '0xf7b885a246a59fa1b3fe88a2971cb1ee8b19c4a7f9c1a791b9845471320220803854a967a1a03820e297c0fc1aabc2e1c4022
8d50228766ebebc93c97577f511')
('dq=', '0x865fe807b8595067ff93d053cc269be6a75134a34e800b741cba39744501a31cffd31cdea6078267a0bd652aeaa39a49c73d9
121fafdfa7e1131a764a12fdb95')
('c=', '0xae05e0c34e2ba4ca3536987cc2cfc3f1f7f53190164d0ac50b44832f0e7224c6fdeebd2c91e3991e7d179c26b1b997295dc972
4925ba431f527fba212703a0d14a34ce133661ae0b6001ee326303d6ccdc27dbd94e0987fae25a84f197c1535bdac9094bfb3846b7ca696b
2e5082bea7bff804da275772ca05dd51b185a4fc30L')
```

解密代码如下

```
InvQ=gmpy2.invert(q,p)
mp=pow(c,dp,p)
mq=pow(c,dq,q)
m=(((mp-mq)*InvQ)%p)*q+mq
print '{:x}'.format(m).decode('hex')
```

解题完整脚本

```
import gmpy2
import binascii
def decrypt(dp,dq,p,q,c):
    InvQ = gmpy2.invert(q,p)
    mp = pow(c,dp,p)
    mq = pow(c,dq,q)
    m=(((mp-mq)*InvQ)%p)*q+mq
    print (binascii.unhexlify(hex(m)[2:]))

p=0xf85d730bbf09033a75379e58a8465f8048b8516f8105ce2879ce774241305b6eb4ea506b61eb7e376d4fcd425c76e80cb748ebfaf3a8
52b5cf3119f028cc5971
q=0xc1f34b4f826f91c5d68c5751c9af830bc770467a68699991be6e847c29c13170110ccd5e855710950abab2694b6ac730141152758acb
eca0c5a51889cbe84d57
dp=0xf7b885a246a59fa1b3fe88a2971cb1ee8b19c4a7f9c1a791b9845471320220803854a967a1a03820e297c0fc1aabc2e1c40228d5022
8766ebebc93c97577f511
dq=0x865fe807b8595067ff93d053cc269be6a75134a34e800b741cba39744501a31cffd31cdea6078267a0bd652aeaa39a49c73d9121faf
dfa7e1131a764a12fdb95
c=0xae05e0c34e2ba4ca3536987cc2cfc3f1f7f53190164d0ac50b44832f0e7224c6fdeebd2c91e3991e7d179c26b1b997295dc9724925ba
431f527fba212703a0d14a34ce133661ae0b6001ee326303d6ccdc27dbd94e0987fae25a84f197c1535bdac9094bfb3846b7ca696b2e5082
bea7bff804da275772ca05dd51b185a4fc30
decrypt(dp,dq,p,q,c)
```

0x07 dp泄露

场景介绍

假设题目给出公钥n,e以及dp

```
('dp=', '0x7f1344a0b8d2858492aaf88d692b32c23ef0d2745595bc5fe68de384b61c03e8fd054232f2986f8b279a0105b7bee85f74378
c7f5f35c3fd505e214c0738e1d9')
('n=', '0x5eee1b4b4f17912274b7427d8dc0c274dc96baa72e43da36ff39d452ff6f2ef0dc6bf7eb9bdab899a6bb718c070687feff517f
cf5377435c56c248ad88caddad6a9cefa0ca9182daffcc6e48451d481f37e6520be384bedb221465ec7c95e2434bf76568ef81e988039829
a2db43572e2fe57e5be0dc5d94d45361e96e14bd65L')
('e=', '0x10001')
('c=', '0x510fd8c3f6e21dfc0764a352a2c7ff1e604e1681a3867480a070a480f722e2f4a63ca3d7a92b862955ab4be76cde43b51576a1
28fba49348af7a6e34b335cfdbda8e882925b20503762edf530d6cd765bfa951886e192b1e9aeed61c0ce50d55d11e343c78bb617d8a0adb
7b4cf3b913ee85437191f1136e35b94078e68bee8dL')
```

给出密文要求解明文
我们可以通过n，e，dp求解私钥d

求解公式推导

公式推导参考简书
https://www.jianshu.com/p/74270dc7a14b
首先dp是

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

以下推导过程如果有问题欢迎指正
现在我们可以知道的是

```
c≡m^e mod n
m≡c^d mod n
φ(n)=(p-1)*(q-1)
d*e≡1 mod φ(n)
dp≡d mod (p-1)
```

由上式可以得到

```
dp*e≡d*e mod (p-1)
```

因此可以得到

```
d*e=k*(p-1)+dp*ed*e≡1 mod φ(n)
```

我们将式1带入式2可以得到

```
k*(p-1)+dp*e≡1 mod (p-1)*(q-1)
```

故此可以得到

```
k2*(p-1)*(q-1)+1=k1*(p-1)+dp*e
```

变换一下

```
(p-1)*[k2*(q-1)-k1]+1=dp*e
```

因为

```
dp<p-1
```

可以得到

```
e>k2*(q-1)-k1
```

我们假设

```
x=k2*(q-1)-k1
```

可以得到x的范围为

```
(0,e)
```

因此有

```
x*(p-1)+1=dp*e
```

那么我们可以遍历

```
x∈(0,e)
```

求出p-1，求的方法也很简单，遍历65537种可能，其中肯定有一个p可以被n整除那么求出p和q，即可利用

```
φ(n)=(p-1)*(q-1)d*e≡1 mod φ(n)
```

推出

```
d≡1*e-1 mod φ(n)
```

注：这里的-1为逆元，不是倒数的那个-1

公式的python实现

求解私钥d脚本如下

```
def getd(n,e,dp):
    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
                return d
```

解题完整脚本

```
import gmpy2
import binascii

def getd(n,e,dp):
    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
                return d

dp=0x7f1344a0b8d2858492aaf88d692b32c23ef0d2745595bc5fe68de384b61c03e8fd054232f2986f8b279a0105b7bee85f74378c7f5f3
5c3fd505e214c0738e1d9
n=0x5eee1b4b4f17912274b7427d8dc0c274dc96baa72e43da36ff39d452ff6f2ef0dc6bf7eb9bdab899a6bb718c070687feff517fcf5377
435c56c248ad88caddad6a9cefa0ca9182daffcc6e48451d481f37e6520be384bedb221465ec7c95e2434bf76568ef81e988039829a2db43
572e2fe57e5be0dc5d94d45361e96e14bd65
e=0x10001
c=0x510fd8c3f6e21dfc0764a352a2c7ff1e604e1681a3867480a070a480f722e2f4a63ca3d7a92b862955ab4be76cde43b51576a128fba4
9348af7a6e34b335cfdbda8e882925b20503762edf530d6cd765bfa951886e192b1e9aeed61c0ce50d55d11e343c78bb617d8a0adb7b4cf3
b913ee85437191f1136e35b94078e68bee8d


d=getd(n,e,dp)
m=pow(c,d,n)
print (binascii.unhexlify(hex(m)[2:]))
```

0x08 e与φ(n)不互素

给出p、q、e、c

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
import gmpy2
from Crypto.Util.number import *
# 当e约去公约数后与phi互素
def decrypt(p, q, e, c):
    n = p * q
    phi = (p - 1) * (q - 1)
    t = gmpy2.gcd(e, phi)
    d = gmpy2.invert(e // t, phi)
    m = pow(c, d, n)
    msg = gmpy2.iroot(m, t)
    if msg[1]:
        print(long_to_bytes(msg[0]))

p= 13540627291583966394898250825916833919641342303370737735158271740813520116129194741169039807072513653441800075006852381645878610003713554206974980382580317624589966370001891820445790908293428678798457792081972207161432583211754994917638605557791766839271768364393327974197155313304496567221751595800601842520 7
q= 14149996777755469815782739858807319054604816114244237104331909179320215939293711731790931683002149273736901797425241294882487818200413243716587283676944232191985031274210566004860441962404283572352416239402475111512429494403506484997417885317393735452834730615296387016523054424102807140640940320044291046001
e= 894
c= 28559974064253189015422017559243784499999078040381563030766145900171317631761513862851614432541315323279681989780188110742586591305472895467735202745769931470241636001320502766050221008512560718117689068928596388232531147242268939746534967339141354828459257754456606907626686604793042753056632918392450627941697570155807444883582046212527297316729530405043456865211936635934057465948479380516470958503957453972270235271648022690005032266165001737988661439758553428503679954723761335655556280128950804016154708400036019313828109176059303015820063442721465546509760080534601397110717005135597191266323747240286658346 23
decrypt(p,q,e,c)
```

