攻防世界 crypto 入门题之easy_RSA

原创

沐一 · 林 于 2021-08-07 11:06:07 发布 503 收藏 3

分类专栏： CTF 密码学 文章标签： unctf

本文链接：https://blog.csdn.net/xiao__1bai/article/details/119465149

版权

CTF 同时被 2 个专栏收录

167 篇文章 6 订阅

订阅专栏

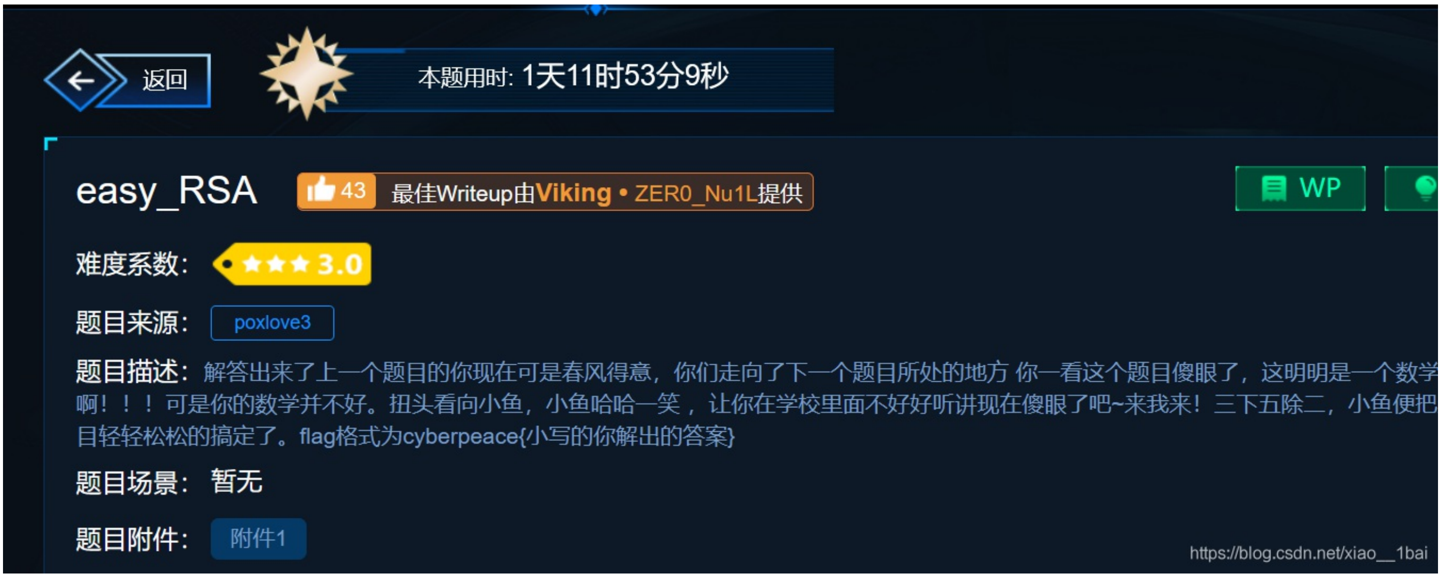
密码学

51 篇文章 1 订阅

订阅专栏

攻防世界 crypto 入门题之easy_RSA

继续开启全栈梦想之逆向之旅~
这题是攻防世界crypto 入门题之easy_RSA



本题用时: 1天11时53分9秒

easy_RSA

最佳Writeup由Viking • ZER0_Nu1L提供

难度系数: ★★★ 3.0

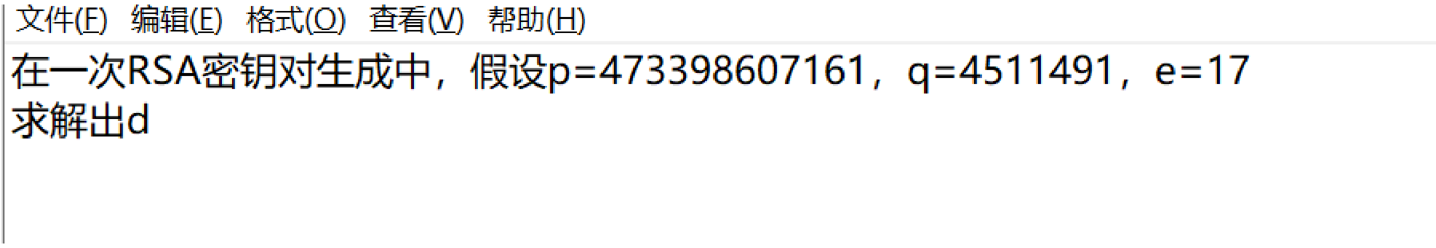
题目来源: poxlove3

题目描述: 解答出来了上一个题目的你现在可是春风得意，你们走向了下一个题目所处的地方 你一看这个题目傻眼了，这明明是一个数学啊！！！可是你的数学并不好。扭头看向小鱼，小鱼哈哈一笑，让你在学校里面不好好听讲现在傻眼了吧~来我来！三下五除二，小鱼便把目轻轻松松的搞定了。flag格式为cyberpeace{小写的你解出的答案}

题目场景: 暂无

题目附件: 附件1

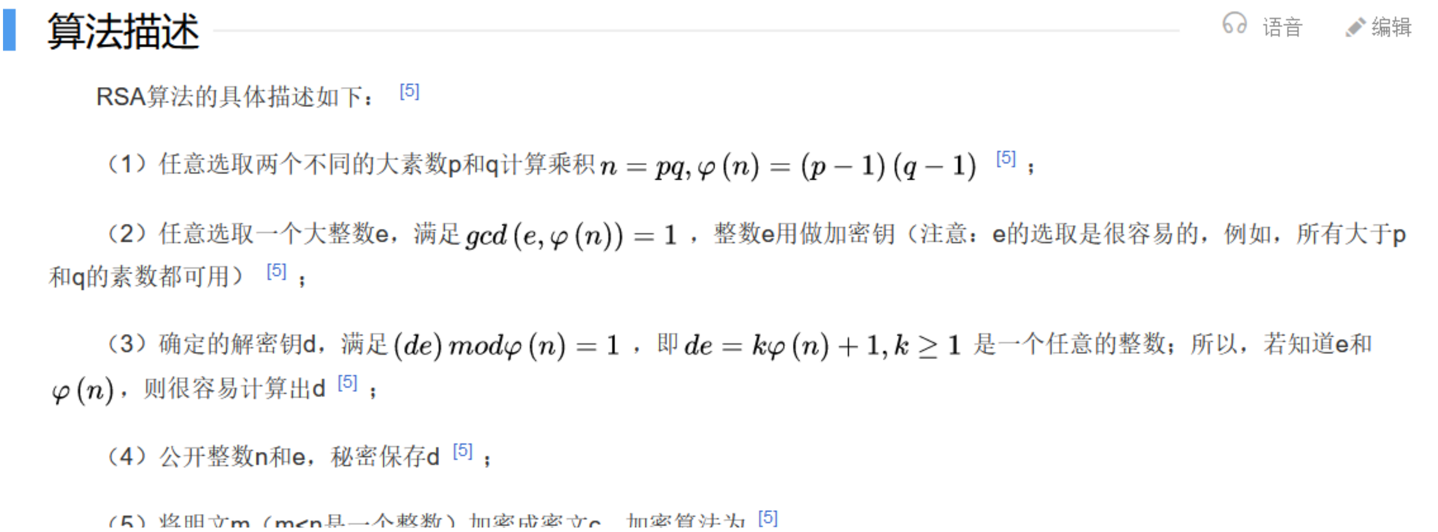
RSA的密码学听说了好久，主要是战队的队友之前有研究，而我却是一点都不了解，这次遇到了，就研究一下做题方法和技巧，密码学目前是不打算深究了，毕竟数学也不太好，所以我现在的目的就是懂得用对应的脚本来解这类RSA题。
附件下下来后就两行字：