场景介绍

假设题目给出两组公钥n,e以及第一组、第二组加密后的密文

```
('n1=', '0xbf510b8e2b169fbce366eb15a4f6c71b370f02f2108c7feb482234a386185bce1a740fa6498e04edbdf2a639e320619d9f39d3e740ebaf578af0426bc3e851001a1d599108a08725347f6680a7f5581a32d91505023701872c3df723e8de9f201d3b17059bebff944b915045870d757eb6d6d009eb4561cc7e4b89968e4433a9L')
('e1=', '0x15d6439c6')
('c1=', '0x43e5cc4c99c3040aef2ccb0d4c45266f6b75cd7f9f1be105766689283f0886061c9cd52ac2b2b6c1b7d250c2079f354ca9b988db5556336201f3b5e489916b3b60b80c34bef8f608d7471fafaf14bee421b60630f42c5cc813356e786ff10e5efa334b8a73b7ea06afa6043f33be6a31010d306ba60516243add65c183da843aL')
('n2=', '0xba85d38d1bfc3fb281927c9246b5b771ac3344ca9fe1c2d9c793a886bffb5c84558f4a578cd5ba9e777a4e08f66d0cabe05b9aa2ae8d075778b5fbfff318a7f9b6f22e2eff6f79d8c1148941b3974f3e83a4a4f1520ad42336eddc572ec7ea04766eb798b2f1b1b52009b3eeea7741b2c55e3c7c11c5cf6a4e204c6b0d312f49L')
('e2=', '0x2c09848c6')
('c2=', '0x79ec6350649377f69b475eca83a7d9d5356a1d62e29933e9c8e2b19b4b23626a581037aba3be6d7f73d5bed049350e41c1ed4cdc3e10ee34ec576ef3449be2f7d930c759612e1c23c4db71d0e5185a80b548031e3857dd93eca4af017fcd25895fcc4e8a2b36c1dd36b8cd9cc9200e2879f025928fe346e2cfae5200e66de6ccL')
```

首先用公约数分解可以分解得到n1、n2的因子
但是发现e和φ(n)是不互为素数的，所以我们无法求出私钥d。

解题公式推导

```
gcd(e1,(p-1)*(q1-1))
gcd(e2,(p-1)*(q2-1))
```

得到结果为79858
也就是说，e和φ(n)不互素且具有公约数79858

1、首先我们发现n1、n2可以用公约数分解出p、q
但是由于e与φ(n)不互素，所以我们无法求解得到私钥d
只有当他们互素时，才能保证e的逆元d唯一存在。
公式推导过程参考博客
https://blog.csdn.net/chenzzhenguo/article/details/94339659

2、下面进行等式运算，来找到解题思路
还是要求逆元，则要找到与φ(n)互素的数

$$gcd(e,\varphi(n)) = b$$

$$ed \equiv 1 \bmod \varphi(n)$$

$$e = a * b$$

$$abd \equiv 1 \bmod \varphi(n)$$

$$mab \equiv c \bmod n$$

$$cbd \equiv mabbd \equiv mb \bmod n$$

我们已知b=79858
从上面的推算，可得a与φ(n)互素，于是可唯一确定bd
于是求出bd
gmpy2.invert(a,φ(n))
然后想到bd/b，求出d，然后求明文。可是，经测试求出的是乱码，这个d不是我们想要的

3、想一下，给两组数据，应该有两组数据的作用，据上面的结论，我们可以得到一个同余式组

$$res_1 \equiv m^{79858} \bmod n_1$$
$$res_2 \equiv m^{79858} \bmod n_2$$

进一步推导

$$res1 \equiv m^{79858} \bmod p$$
$$res1 \equiv m^{79858} \bmod q_1$$
$$res2 \equiv m^{79858} \bmod q_2$$

可以计算出特解m
m=solve_crt([m1,m2,m3], [q1,q2,p])
我们想到模n1,n2不行那模q1*q2呢，
这里res可取特值m

$$res \equiv m^{79858} \bmod q_1q_2$$

那么问题就转化为求一个新的rsa题目
e=79858，经计算发现此时e与φ(n)=(q1-1)(q2-1)，还是有公因数2。
那么，我们参照上述思路，可得出m^2，此时直接对m开方即可。

$$c \equiv m^e \bmod q_1q_2$$
$$e = 2*39929$$
$$2*39929*d \equiv 1 \bmod q_1q_2$$
$$m^2 \equiv c^{2d} \bmod q_1q_2$$

完整解题脚本

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
import gmpy2
import binascii


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

```
n1=0xbf510b8e2b169fbce366eb15a4f6c71b370f02f2108c7feb482234a386185bce1a740fa6498e04edbdf2a639e320619d9f39d3e740e
baf578af0426bc3e851001a1d599108a08725347f6680a7f5581a32d91505023701872c3df723e8de9f201d3b17059bebff944b915045870
d757eb6d6d009eb4561cc7e4b89968e4433a9
n2=0xba85d38d1bfc3fb281927c9246b5b771ac3344ca9fe1c2d9c793a886bffb5c84558f4a578cd5ba9e777a4e08f66d0cabe05b9aa2ae8
d075778b5fbfff318a7f9b6f22e2eff6f79d8c1148941b3974f3e83a4a4f1520ad42336eddc572ec7ea04766eb798b2f1b1b52009b3eeea7
741b2c55e3c7c11c5cf6a4e204c6b0d312f49

p=gcd(n1,n2)
q1=n1//p
q2=n2//p



c1=0x43e5cc4c99c3040aef2ccb0d4c45266f6b75cd7f9f1be105766689283f0886061c9cd52ac2b2b6c1b7d250c2079f354ca9b988db555
6336201f3b5e489916b3b60b80c34bef8f608d7471fafaf14bee421b60630f42c5cc813356e786ff10e5efa334b8a73b7ea06afa6043f33b
e6a31010d306ba60516243add65c183da843a
c2=0x79ec6350649377f69b475eca83a7d9d5356a1d62e29933e9c8e2b19b4b23626a581037aba3be6d7f73d5bed049350e41c1ed4cdc3e1
0ee34ec576ef3449be2f7d930c759612e1c23c4db71d0e5185a80b548031e3857dd93eca4af017fcd25895fcc4e8a2b36c1dd36b8cd9cc92
00e2879f025928fe346e2cfae5200e66de6cc
e1 =0x15d6439c6
e2 =0x2c09848c6

#print(gcd(e1,(p-1)*(q1-1)))
#print(gcd(e2,(p-1)*(q2-1)))


e1=e1//gcd(e1,(p-1)*(q1-1))
e2=e2//gcd(e2,(p-1)*(q2-1))


phi1=(p-1)*(q1-1);phi2=(p-1)*(q2-1)
d1=gmpy2.invert(e1,phi1)
d2=gmpy2.invert(e2,phi2)
f1=pow(c1,d1,n1)
f2=pow(c2,d2,n2)


def GCRT(mi, ai):
    curm, cura = mi[0], ai[0]
    for (m, a) in zip(mi[1:], ai[1:]):
        d = gmpy2.gcd(curm, m)
        c = a - cura
        K = c // d * gmpy2.invert(curm // d, m // d)
        cura += curm * K
        curm = curm * m // d
        cura %= curm
    return (cura % curm, curm)


f3,lcm = GCRT([n1,n2],[f1,f2])
n3=q1*q2
c3=f3%n3
phi3=(q1-1)*(q2-1)

d3=gmpy2.invert(39929,phi3)#39929是79858//gcd((q1-1)*(q2-1),79858) 因为新的e和ϕ(n)还是有公因数2
m3=pow(c3,d3,n3)

if gmpy2.iroot(m3,2)[1] == 1:
    flag=gmpy2.iroot(m3,2)[0]
    print(binascii.unhexlify(hex(flag)[2:].strip("L")))
```

0x09 公钥n由多个素数因子组成

场景介绍

题目如下

```
('n=', '0xf1b234e8a03408df4868015d654dcb931f038ef4fc0be8658c9b951ee6c60d23689a1bfb151e74df0910fa1cf8a542282a65')
('e=', '0x10001')
('c=', '0x22fda6137013bac19754f78e8d9658498017f05a4b0814f2af97dc2c60fdc433d2949ea27b13337961ef3c4cf27452ad3c95')
```

因为这题的公钥n是由四个素数相乘得来的，
其中四个素数的值相差较小，或者较大，都会造成n更容易分解的结果
例如出题如下

```
p=getPrime(100)
q=gmpy2.next_prime(p)
r=gmpy2.next_prime(q)
s=gmpy2.next_prime(r)
n=p*q*r*s
```

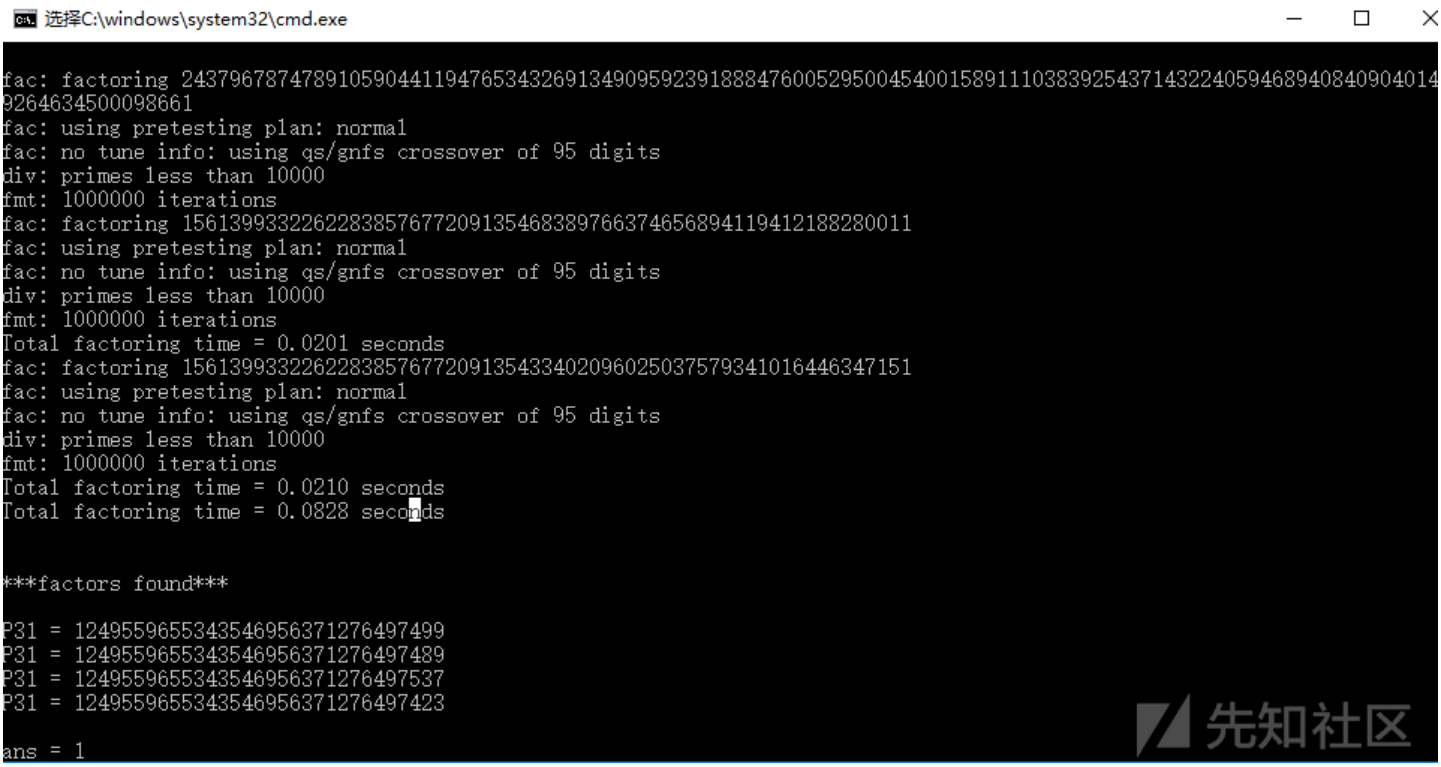
因为p、q、r、s十分接近，所以可以使用yafu直接分解

yafu分解

使用

```
factor(*)
```

括号中为要分解的数



公钥n由多素数相乘解题脚本

```
import binascii
import gmpy2
p=14qBrMavy2pW9umzuhd7eMDVpwQ62xRtPK
q=14qBrMavy2pW9umzuhd7eMDVpwQ62xRtPK
r=14qBrMavy2pW9umzuhd7eMDVpwQ62xRtPK
s=14qBrMavy2pW9umzuhd7eMDVpwQ62xRtPK
e=0x10001
c=0x22fda6137013bac19754f78e8d9658498017f05a4b0814f2af97dc2c60fdc433d2949ea27b13337961ef3c4cf27452ad3c95
n=p*q*r*s

phi=(p-1)*(q-1)*(r-1)*(s-1)
d=gmpy2.invert(e,phi)
m=pow(c,d,n)
print(binascii.unhexlify(hex(m)[2:].strip("L")))
```

0x10 小明文攻击

小明文攻击是基于低加密指数的，主要分成两种情况。

明文过小，导致明文的e次方仍然小于n

```
('n=', '0xad03794ef170d81aad370dccb7b92af7d174c10e0ae9ddc99b7dc5f93af6c65b51cc9c40941b002c7633caf8cd50e1b73aa942
c8488d46c0032064306de388151814982b6d35b4e2a62dd647f527b31b4f826c36848dc52999574a8694460e1b59b4e96bda1341d3ba5f99
1f0000a56004d47681ecfd37a5e64bd198617f8dadL')
('e=', '0x3')
('c=', '0x10652cdf6f422470ea251f77L')
```

这种情况直接对密文e次开方，即可得到明文

解题脚本

```
import binascii
import gmpy2
n=0xad03794ef170d81aad370dccb7b92af7d174c10e0ae9ddc99b7dc5f93af6c65b51cc9c40941b002c7633caf8cd50e1b73aa942c8488d
46c0032064306de388151814982b6d35b4e2a62dd647f527b31b4f826c36848dc52999574a8694460e1b59b4e96bda1341d3ba5f991f0000
a56004d47681ecfd37a5e64bd198617f8dad
e=0x3
c=0x10652cdf6f422470ea251f77

m=gmpy2.iroot(c, 3)[0]
print(binascii.unhexlify(hex(m)[2:].strip("L")))
```