在一次RSA密钥对生成中，假设p=473398607161，q=4511491，e=17
求解出d

看着都不难，但说实话我连RSA是什么都不知道，查了下百度，根据百度百科的说法：

> RSA公开密钥密码体制是一种使用不同的加密密钥与解密密钥，“由已知加密密钥推导出解密密钥在计算上是不可行的”密码体制 [2] 。
> 在公开密钥密码体制中，加密密钥（即公开密钥）PK是公开信息，而解密密钥（即秘密密钥）SK是需要保密的。加密算法E和解密算法D也都是公开的。虽然解密密钥SK是由公开密钥PK决定的，但却不能根据PK计算出SK [2] 。
> RSA公开密钥密码体制的原理是：根据数论，寻求两个大素数比较简单，而将它们的乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥 [4] 。



算法描述

RSA算法的具体描述如下： [5]

（1）任意选取两个不同的大素数p和q计算乘积 $n = pq, \varphi(n) = (p-1)(q-1)$ [5] ；

（2）任意选取一个大整数e，满足 $gcd(e, \varphi(n)) = 1$ ，整数e用做加密钥（注意：e的选取是很容易的，例如，所有大于p和q的素数都可用） [5] ；

（3）确定的解密钥d，满足 $(de) mod\varphi(n) = 1$ ，即 $de = k\varphi(n) + 1, k \geq 1$ 是一个任意的整数；所以，若知道e和 $\varphi(n)$，则很容易计算出d [5] ；

（4）公开整数n和e，秘密保存d [5] ；

（5）将明文m（m<n是一个整数）加密成密文c，加密算法为 [5]

（5）将明文m（m<n是一个整数）加密成密文c，加密算法为

$$c = E(m) = m^e modn$$

（6）将密文c解密为明文m，解密算法为 [5]

$$m = D(c) = c^d modn$$

然而只根据n和e（注意：不是p和q）要计算出d是不可能的。因此，任何人都可对明文进行加密，但只有授权用户（知道d）才可对密文解密 [5] 。

算法攻击

语音 编辑

迄今为止，对RSA的攻击已经很多，但都没有对它构成真正的威胁。在这里，我们讨论一些典型的攻击方法 [8] 。

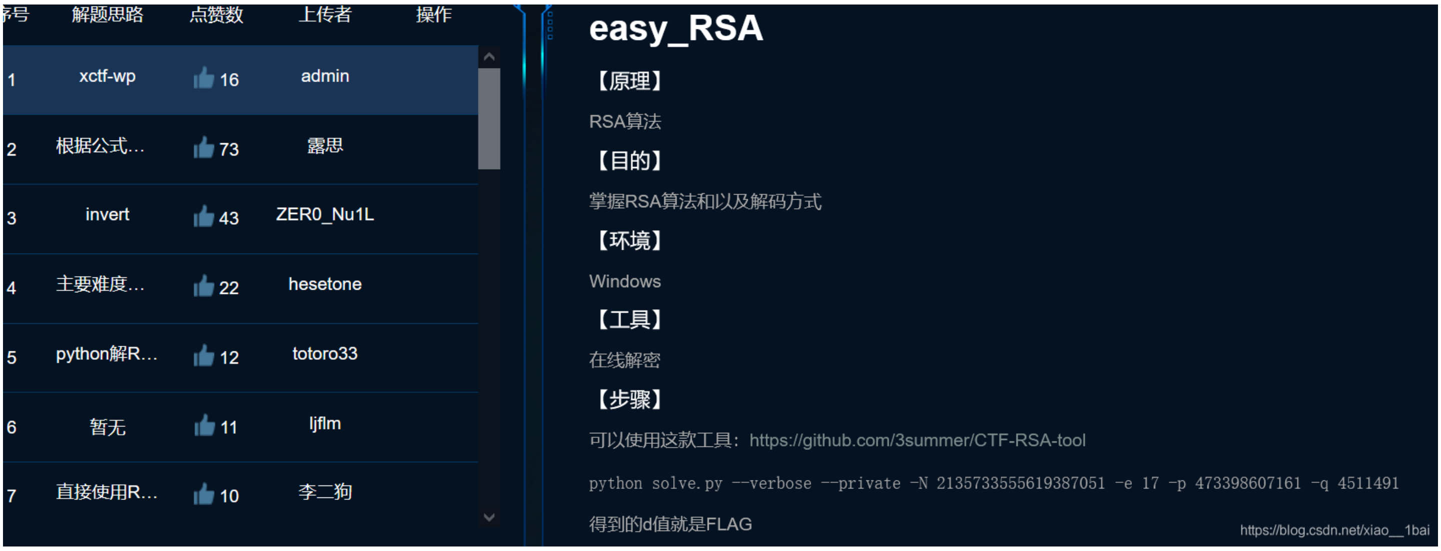
RSA的选择密码攻击

RSA在选择密码攻击面前显得很脆弱。一般攻击者是将某一信息进行下伪装，让拥有私钥的实体签名；然后，经过计算就可得到它所想要的信息。实际上，攻击利用的都是同一个弱点，即存在这样一个事实：乘幂保留了输入的乘法结构。前面已经提到，这个固有的问题来自于公钥密码系统的最基本的特征，即每个人都能使用公钥加密信息。从算法上无法解决这一问题，改进措施有两条：是采用好的公钥协议保证工作过程中实体不对其他实体任意产生的信息解密，不对自己一无所知的信息签名；二是决不对陌生人送来的随机文档签名，或签名时首先对文档作Hash处理，或同时使用不同的签名算法 [8] 。