明文的三次方虽然比n大，但是大不了多少

```
('n=', '0x9683f5f8073b6cd9df96ee4dbe6629c7965e1edd2854afa113d80c44f5dfcf030a18c1b2ff40575fe8e222230d7bb5b6dd8c41
9c9d4bca1a7e84440a2a87f691e2c0c76caaab61492db143a61132f584ba874a98363c23e93218ac83d1dd715db6711009ceda2a31820bba
caf1b6171bbaa68d1be76fe986e4b4c1b66d10af25L')
('e=', '0x3')
('c=', '0x8541ee560f77d8fe536d48eab425b0505e86178e6ffefa1b0c37ccbfc6cb5f9a7727baeb3916356d6fce3205cd4e586a1cc407
703b3f709e2011d7b66eaaeea9e381e595b4d515c433682eb3906d9870fadbffd0695c0168aa26447f7a049c260456f51e937ce75b74e5c3
c2bd7709b981898016a3a18f15ae99763ff40805aaL')
```

爆破即可，每次加上一个n

```
i = 0
while 1:
    res = iroot(c+i*n,3)
    if(res[1] == True):
        print res
        break
    print "i="+str(i)
    i = i+1
```

完整脚本

```
import binascii
import gmpy2

n=0x9683f5f8073b6cd9df96ee4dbe6629c7965e1edd2854afa113d80c44f5dfcf030a18c1b2ff40575fe8e222230d7bb5b6dd8c419c9d4b
ca1a7e84440a2a87f691e2c0c76caaab61492db143a61132f584ba874a98363c23e93218ac83d1dd715db6711009ceda2a31820bbacaf1b6
171bbaa68d1be76fe986e4b4c1b66d10af25
e=0x3
c=0x8541ee560f77d8fe536d48eab425b0505e86178e6ffefa1b0c37ccbfc6cb5f9a7727baeb3916356d6fce3205cd4e586a1cc407703b3f
709e2011d7b66eaaeea9e381e595b4d515c433682eb3906d9870fadbffd0695c0168aa26447f7a049c260456f51e937ce75b74e5c3c2bd77
09b981898016a3a18f15ae99763ff40805aa

i = 0
while 1:
    res = gmpy2.iroot(c+i*n,3)
    if(res[1] == True):
        m=res[0]
        print(binascii.unhexlify(hex(m)[2:].strip("L")))
        break
    print "i="+str(i)
    i = i+1
```

0x11 低加密指数广播攻击

场景介绍

如果选取的加密指数较低，并且使用了相同的加密指数给一个接受者的群发送相同的信息，那么可以进行广播攻击得到明文。
这个识别起来比较简单，一般来说都是给了三组加密的参数和明密文，其中题目很明确地能告诉你这三组的明文都是一样的，并且e都取了一个较小的数字。

```
('n=', '0x683fe30746a91545a45225e063e8dc64d26dbf98c75658a38a7c9dfd16dd38236c7aae7de5cbbf67056c9c57817fd3da79dc49
55217f43caefde3b56a46acf5dL', 'e=', '0x7', 'c=', '0x673c72ace143441c07cba491074163c003f1a550eab56b1255e5ea9fa2bb
d68fd6a9ccb48db9fd66d5dfc6a55c79cad3d9de53f700a1e3c2a29731dc56ba43cdL')
('n=', '0xa39292e6ad271bb6a2d1345940dfab8001a53d28bc7468f285d2873d784004c2653549c589dae91c6d8238977ff1c4bea4f17d
424a0fc4d5587661cc7dde3a77L', 'e=', '0x7', 'c=', '0x6111357d180d966a495f38566ebe4ea51fa0d54159b22bbd443cde938768
7d87c08638483b39221883453a5ad09f6a0e3726b214e8e333037d178a3d0f125343L')
('n=', '0x52c32366d84d34564a5fdc1650fc401c41ad2a63a2d6ef57c32c7887bb25da9d42c0acfb887c6334c938839c9a43aca93b2c74
68915d1846576f92c342046d1fL', 'e=', '0x7', 'c=', '0x26cd2225c0229b6a3f1d1d685e53d114aa3d792737d040fbc14189336ac1
2fb780872792b0c0b259847badffd1427897ede0d60247aa5e79633f27ccb43e7cc2L')
```

解题脚本

```
import binascii,gmpy2

n =  [
0x683fe30746a91545a45225e063e8dc64d26dbf98c75658a38a7c9dfd16dd38236c7aae7de5cbbf67056c9c57817fd3da79dc4955217f43
caefde3b56a46acf5d,
0xa39292e6ad271bb6a2d1345940dfab8001a53d28bc7468f285d2873d784004c2653549c589dae91c6d8238977ff1c4bea4f17d424a0fc4
d5587661cc7dde3a77,
0x52c32366d84d34564a5fdc1650fc401c41ad2a63a2d6ef57c32c7887bb25da9d42c0acfb887c6334c938839c9a43aca93b2c7468915d18
46576f92c342046d1f
]
c =  [
0x673c72ace143441c07cba491074163c003f1a550eab56b1255e5ea9fa2bbd68fd6a9ccb48db9fd66d5dfc6a55c79cad3d9de53f700a1e3
c2a29731dc56ba43cd,
0x6111357d180d966a495f38566ebe4ea51fa0d54159b22bbd443cde9387687d87c08638483b39221883453a5ad09f6a0e3726b214e8e333
037d178a3d0f125343,
0x26cd2225c0229b6a3f1d1d685e53d114aa3d792737d040fbc14189336ac12fb780872792b0c0b259847badffd1427897ede0d60247aa5e
79633f27ccb43e7cc2
]
def CRT(mi, ai):
    assert(reduce(gmpy2.gcd,mi)==1)
    assert (isinstance(mi, list) and isinstance(ai, list))
    M = reduce(lambda x, y: x * y, mi)
    ai_ti_Mi = [a * (M / m) * gmpy2.invert(M / m, m) for (m, a) in zip(mi, ai)]
    return reduce(lambda x, y: x + y, ai_ti_Mi) % M
e=0x7
m=gmpy2.iroot(CRT(n, c), e)[0]
print(binascii.unhexlify(hex(m)[2:].strip("L")))
```

0x12 低解密指数攻击

场景介绍

主要利用的是私钥d很小，表现形式一般是e很大

```
n = 92476066235238477726989531616164556648218671835712180569700997513016822051231157160894867998374473979253088
797677599481717599494576027819752790962179346 9
e = 27587468384672288862881213094354358587433516035212531881921186101712498639965289973292625430363076074737388 3
459357754943123330255004095032906863940320 69
```