RSA的小指数攻击

当公钥e取较小的值，虽然会使加密变得易于实现，速度有所提高，但这样做也是不安全的。最简单的办法就是e和d都取较大的值 [8] 。

不知道你看不看得懂，反正我是大概看懂了，d通常是我们要求的。接下来就是重头戏了，RSA的脚本工具，查看官方admin的WP：



序号	解题思路	点赞数	上传者	操作
1	xctf-wp	16	admin	
2	根据公式...	73	露思	
3	invert	43	ZER0_Nu1L	
4	主要难度...	22	hesetone	
5	python解R...	12	totoro33	
6	暂无	11	ljflm	
7	直接使用R...	10	李二狗	

easy_RSA

【原理】

RSA算法

【目的】

掌握RSA算法和以及解码方式

【环境】

Windows

【工具】

在线解密

【步骤】

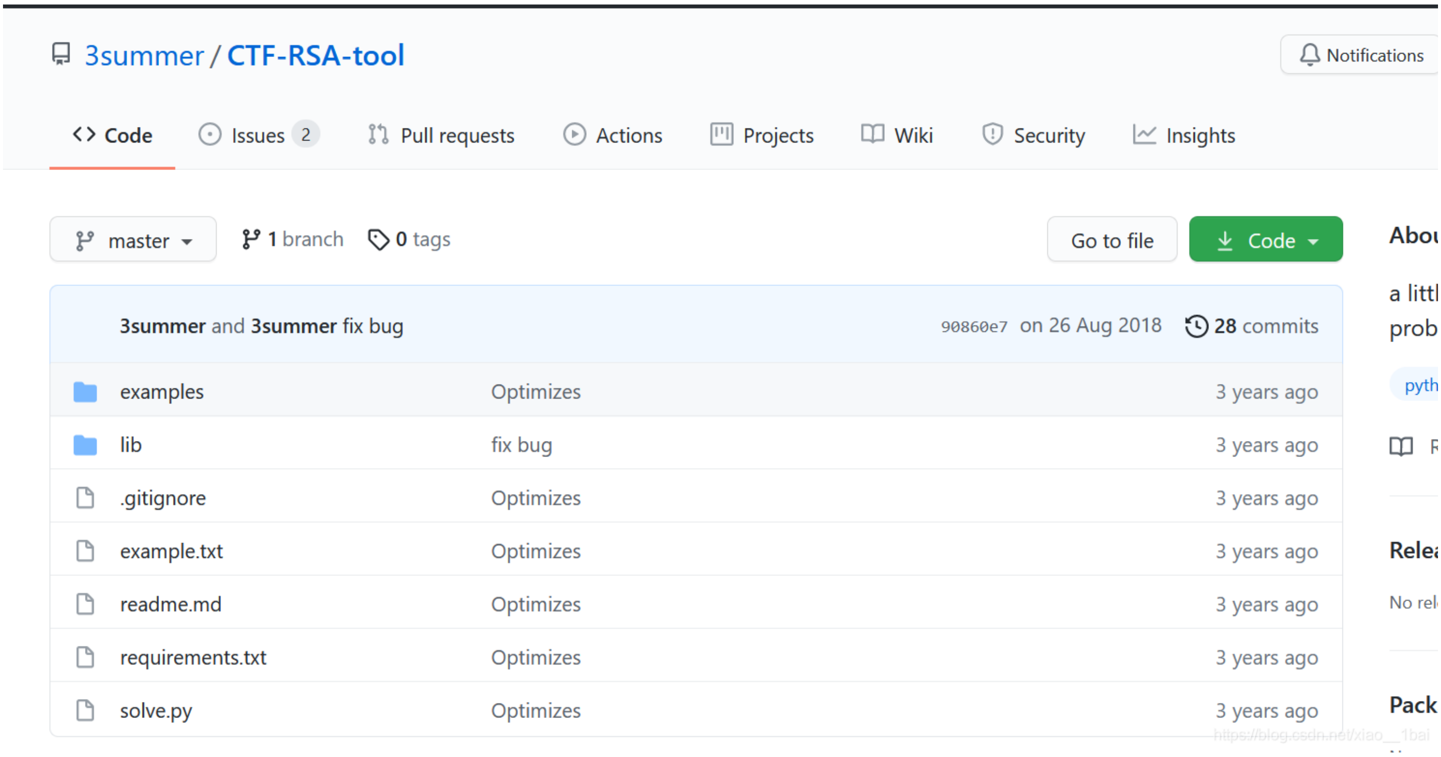
可以使用这款工具：https://github.com/3summer/CTF-RSA-tool

```
python solve.py --verbose --private -N 2135733555619387051 -e 17 -p 473398607161 -q 4511491
```

得到的d值就是FLAG

锁定了我要找的工具：

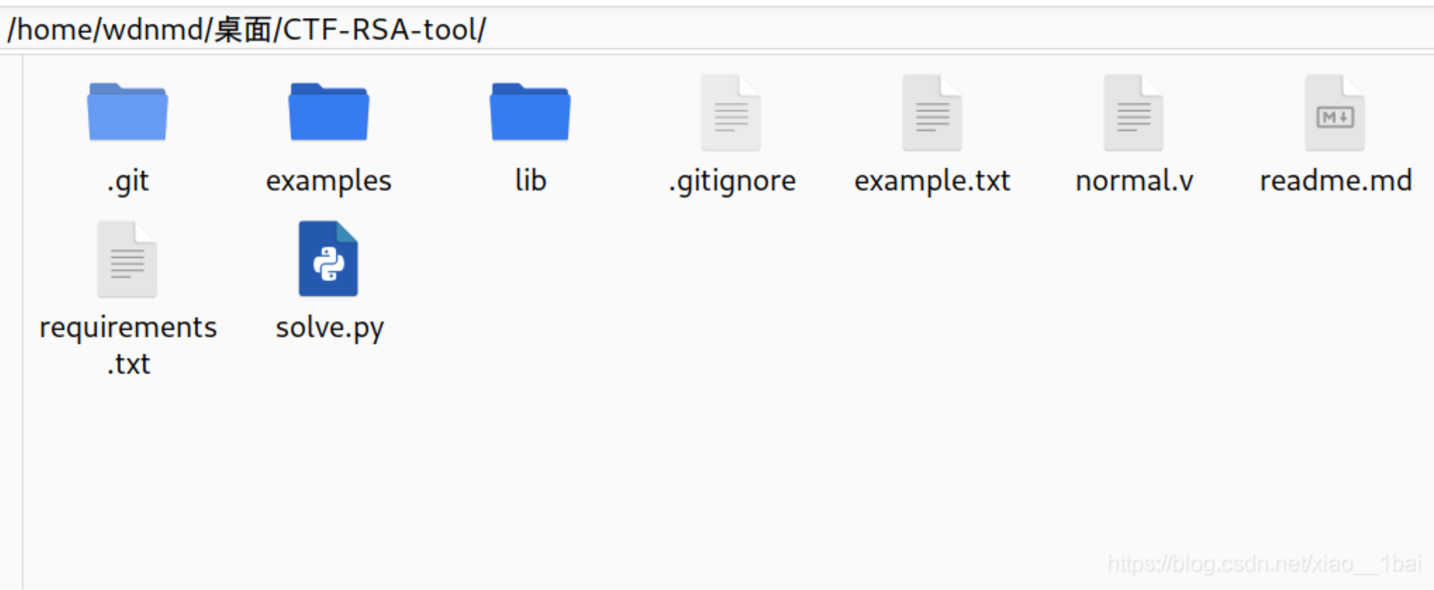
```
https://github.com/3summer/CTF-RSA-tool
```