攻击脚本

github上有开源的攻击代码https://github.com/pablocelayes/rsa-wiener-attack

求解得到私钥d

```
def rational_to_contfrac (x, y):
    '''
    Converts a rational x/y fraction into
    a list of partial quotients [a0, ..., an]
    '''
    a = x//y
    if a * y == x:
        return [a]
    else:
        pquotients = rational_to_contfrac(y, x - a * y)
        pquotients.insert(0, a)
        return pquotients
def convergents_from_contfrac(frac):
    '''
    computes the list of convergents
```

```
    using the list of partial quotients
    '''
    convs = [];
    for i in range(len(frac)):
        convs.append(contfrac_to_rational(frac[0:i]))
    return convs

def contfrac_to_rational (frac):
    '''Converts a finite continued fraction [a0, ..., an]
     to an x/y rational.
     '''
    if len(frac) == 0:
        return (0,1)
    elif len(frac) == 1:
        return (frac[0], 1)
    else:
        remainder = frac[1:len(frac)]
        (num, denom) = contfrac_to_rational(remainder)
        # fraction is now frac[0] + 1/(num/denom), which is
        # frac[0] + denom/num.
        return (frac[0] * num + denom, num)

def egcd(a,b):
    '''
    Extended Euclidean Algorithm
    returns x, y, gcd(a,b) such that ax + by = gcd(a,b)
    '''
    u, u1 = 1, 0
    v, v1 = 0, 1
    while b:
        q = a // b
        u, u1 = u1, u - q * u1
        v, v1 = v1, v - q * v1
        a, b = b, a - q * b
    return u, v, a

def gcd(a,b):
    '''
    2.8 times faster than egcd(a,b)[2]
    '''
    a,b=(b,a) if a<b else (a,b)
    while b:
        a,b=b,a%b
    return a

def modInverse(e,n):
    '''
    d such that de = 1 (mod n)
    e must be coprime to n
    this is assumed to be true
    '''
    return egcd(e,n)[0]%n

def totient(p,q):
    '''
    Calculates the totient of pq
    '''
    return (p-1)*(q-1)
```

```
def bitlength(x):
    '''
    Calculates the bitlength of x
    '''
    assert x >= 0
    n = 0
    while x > 0:
        n = n+1
        x = x>>1
    return n


def isqrt(n):
    '''
    Calculates the integer square root
    for arbitrary large nonnegative integers
    '''
    if n < 0:
        raise ValueError('square root not defined for negative numbers')

    if n == 0:
        return 0
    a, b = divmod(bitlength(n), 2)
    x = 2**(a+b)
    while True:
        y = (x + n//x)//2
        if y >= x:
            return x
        x = y


def is_perfect_square(n):
    '''
    If n is a perfect square it returns sqrt(n),

    otherwise returns -1
    '''
    h = n & 0xF; #last hexadecimal "digit"

    if h > 9:
        return -1 # return immediately in 6 cases out of 16.

    # Take advantage of Boolean short-circuit evaluation
    if ( h != 2 and h != 3 and h != 5 and h != 6 and h != 7 and h != 8 ):
        # take square root if you must
        t = isqrt(n)
        if t*t == n:
            return t
        else:
            return -1

    return -1

def hack_RSA(e,n):
    frac = rational_to_contfrac(e, n)
    convergents = convergents_from_contfrac(frac)

    for (k,d) in convergents:
        #check if d is actually the key
        if k!=0 and (e*d-1)%k == 0:
```

```
            phi = (e*d-1)//k
            s = n - phi + 1
            # check if the equation x^2 - s*x + n = 0
            # has integer roots
            discr = s*s - 4*n
            if(discr>=0):
                t = is_perfect_square(discr)
                if t!=-1 and (s+t)%2==0:
                    print("\nHacked!")
                    return d

def main():
    n = 924760662352384777269895316161645566482186718357121805697009975130168220512311571608948679983744739792530888797677599481717599494576027819752790962179346 9
    e = 275874683846722888628812130943543585874335160352125318819211861017124986399652899732926254303630760747373883459357754943123330255004095032906863940320 69
    d=hack_RSA(e,n)
    print ("d=")
    print (d)

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

0x13 共模攻击

场景介绍

识别：若干次加密，e不同，n相同，m相同。就可以在不分解n和求d的前提下，解出明文m。

```
('n=', '0xc42b9d872f8ecf90b4832199771bbd8d9bafb213747d905a644baa42144f316dc224e7914f8a5d361eeab930adf5ea7fbe1416e58b3fae34ca7e6d2a3145e04af02cf5a4f14539fff032bccd7bb9cf85b12d7d36dbc870b57e11aa5704304d08eff685fe4ccd707e308dfac6a1167d79199ffa9396c4f2efb4770256253d1407L')
('e1=', '0xc21000af014a98b2455dec479L')
('e2=', '0x9935842d63b75899ddd81b467L')
('c1=', '0xc0204d515a275954bbc8390d80efa1cca3bb29724ed7ba18f861913e28b6400298603b920d484284ad9c1c175587496300355395cb06b32603e779ec9b97f7eea6bb0de42c54f7f60e6e1171496efef0de8048e6074658084d080bd346db426888084e6dd45cb89b283247443de75328d47f9bd64adbd9be86043c6d13c7ed41L')
('c2=', '0xc4053ed3455c15174e5699ab6eb09b830a98b79e92e7518b713e828faca4d6d02306a65a8ec70893ca8a56943a7074e6de8649f099164cad33b8ca93fce1656f0712b990cce06642250c52a80d19c2afa94a4e158139028ac89c811e6be8d7b6984b6c1edcdd752e4955e3a6f1ab38cf2edb4474a80e03d6c313eb8ebf4e98ccL')
```

推导过程

```
首先，两个加密指数互质：
gcd(e1,e2)=1

即存在s1、s2使得：
s1+*e1+s2*e2=1

又因为：
c1≡m^e1 mod n
c2≡m mod n

代入化简可得：
c1^s1 * c2^s2 ≡ m mod n

即可求出明文
```

公式的python实现如下

```
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
s = egcd(e1, e2)
s1 = s[1]
s2 = s[2]
if s1<0:
   s1 = - s1
   c1 = modinv(c1, n)
elif s2<0:
   s2 = - s2
   c2 = modinv(c2, n)
m=(pow(c1,s1,n)*pow(c2,s2,n)) % n
```

完整解题脚本

```
import sys
import binascii
sys.setrecursionlimit(1000000)
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

c1=0xc0204d515a275954bbc8390d80efa1cca3bb29724ed7ba18f861913e28b6400298603b920d484284ad9c1c175587496300355395cb0
6b32603e779ec9b97f7eea6bb0de42c54f7f60e6e1171496efef0de8048e6074658084d080bd346db426888084e6dd45cb89b283247443de
75328d47f9bd64adbd9be86043c6d13c7ed41
n=0xc42b9d872f8ecf90b4832199771bbd8d9bafb213747d905a644baa42144f316dc224e7914f8a5d361eeab930adf5ea7fbe1416e58b3f
ae34ca7e6d2a3145e04af02cf5a4f14539fff032bccd7bb9cf85b12d7d36dbc870b57e11aa5704304d08eff685fe4ccd707e308dfac6a116
7d79199ffa9396c4f2efb4770256253d1407
e1=0xc21000af014a98b2455dec479
c2=0xc4053ed3455c15174e5699ab6eb09b830a98b79e92e7518b713e828faca4d6d02306a65a8ec70893ca8a56943a7074e6de8649f0991
64cad33b8ca93fce1656f0712b990cce06642250c52a80d19c2afa94a4e158139028ac89c811e6be8d7b6984b6c1edcdd752e4955e3a6f1a
b38cf2edb4474a80e03d6c313eb8ebf4e98cc
e2=0x9935842d63b75899ddd81b467

s = egcd(e1, e2)
s1 = s[1]
s2 = s[2]

if s1<0:
   s1 = - s1
   c1 = modinv(c1, n)
elif s2<0:
   s2 = - s2
   c2 = modinv(c2, n)
m=(pow(c1,s1,n)*pow(c2,s2,n)) % n
print(m)
print (binascii.unhexlify(hex(m)[2:].strip("L")))
```

0x14 Stereotyped messages攻击

场景介绍

```
('n=', '0xf85539597ee444f3fcad07142ecf6eaae5320301244a7cedc50b2beed7e60ffa11ccf28c1a590fb81346fb16b0cecd046a1f63
f0bf93185c109b8c93068ec02fL')
('e=', '0x3')
('c=', '0xa75c3c8a19ed9c911d851917e442a8e7b425e4b7f92205ca532a2ab0f5abe6cb86d164cc61374877f9e88e7bca606b43c79f1d
59deadfcc68c3db52e5fc42f0L')
('m=', '0x666c6167206973203a7465737431323131313131313131313133343536000000000000000000L')
```

给了明文的高位，可以尝试使用Stereotyped messages攻击
我们需要使用sage实现该算法
可以安装SageMath
或者在线网站https://sagecell.sagemath.org/

攻击脚本

```
e = 0x3
b=0x666c6167206973203a7465737431323131313131313131313133343536000000000000000000
n = 0xf85539597ee444f3fcad07142ecf6eaae5320301244a7cedc50b2beed7e60ffa11ccf28c1a590fb81346fb16b0cecd046a1f63f0bf
93185c109b8c93068ec02f
c=0xa75c3c8a19ed9c911d851917e442a8e7b425e4b7f92205ca532a2ab0f5abe6cb86d164cc61374877f9e88e7bca606b43c79f1d59dead
fcc68c3db52e5fc42f0
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))
```

可以求解出m的低位

0x15 Factoring with high bits known攻击

场景介绍