3summer and 3summer fix bug 90860e7 on 26 Aug 2018 28 commits

examples	Optimizes	3 years ago
lib	fix bug	3 years ago
.gitignore	Optimizes	3 years ago
example.txt	Optimizes	3 years ago
readme.md	Optimizes	3 years ago
requirements.txt	Optimizes	3 years ago
solve.py	Optimizes	3 years ago

kali虚拟机中直接git clone下载



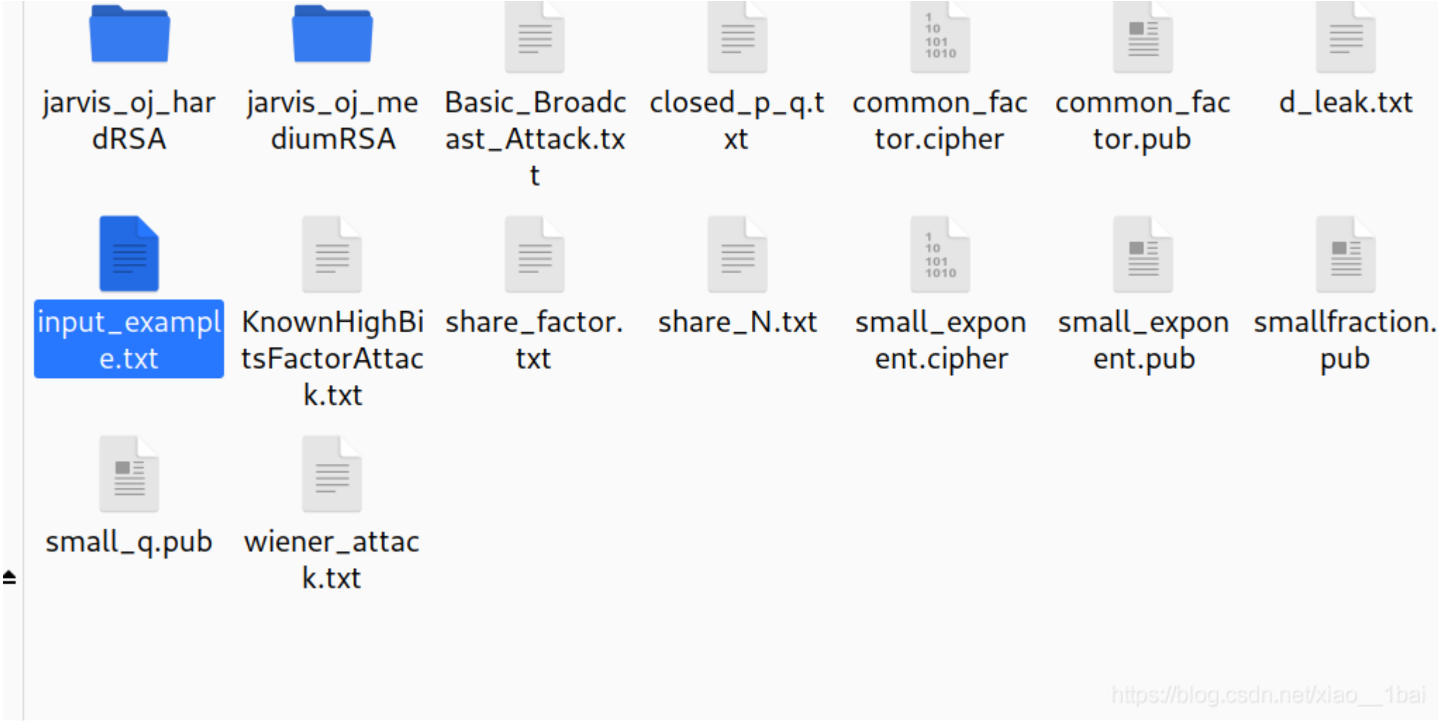
这是python2编写的，2018年的脚本，按照要求搞好配置后开始研究这脚本在RSA中的利用，希望能解答大多数RSA题目：(PS：这里不得不吐槽一下，这个脚本工具里的sagemath配置搞了我将近一整天，最后还是搞不出来，其实安转好了，就是那个from sage.all_cmdline import *老是报错说没有sage.all_cmdline这个模块，百度等浏览器上的资料又少，没办法，只能放弃了)

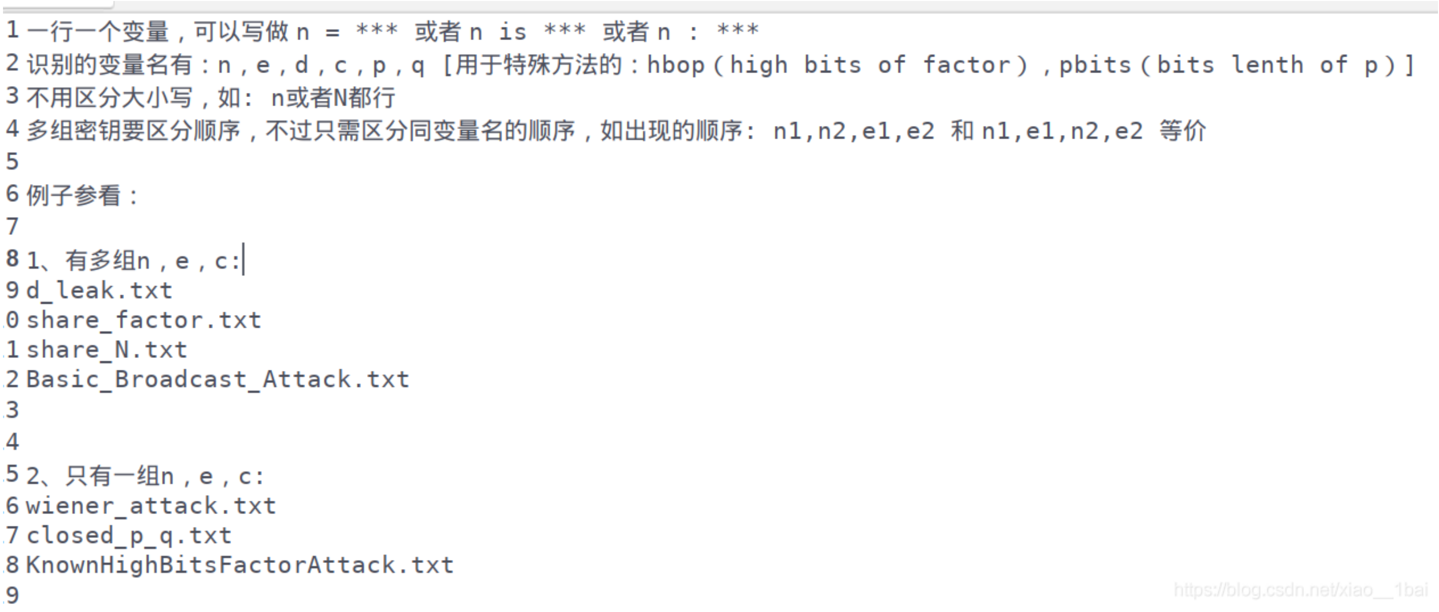
首先看该脚本工具README.md中的说明了解一些参数的用法和大概控制流程(中英注释都写在那里了)：

```
usage: solve.py [-h]
```

```
usage: solve.py [-h]
用法：solve.py[-h] (--decrypt DECRYPT | -c DECRYPT_INT | --private | -i INPUT | -g)
[--createpub] [-o OUTPUT] [--dumpkey] [--enc2dec ENC2DEC] [-k KEY] [-N N] [-e E] [-d D] [-p P] [-q Q] [--KHBFA K
HBFA][--pbits PBITS]
[-v]
It helps CTFer to get first blood of RSA-base CTF problems  它有助于CTFer获得RSA基础CTF问题的第一滴血
-v, --verbose         print details  详细的打印细节

optional arguments:可选参数(注意！！！这里之间只可选一个)：
-h, --help            show this help message and exit  --帮助显示此帮助消息并退出

--decrypt DECRYPT     decrypt a file, usually like "flag.enc" 解密文件，通常类似于“flag.enc”
(通常搭配k的.pem或.pub一起使用)

-c DECRYPT_INT,
--decrypt_int DECRYPT_INT  解密长整形数

--private             Print private key if recovered 打印私钥（如果已解密）

-i INPUT  input a file with all necessary parameters (see examples/input_example.txt)
输入包含所有必要参数的文件（请参见示例/输入（示例.txt）

-g, --gadget          Use some gadgets to pre-process your data first 使用一些小工具先预处理数据

some gadgets:一些小工具预处理数据：

--createpub           Take N and e and output to file specified by "-o" or just print it
获取N和e并输出到由“-o”指定的文件或者直接打印出来就行了

-o OUTPUT, --output OUTPUT  输出  Specify the output file path in --createpub mode.
在--createpub模式下指定输出文件路径。

--dumpkey             Just print the RSA variables from a key - n,e,d,p,q
只打印一个key-n、e、d、p、q中的RSA变量

--enc2dec ENC2DEC     get cipher (in decimalism) from a encrypted file
从加密文件中获取密码（十进制）

the RSA variables:RSA变量：Specify the variables whatever you got指定您得到的变量

-k KEY,   pem file, usually like ".pub" or ".pem", and it begins with "-----BEGIN"
pem文件，通常类似于“.pub”或“.pem”，并以“-----BEGIN”开始

-N N                  the modulus 模量

-e E                  the public exponent 公共指数

-d D                  the private exponent 私人指数

-p P                  one factor of modulus 模量的一个因子

-q Q                  one factor of modulus 模量的一个因子

extra variables:额外变量：Used in some special methods 在一些特殊的方法中使用：

--KHBFA KHBFA         use Known High Bits Factor Attack, this specify the High Bits of factor
使用已知的高位因子攻击，这指定因子高位

--pbits PBITS         customize the bits lenth of factor, default is half of n`s bits lenth
自定义因子的位长度，默认值为n's比特长度
```

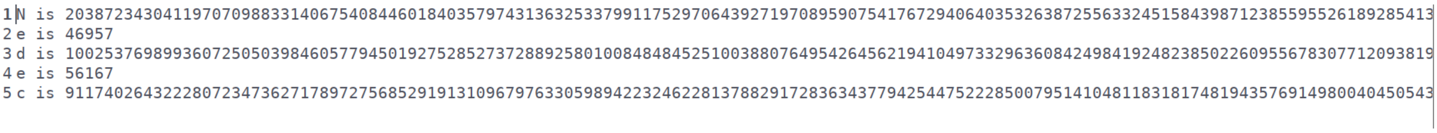
下面来看对应的解题例子，首先进入examples文件夹中打开input_example.txt



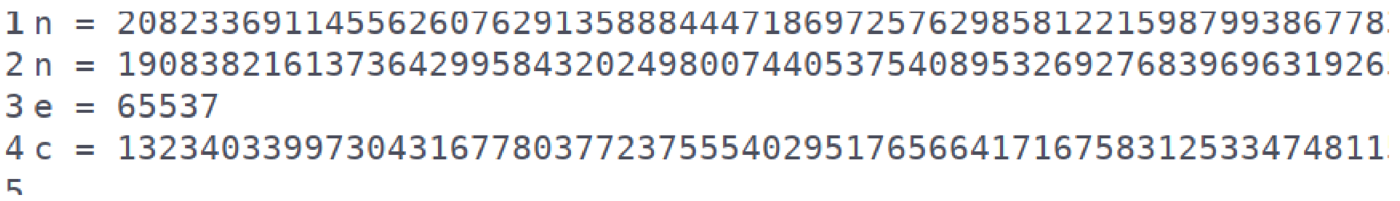


```
一行一个变量，可以写做 n = *** 或者 n is *** 或者 n : ***
识别的变量名有：n，e，d，c，p，q [用于特殊方法的：hbop（high bits of factor），pbits（bits lenth of p）]
不用区分大小写，如： n或者N都行
多组密钥要区分顺序，不过只需区分同变量名的顺序，如出现的顺序： n1,n2,e1,e2 和 n1,e1,n2,e2 等价

例子参看：

1、有多组n，e，c:
d_leak.txt
share_factor.txt
share_N.txt
Basic_Broadcast_Attack.txt


2、只有一组n，e，c:
wiener_attack.txt
closed_p_q.txt
KnownHighBitsFactorAttack.txt