```
('n=', '0xb50193dc86a450971312d72cc8794a1d3f4977bcd1584a20c31350ac70365644074c0fb50b090f38d39beb366babd784d6555d
6de3be54dad3e87a93a703abddL')
('p=', '0xd7e990dec6585656512c841ac932edaf048184bac5ebf9967000000000000000L')
('e=', '0x3')
('c=', '0x428a95e5712e8aa22f6d4c39ee5ec85f422608c2f141abf22799c1860a5e343068ab55dfb5c99a7085714f4ce8950e85d8ed0a
11fce3516cf66a641dca8321eeL')
```

题目给出p的高位

攻击脚本

该后门算法依赖于Coppersmith partial information attack算法, sage实现该算法

```
p = 0xd7e990dec6585656512c841ac932edaf048184bac5ebf9967000000000000000
n = 0xb50193dc86a450971312d72cc8794a1d3f4977bcd1584a20c31350ac70365644074c0fb50b090f38d39beb366babd784d6555d6de3
be54dad3e87a93a703abdd

kbits = 60
PR.<x> = PolynomialRing(Zmod(n))
f = x + p
x0 = f.small_roots(X=2^kbits, beta=0.4)[0]
print "x: %s" %hex(int(x0))
p = p+x0
print "p: ", hex(int(p))
assert n % p == 0
q = n/int(p)
print "q: ", hex(int(q))
```

其中kbit是未知的p的低位位数
x0为求出来的p低位

0x16 Partial Key Exposure Attack

场景介绍

```
('n=', '0x56a8f8cbc72ff68e67c72718bd16d7e98150aea08780f6c4f532d20ca3c92a0fb07c959e008cbcbeac744854bc4203eb9b2996
e9cf630133bc38952a2c17c27dL')
('d&((1<<256)-1)=', '0x594b6c9631c4987f588399f22466b51fc48ed449b8aae0309b5736ef0b741893')
('e=', '0x3')
('c=', '0xca2841cbc52c8307e0f2c48f8b14bc0846ece4111453362e6aee4b81f44f2a14df1c58836d4937f3b868148140ee36e9a7e910
dd84c2dc869ead47711412038L')
```

题目给出一组公钥n,e以及加密后的密文
给私钥d的低位

攻击脚本

记N=pq为n比特RSA模数,e和d分别为加解密指数,v为p和q低位相同的比特数,即p≡qmod2v且p≠qmod2v+1.
1998年,Boneh、Durfee和Frankel首先提出对RSA的部分密钥泄露攻击:当v=1,e较小且d的低n/4比特已知时,存在关于n的多项式时间算法分解N.
2001年R.Steinfeld和Y.Zheng指出,当v较大时,对RSA的部分密钥泄露攻击实际不可行.

当v和e均较小且解密指数d的低n/4比特已知时,存在关于n和2v的多项式时间算法分解N.

```
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 =0x56a8f8cbc72ff68e67c72718bd16d7e98150aea08780f6c4f532d20ca3c92a0fb07c959e008cbcbeac744854bc4203eb9b2996e
9cf630133bc38952a2c17c27d
    e = 0x3
    d = 0x594b6c9631c4987f588399f22466b51fc48ed449b8aae0309b5736ef0b741893
    beta = 0.5
    epsilon = beta^2/7

    nbits = n.nbits()
    kbits = 255
    d0 = d & (2^kbits-1)
    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 hex(d)
    print hex(inverse_mod(e, (p-1)*(q-1)))
```

kbits是私钥d泄露的位数255

0x17 Padding Attack

场景介绍

```
('n=', '0xb33aebb1834845f959e05da639776d08a344abf098080dc5de04f4cbf4a1001dL')
('e=', '0x3')
('c1=pow(hex_flag,e,n)', '0x3aa5058306947ff46b0107b062d75cf9e497cdb1f120d02eaeca30f76492c550L')
('c2=pow(hex_flag+1,e,n)', '0x6a645739f25380a5e5b263ff5e5b4b9324381f6408a11fdaab0488209145fb3eL')
```

原理参考

https://www.anquanke.com/post/id/158944

意思很简单
1.pow(mm, e) != pow(mm, e, n)
2.利用rsa加密m+padding
值得注意的是，e=3，padding可控
那么我们拥有的条件只有
n,e,c,padding
所以这里的攻击肯定是要从可控的padding入手了

攻击脚本

```
import gmpy
def getM2(a,b,c1,c2,n,e):
    a3 = pow(a,e,n)
    b3 = pow(b,e,n)
    first = c1-a3*c2+2*b3
    first = first % n
    second = e*b*(a3*c2-b3)
    second = second % n
    third = second*gmpy.invert(first,n)
    third = third % n
    fourth = (third+b)*gmpy.invert(a,n)
    return fourth % n
e=0x3
a=1
b=-1
c1=0x3aa5058306947ff46b0107b062d75cf9e497cdb1f120d02eaeca30f76492c550
c2=0x6a645739f25380a5e5b263ff5e5b4b9324381f6408a11fdaab0488209145fb3e
padding2=1
n=0xb33aebb1834845f959e05da639776d08a344abf098080dc5de04f4cbf4a1001d
m = getM2(a,b,c1,c2,n,e)-padding2
print hex(m)
```

通过上面介绍的那篇文章的推导过程我们可以知道
a等于1
b=padding1-padding2
这边我们的padding1是第一个加密的明文与明文的差，本题是0
padding2是第二个加密的明文与明文的差，本题是1
所以b是-1
我们这边是用的那篇文章的Related Message Attack

0x18 RSA LSB Oracle Attack

场景介绍

参考博客https://www.sohu.com/a/243246344_472906
适用情况：可以选择密文并泄露最低位。
在一次RSA加密中，明文为m，模数为n，加密指数为e，密文为c。
我们可以构造出c'=((2^e)*c)%n=((2e)(m^e))%n=((2*m)e)%n， 因为m的两倍可能大于n，所以经过解密得到的明文是 m'=(2m)%n 。
我们还能够知道 m' 的最低位lsb 是1还是0。
因为n是奇数，而2m是偶数，所以如果lsb 是0，说明(2m)%n 是偶数，没有超过n，即m<n/2.0，反之则m>n/2.0 。
举个例子就能明白2%3=2 是偶数，而4%3=1 是奇数。
以此类推，构造密文c"=(4^e)c)%n 使其解密后为m"=(4m)%n ，判断m" 的奇偶性可以知道m 和 n/4 的大小关系。
所以我们就有了一个二分算法，可以在对数时间内将m的范围逼近到一个足够狭窄的空间。

攻击脚本

```
def brute_flag(encrypted_flag, n, e):

    flag_count = n_count = 1
    flag_lower_bound = 0
    flag_upper_bound = n
    ciphertext = encrypted_flag
    mult = 1
    while flag_upper_bound > flag_lower_bound + 1:
        sh.recvuntil("input your option:")
        sh.sendline("D")
        ciphertext = (ciphertext * pow(2, e, n)) % n
        flag_count *= 2
        n_count = n_count * 2 - 1

        print("bit = %d" % mult)
        mult += 1



        sh.recvuntil("Your encrypted message:")
        sh.sendline(str(ciphertext))

        data=sh.recvline()[:-1]
        if(data=='The plain of your decrypted message is even!'):
            flag_upper_bound = n * n_count / flag_count
        else:
            flag_lower_bound = n * n_count / flag_count
            n_count += 1
    return flag_upper_bound
```

e很小

e=3 小公钥指数攻击 小明文攻击

```
import gmpy2
import binascii
enf = open('flag.enc', 'rb')
c = int(binascii.b2a_hex(enf.read()), 16)
print c
e = 3
n = 0x00b0bee5e3e9e5a7e8d00b493355c618fc8c7d7d03b82e409951c182f398dee3104580e7ba70d383ae5311475656e8a964d380cb15
7f48c951adfa65db0b122ca40e42fa709189b719a4f0d746e2f6069baf11cebd650f14b93c977352fd13b1eea6d6e1da775502abff89d3a8
b3615fd0db49b88a976bc20568489284e181f6f11e270891c8ef80017bad238e363039a458470f1749101bc29949d3a4f4038d4639388515
79c7525a69984f15b5667f34209b70eb261136947fa123e549dfff00601883afd936fe411e006e4e93d1a00b0fea541bbfc8c5186cb62205
03a94b2413110d640c77ea54ba3220fc8f4cc6ce77151e29b3e06578c478bd1bebe04589ef9a197f6f806db8b3ecd826cad24f5324ccdec6
e8fead2c2150068602c8dcdc59402ccac9424b790048ccdd9327068095efa010b7f196c74ba8c37b128f9e1411751633f78b7b9e56f71f77
a1b4daad3fc54b5e7ef935d9a72fb176759765522b4bbc02e314d5c06b64d5054b7b096c601236e6ccf45b5e611c805d335dbab0c35d226c
c208d8ce4736ba39a0354426fae006c7fe52d5267dcfb9c3884f51fddfdf4a9794bcfe0e1557113749e6c8ef421dba263aff68739ce00ed8
0fd0022ef92d3488f76deb62bdef7bea6026f22a1d25aa2a92d124414a8021fe0c174b9803e6bb5fad75e186a946a17280770f1243f43874
46ccceb2222a965cc30b3929L
k = 99999999
while True:
    k += 1
    if k % 1000000 == 0:
        print k
    res, isInt = gmpy2.iroot(c + k * n, e)
    if isInt:
        print k, res
        break
```

e=3 知道c n e利用多线程可以快速跑出来

```
import multiprocessing
import gmpy2
import binascii
c = 22408185322784300053248993692151779137140970318280573606074881120426854604326956262661356102856114055354085190341424057585533037021821906784112 5
e = 3
n = 19129746300443748089744511925693947853301476248004357669024520036817158717601179210185436310254597703793320421557022922735601195628135502538439530832496047160937050406231131694714387997658424397867926671783362 09
mod = c % n
def run(i, j):
  for k in range(i, j):
    res, is_exact = gmpy2.iroot(k * n + mod, 3)
    if is_exact:
      print(k, res)
      b = hex(res)
      b = b[2:]
      flag = binascii.a2b_hex(b)
      print (flag)


for i in range(0, 100000000, 18750000):
  try:
    p = multiprocessing.Process(target = run, args = (i, i + 18750000))
    p.start()
  except:
    continue
```

e = 2 rabin RSA

```
# coding=utf-8
import gmpy2
import string
with open('flag.enc', 'r') as f:
    cipher = f.read().encode('hex')
    cipher = string.atoi(cipher, base=16)
N=0xC2636AE5C3D8E43FFB97AB09028F1AAC6C0BF6CD3D70EBCA281BFFE97FBE30DD
p = 319576316814478949870590164193048041239
q = 275127860351348928173285174381581152299
# 计算yp和yq
inv_p = gmpy2.invert(p, q)
inv_q = gmpy2.invert(q, p)
# 计算mp和mq
mp = pow(cipher, (p + 1) / 4, p)
mq = pow(cipher, (q + 1) / 4, q)
# 计算a,b,c,d
a = (inv_p * p * mq + inv_q * q * mp) % N
b = N - int(a)
c = (inv_p * p * mq - inv_q * q * mp) % N
d = N - int(c)
for i in (a, b, c, d):
    s = '%x' % i
    if len(s) % 2 != 0:
        s = '0' + s
    print s.decode('hex')
```

e很大 维纳攻击

e较大 n也非常大 不需要 分解n得到p和q 直接可以求d

```
from RSAwienerHacker import hack_RSA
e=354611102441307572056572181827925899198345350228753730931089393275463916544456626894245415096107834465778409532373187125318554614722599301791528916212839368121066035541008808261534500586023652767712271625785204280964688004680328300124849680477105302519377370092578107827116821391826210972320377614967547827619
n=460657813884289609896372056585544172485318117026246263899744329237492701820627219556007788200590119136173895989001382151536006853823326382892363143604314518686388786002989248800814861248595075326277099645338694977097459168530898776007293695728101976069423971696524237755227187061418202849911479124793990722597
d=hack_RSA(e,n)
c=382309913162293996518235675906923010600446204121917377646323846805462562284515182388429652213947118483378324594438444468894683621541882148407367446578858589438101776758719911114666531582571911396056999163473082949956645302808168504827405306022545591237591211063383592202426377759190269335633260694494243911 92
m=pow(c ,d ,n)
print hex(m)
```

e = 3，m1和m2只差1

题目：

```
from Crypto.Util.number import *
flag = b'**************************************'
p=getPrime(256)
q=getPrime(256)
n=p*q
e=3
c1=pow(bytes_to_long(flag),e,n)
c2=pow(bytes_to_long(flag)+1,e,n)
print("n=",n)
print("e=",e)
print("c1=",c1)
print("c2=",c2)
# n = 420442077361747994356485916728682113300922362780417257326359011778562271852516123659723339843940210082627219095721846478625969263280495551697947188479617 1
# e = 3
# c1 =247298053457628139255888647694054941115154174139543503517821606705842427457919986048213134098664321411469117276352923183237332360061264585656418599864426 6
# c2 =318704993781182337396532094613621984050007025549122207730381779552775024105357695776796531342045645898375985111061569631477338013273201711520253285599699 9
```

exp:

```
import gmpy2
from Crypto.Util.number import *
n = 42044207736174799435648591672868211330092236278041725732635901177856227l
e = 3
c1 =24729805345762813925588864769405494111515417413954350351782160670584242l
c2 =31870499378118233739653209461362198405000702554912220773038177955277502l
def get_m(a, b, c1, c2, n):
    a3 = pow(a, 3, n)
    b3 = pow(b, 3, n)
    tmp1 = ((c2 + 2*a3*c1 - b3) * b) % n
    tmp2 = ((c2 - a3*c1 + 2*b3) * a) % n
    tmp3 = gmpy2.invert(tmp2, n)
    tmp4 = (tmp1 * tmp3) % n
    return tmp4
m = get_m(1, 1, c1, c2, n)
print(long_to_bytes(m))
```

已知e,d,n求p,q

```
import random
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 = 0xa66791dc6988168de7ab77419bb7fb0c001c62710270075142942e19a8d8c51d053b3e3782a1de5dc5af4ebe99468170114a1d
fe67cdc9a9af55d655620bbab
    e =  0x10001
    d =  0x123c5b61ba36edb1d3679904199a89ea80c09b9122e1400c09adcf7784676d01d23356a7d44d6bd8bd50e94bfc723fa87d886
2b75177691c11d757692df8881
    p,q = getpq(n,e,d)
    print "p: "+hex(p)
    print "q: "+hex(q)