```

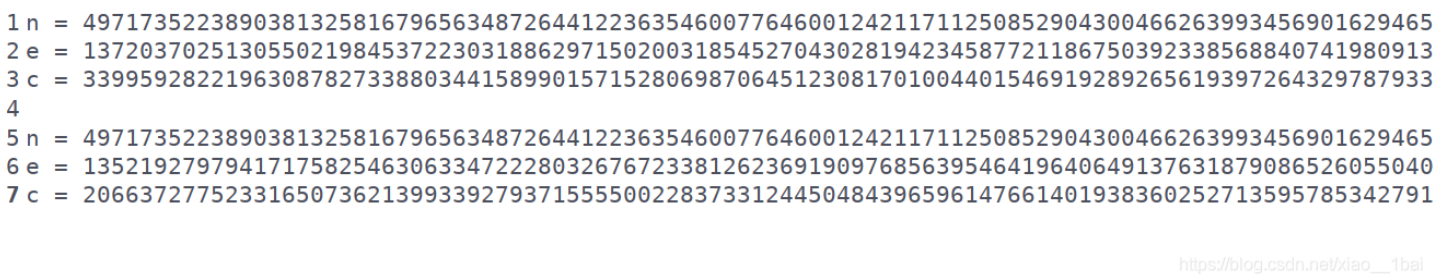
按照上面所说，多组n,e,c在解题时长这个样子：



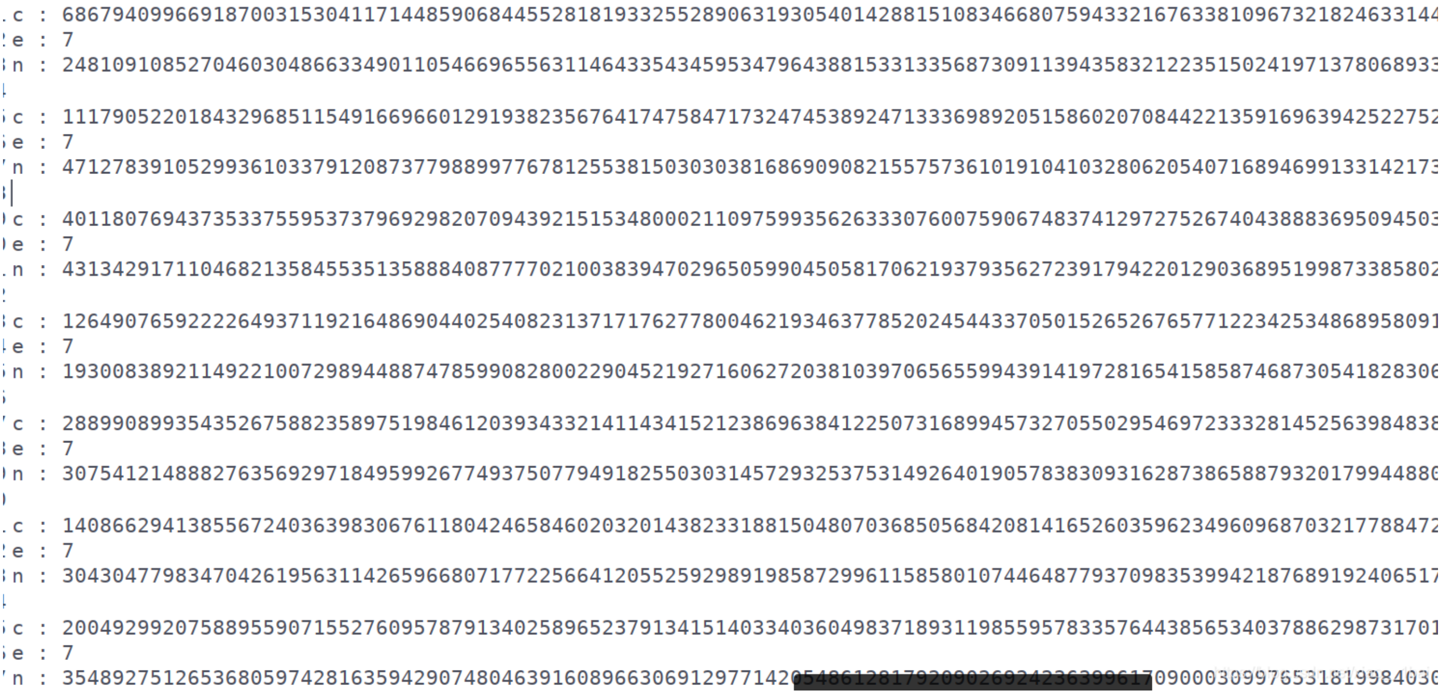
```
N is 20387234304119707098833140675408446018403579743136325337991175297064392719708959075417672940640353263872556332451584398712385595526189285413
e is 46957
d is 10025376989936072505039846057794501927528527372889258010084848452510038807649542645621941049733296360842498419248238502260955678307712093819
e is 56167
c is 91174026432228072347362717897275685291913109679763305989422324622813788291728363437794254475222850079514104811831817481943576914980040450543
```



```
n = 2082336911455626076291358884447186972576298581221598799386778
n = 1908382161373642995843202498007440537540895326927683969631926
e = 65537
c = 1323403399730431677803772375554029517656641716758312533474811

```



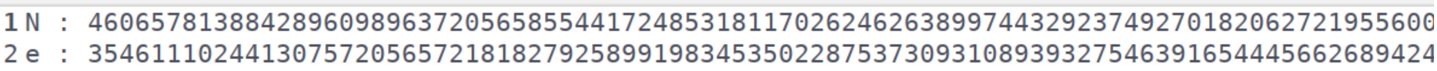
```
n = 49717352238903813258167965634872644122363546007764600124211711250852904300466263993456901629465
e = 13720370251305502198453722303188629715020031854527043028194234587721186750392338568840741980913
c = 33995928221963087827338803441589901571528069870645123081701004401546919289265619397264329787933

n = 49717352238903813258167965634872644122363546007764600124211711250852904300466263993456901629465
e = 13521927979417175825463063347222803267672338126236919097685639546419640649137631879086526055040
c = 20663727752331650736213993392793715555002283733124450484396596147661401938360252713595785342791
```



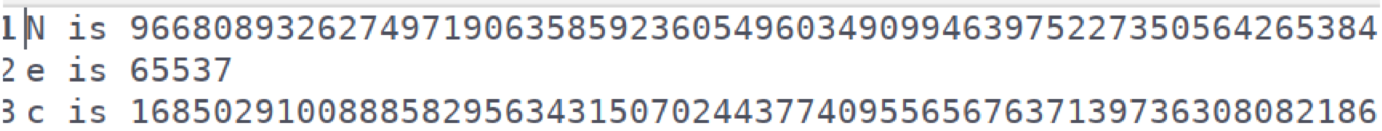
```
c : 68679409966918700315304117144859068445528181933255289063193054014288151083466807594332167633810967321824633144
e : 7
n : 24810910852704603048663349011054669655631146433543459534796438815331335687309113943583212235150241971378068933

c : 11179052201843296851154916696601291938235676417475847173247453892471333698920515860207084422135916963942522752
e : 7
n : 47127839105299361033791208737798899776781255381503030381686909082155757361019104103280620540716894699133142173

c : 40118076943735337559537379692982070943921515348000211097599356263330760075906748374129727526740438883695094503
e : 7
n : 43134291711046821358455351358884087777021003839470296505990450581706219379356272391794220129036895199873385802

c : 12649076592222649371192164869044025408231371717627780046219346377852024544337050152652676577122342534868958091
e : 7
n : 19300838921149221007298944887478599082800229045219271606272038103970656559943914197281654158587468730541828306

c : 28899089935435267588235897519846120393433214114341521238696384122507316899457327055029546972333281452563984838
e : 7
n : 30754121488827635692971849599267749375077949182550303145729325375314926401905783830931628738658879320179944880

c : 14086629413855672403639830676118042465846020320143823318815048070368505684208141652603596234960968703217788472
e : 7
n : 30430477983470426195631142659668071772256641205525929891985872996115858010744648779370983539942187689192406517

c : 20049299207588955907155276095787913402589652379134151403340360498371893119855957833576443856534037886298731701
e : 7
n : 35489275126536805974281635942907480463916089663069129771420[redacted]709000309976531819984030
```

可以看到特征真的就是多个n，e，c，甚至还有d，且不管这里的n,e,c是大还是小，长还是短，都列入多组n,e,c类型里。

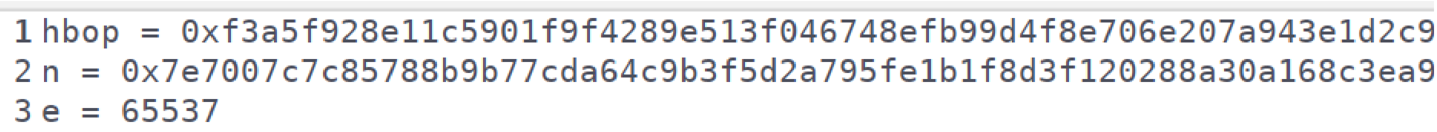
继续看只有一组n,e,c的题目样式：



```
N : 46065781388428960989637205658554417248531811702624626389974432923749270182062721955600
e : 35461110244130757205657218182792589919834535022875373093108939327546391654445662689424
```



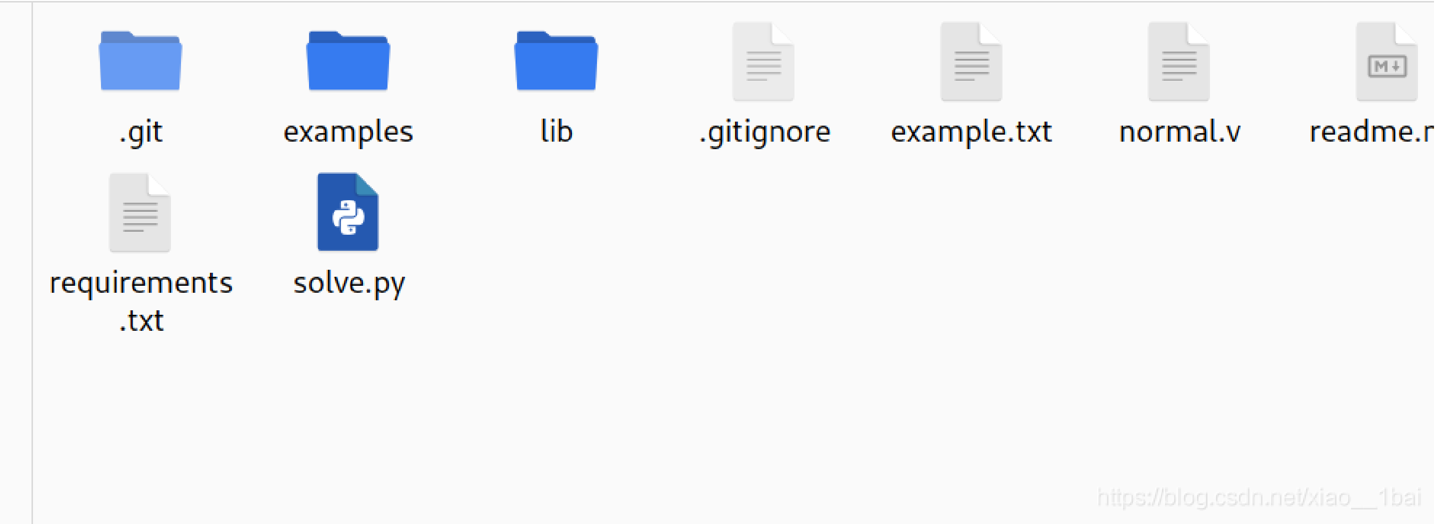
```
N is 966808932627497190635859236054960349099463975227350564265384
e is 65537
c is 168502910088858295634315070244377409556567637139736308082186
```



```
hbop = 0xf3a5f928e11c5901f9f4289e513f046748efb99d4f8e706e207a943e1d2c9
n = 0x7e7007c7c85788b9b77cda64c9b3f5d2a795fe1b1f8d3f120288a30a168c3ea9
e = 65537
```

从这里可以看到一组,n,e,c里面甚至可以没有c，这里的n,e,c也不管大小，长短，这里最后一个hbop的解题要用到前面说得sagemath，这里暂且不说。

然后我们继续观察该脚本工具文件夹根目录下的example.txt，继续分析利用方式：



下面是该文档的内容：

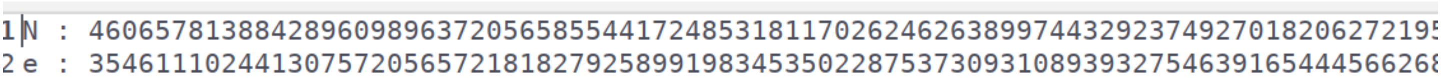
```
# 只需要一组密钥的
# wiener_attack
python2 solve.py --verbose --private -i examples/wiener_attack.txt
# 或者通过命令行，只要指定对应参数就行了
python2 solve.py  --verbose --private -N 460657813884289609896372056585544172485318117026246263899744329237492701820627219556007788200590119136173895989001382151536006853823326382892363143604314518686388786002989248800814861248595075326277099645338694977097459168530898776007293695728101976069423971696524237755227187061418202849911479124793990722597 -e 354611102441307572056572181827925899198345350228753730931089393275463916544456626894245415096107834465778409532373187125318554614722599301791528916212839368121066035541008808261534500586023652767712271625785204280964688004680328300124849680477105302519377370092578107827116821391826210972320377614967547827619

# factordb.com
python2 solve.py --verbose  -k examples/jarvis_oj_mediumRSA/pubkey.pem --decrypt examples/jarvis_oj_mediumRSA/flag.enc

# Boneh and Durfee attack
# TODO: get an example public key solvable by boneh_durfee but not wiener

# small q attack
python2 solve.py --verbose --private -k examples/small_q.pub

# 2017强网杯线上赛 RSA 费马分解（p&q相近时）
python2 solve.py --verbose --private -i examples/closed_p_q.txt

# Common factor between ciphertext and modulus attack
python2 solve.py --verbose -k examples/common_factor.pub --decrypt examples/common_factor.cipher

# small e
python2 solve.py --verbose -k examples/small_exponent.pub  --decrypt examples/small_exponent.cipher

# rabin method when e == 2
python solve.py --verbose -k examples/jarvis_oj_hardRSA/pubkey.pem --decrypt examples/jarvis_oj_hardRSA/flag.enc

# Small fractions method when p/q is close to a small fraction
python2 solve.py --verbose -k examples/smallfraction.pub  --private

# Known High Bits Factor Attack
python2 solve.py --verbose -i examples/KnownHighBitsFactorAttack.txt


# 需要多组密钥的
# 第三届上海市大学生网络安全大赛--rrrsa d泄漏攻击
python2 solve.py --verbose -i examples/d_leak.txt

# 模不互素
python2 solve.py --verbose -i examples/share_factor.txt

# 共模攻击
python2 solve.py --verbose -i examples/share_N.txt

# Basic Broadcast Attack
python2 solve.py --verbose -i examples/Basic_Broadcast_Attack.txt
```