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

给出p+q、p-q、e、c

```
#!/usr/bin/python2
import gmpy2
from Crypto.Util import number
e=12820879
paq=220355386708890057634113463981884498289112848403453281602619133132269222439036401860510043331841759349855907
384879707829508507698890173017385793207675636O4
psq=-261668774009884829653185668154902877376150089546663557561104272531824938594231997862458058981861163241120 85
412374332349572857756846151405410314248589276l8
c=634298970012358425967331187563868817801648987820468814505528165494017788917934594800508560416911989318916318 50
185841945327259580257701334694328660248355158394695998869152900121288763261566072460165128360649173228270012193 33
590596015358650080294864951361180699384866956574079340835286443551749086639650151612 31
p = (paq+psq)/2
q = (paq-psq)/2
# print p
# print q
d = gmpy2.invert(e, (p-1)*(q-1))
# print d
m = pow(c, d, p*q)
print( number.long_to_bytes(m) )
```

e,m相同，存在两个n有公约数

```
import gmpy2
from gmpy2 import invert, iroot
import gmpy2 as gp
from libnum import xgcd, invmod

n=[,,,,,,,,,,,,,,,,,,]
for i in n:
    for j in n:
        if (i<>j):
            pub_p=gmpy2.gcdext(i,j)
            if (pub_p[0]<>1)&(i>j):
                print i
                print j
                print pub_p[0]
                a=i,p=pub_p[0]
q=a/p
p =  gp.mpz()
q =  gp.mpz()
e =  gp.mpz()
c =  gp.mpz()
n = p*q
phi = (p-1) * (q-1)
d = gp.invert(e, phi)
m = pow(c, d, n)
print hex(m)
```

3个n 3个c

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

def chinese_remainder(n, a):
    sum = 0
    prod = reduce(lambda a, b: a * b, n)
    for n_i, a_i in zip(n, a):
        p = prod // n_i
        sum += a_i * gmpy2.invert(p, n_i) * p
    return int(sum % prod)
n1=78642188663937191491235684351005990853149481644703243255021321296087539054265733392095095639539412823093600710316645130404423641473150336492175402885270861906530337207734106926328737198871118125840680572148601743121884788919989184318198417654263598170932154428514561079675550090698019678767738203477097731989
c1=23419685303892339080979695469481275906709035609088426118328601771163101123641599051556995351678670765521269546319724616458499631461037359417701720430452076029312714313804716888119910334476982840024696320503747736428099717113471541651211596481005191146454458591558743268791485623924245960696651150688621664860
n2=98174485544103863705821086588292917749386955237408645745685476234349659452606822650329076955303471252833860010724515777826660887118742978051231030080666542833950748806944312437614585352818344599399156268450521239843157288915059003487783576003027303399985723834248634230998110618288843582573006048070816520647
c2=72080679612442543693944655041130370753964497034378634203383617624269927191363529233872659451561571441107920350406295389613006330637565645758727103723546610079332161151567096389071050158035757745766399510575237344950873632114050632573903701015749830874081198250578516967517980592506626547273178363503100507676
n3=91638855323231795590642755267985988356764327384001022396221901964430032527111968159623063760057482761918901490239790230176524505469897183382928646349163030620342744192731246392941227433195249399795012672172947919435254998997253131826888070173526892674308708289629739522194864912899817994807268945141349669311
c3=22149989692509889061584875630258740744292355239822482581889060656197919681655781672277545701325284646570773490123892626601106871432216449814891757715588851851459306683123591338089745675044763551335899599807235257516935037356212345033087798267959242561085752109746935300735969972249665700075907145744305255616

n=[n1,n2,n3]
c=[c1,c2,c3]
ans=chinese_remainder(n, c)
ans=gmpy2.iroot(ans,3)[0] # e = 3
print(long_to_bytes(ans))
```

c1,c2,e1,e2,n求明文m

c给的是私钥文件

```
from Crypto.Util.number import long_to_bytes, bytes_to_long
from gmpy2 import gcdext,invert

n=0x00b0bee5e3e9e5a7e8d00b493355c618fc8c7d7d03b82e409951c182f398dee3104580e7ba70d383ae5311475656e8a964d380cb157f
48c951adfa65db0b122ca40e42fa709189b719a4f0d746e2f6069baf11cebd650f14b93c977352fd13b1eea6d6e1da775502abff89d3a8b3
615fd0db49b88a976bc20568489284e181f6f11e270891c8ef80017bad238e363039a458470f1749101bc29949d3a4f4038d463938851579
c7525a69984f15b5667f34209b70eb261136947fa123e549dfff00601883afd936fe411e006e4e93d1a00b0fea541bbfc8c5186cb6220503
a94b2413110d640c77ea54ba3220fc8f4cc6ce77151e29b3e06578c478bd1bebe04589ef9a197f6f806db8b3ecd826cad24f5324ccdec6e8
fead2c2150068602c8dcdc59402ccac9424b790048ccdd9327068095efa010b7f196c74ba8c37b128f9e1411751633f78b7b9e56f71f77a1
b4daad3fc54b5e7ef935d9a72fb176759765522b4bbc02e314d5c06b64d5054b7b096c601236e6ccf45b5e611c805d335dbab0c35d226cc2
08d8ce4736ba39a0354426fae006c7fe52d5267dcfb9c3884f51fddfdf4a9794bcfe0e1557113749e6c8ef421dba263aff68739ce00ed80f
d0022ef92d3488f76deb62bdef7bea6026f22a1d25aa2a92d124414a8021fe0c174b9803e6bb5fad75e186a946a17280770f1243f4387446
ccceb2222a965cc30b3929
e1=17
e2=65537
f1=open(r'flag.enc1','rb')
f2=open(r'flag.enc2','rb')
c1=bytes_to_long(f1.read())
c2=bytes_to_long(f2.read())
print c1
print c2
(tmp,s1,s2)=gcdext(e1,e2)
if s1<0:
    s1=-s1
    c1=invert(c1,n)
else:
    s2=-s2
    c2=invert(c2,n)
m=(pow(c1,s1,n)*pow(c2,s2,n))%n
print(long_to_bytes(m))
```

两个e

eg:已知p ^ 2 + q ^ 2和pq，联立方程组可解出p，q

此外本题每访问一次都会给一个随机的e和c，但是p ^ 2 + q ^ 2和pq不变，因此可以考虑共模攻击

```
#!/usr/bin/python
from gmpy2 import invert
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)
c1 = 59877139582687352143541548034973171251433089217522033076662312278537223143367839741257489018787176085179836
44756468600554724706931820604405809671267324088287648700075363357695118762608245205739707554231584617488361207036
60504152024656969166179317751436257687121611066149319910101207191417262758066281544472140
e1 = 15533099
c2 = 44371570621538697766905202215101491859832484171964459710651473062538547875233852066369510822575533385370219
53503927191098883887433368047224410015223628824803415933860404116570630779675498114063858328227239158304614959826
83861052795103644456671970927564437114146569661969695948007095524971989600804533797881442
e2 = 13190117
A = 18332234375237505797544828779906069683502850019432756336685642353814494664629249967435614021177566249713325382669056095396545736811010849764125446099799455484068167330820319920874190947249040989780074894509246090037855552362895946210214268288154664244838705559409467076188500317689303774159559710200627758805 0
B = -50864244634108621571921738287685924188849738854375380135025289437002856457638682228327033386793089351328391948021747712241952777471385696279849869973597 6
n = (A-B**2)/2
# print n
s = egcd(e1, e2)
s1 = s[1]
s2 = s[2]
if s1<0:
    s1 = - s1
    c1 = invert(c1, n)
elif s2<0:
    s2 = - s2
    c2 = invert(c2, n)
m = pow(c1,s1,n)*pow(c2,s2,n) % n
print hex(m)[2:].decode('hex')
```

附录

RSA PEM文件格式

1.PKCS#1私钥格式文件

```
-----BEGIN RSA PRIVATE KEY-----
-----END RSA PRIVATE KEY-----
```

2.PKCS#8私钥格式文件

```
-----BEGIN  PRIVATE KEY-----
-----END PRIVATE KEY-----
```

3. PEM公钥格式文件

```
-----BEGIN PUBLIC KEY-----
-----END PUBLIC KEY-----
```

4. PEM RSAPublicKey公钥格式文件

```
-----BEGIN RSA PUBLIC KEY-----
-----END RSA PUBLIC KEY-----
```

参考链接

有关密码学的知识可以看WIKI：

> https://ctf-wiki.org/crypto/introduction
> https://www.jianshu.com/p/c945b0f0de0a