按着顺序来：

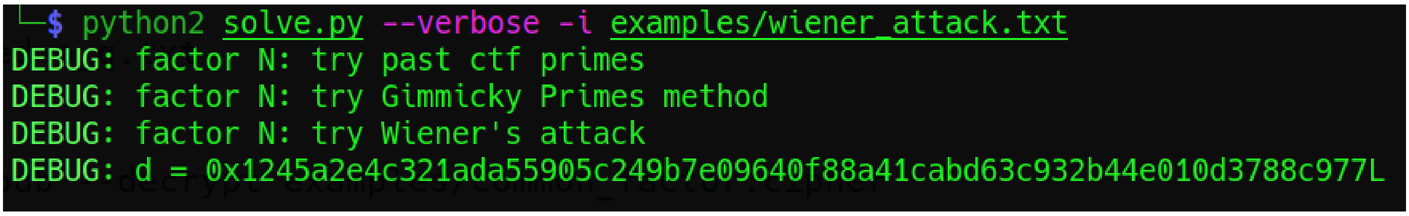
1： 只需要一组密钥的

wiener_attack

```
python2 solve.py --verbose -i examples/wiener_attack.txt
```



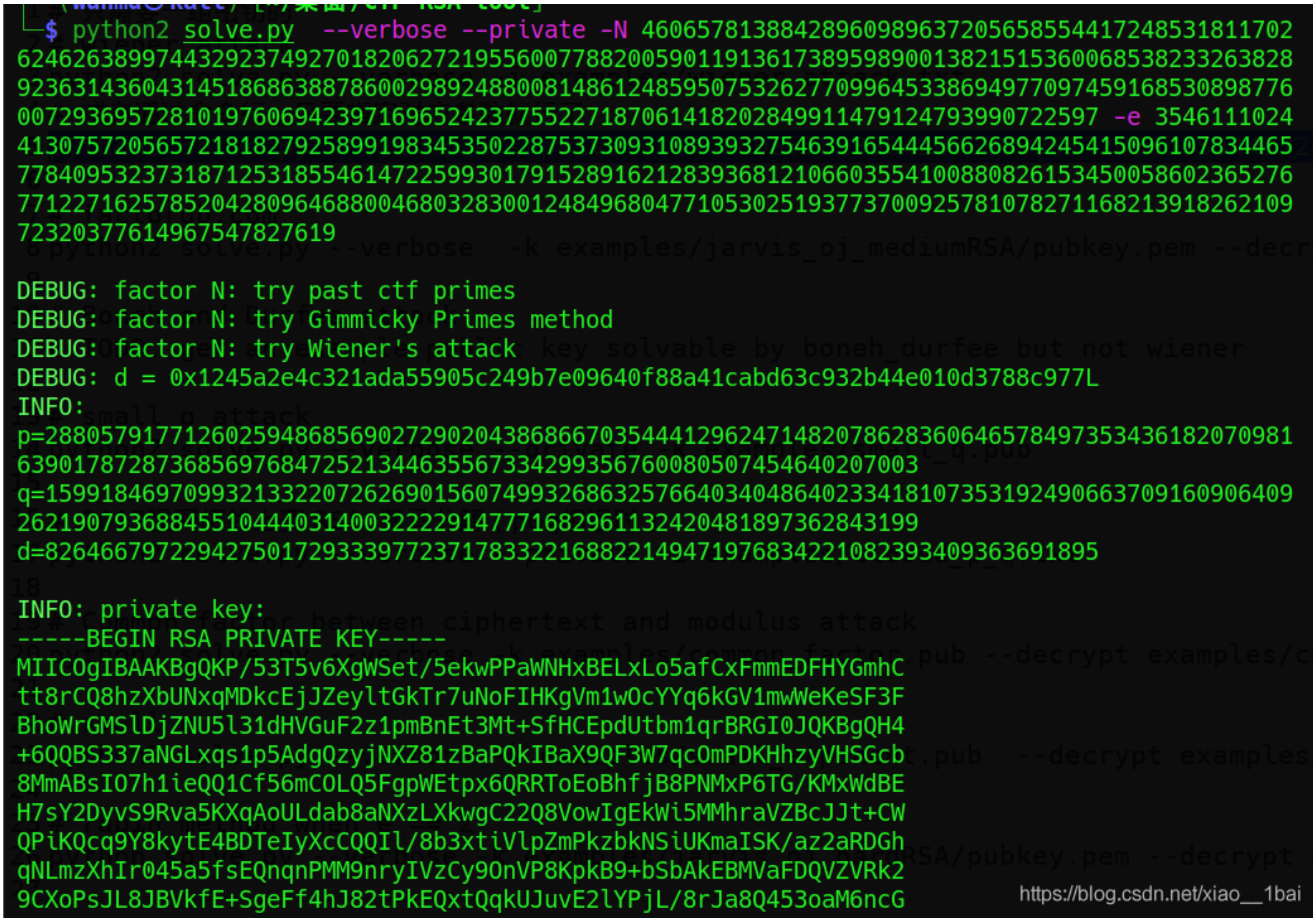
```
N : 4606578138842896098963720565855441724853181170262462638997443292374927018206272195
e : 354611102441307572056572181827925899198345350228753730931089393275463916544456626
```



```
└─$ python2 solve.py --verbose -i examples/wiener_attack.txt
DEBUG: factor N: try past ctf primes
DEBUG: factor N: try Gimmicky Primes method
DEBUG: factor N: try Wiener's attack
DEBUG: d = 0x1245a2e4c321ada55905c249b7e09640f88a41cabd63c932b44e010d3788c977L
```

2：或者通过命令行，只要指定对应参数就行了

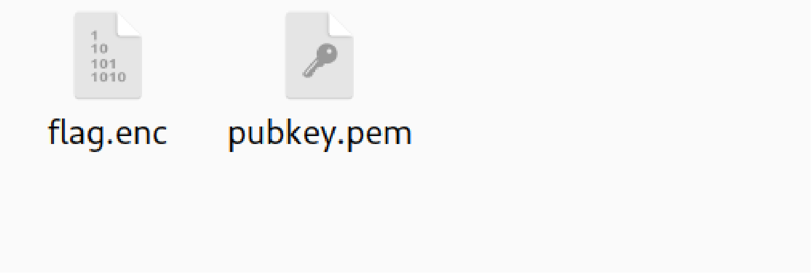
```
python2 solve.py  --verbose --private -N 460657813884289609896372056585544172485318117026246263899744329237492701820627219556007788200590119136173895989001382151536006853823326382892363143604314518686388786002989248800814861248595075326277099645338694977097459168530898776007293695728101976069423971696524237755227187061418202849911479124793990722597 -e 354611102441307572056572181827925899198345350228753730931089393275463916544456626894245415096107834465778409532373187125318554614722599301791528916212839368121066035541008808261534500586023652767712271625785204280964688004680328300124849680477105302519377370092578107827116821391826210972320377614967547827619
```

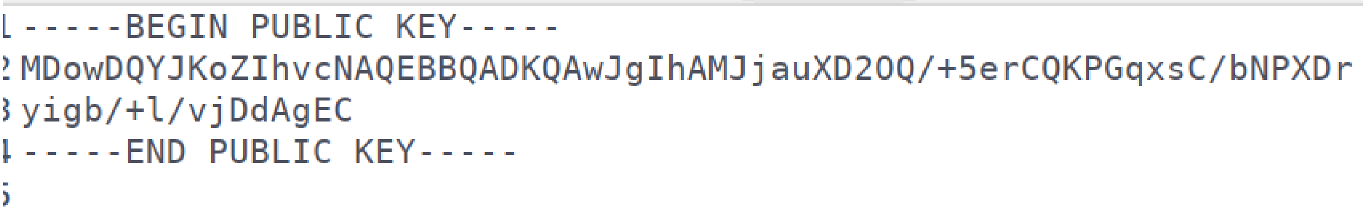


```
└─$ python2 solve.py  --verbose --private -N 46065781388428960989637205658554417248531811702
62462638997443292374927018206272195560077882005901191361738959890013821515360068538233263828
92363143604314518686388786002989248800814861248595075326277099645338694977097459168530898776
007293695728101976069423971696524237755227187061418202849911479124793990722597 -e 3546111024
41307572056572181827925899198345350228753730931089393275463916544456626894245415096107834465
77840953237318712531855461472259930179152891621283936812106603554100880826153450058602365276
77122716257852042809646880046803283001248496804771053025193773700925781078271168213918262109
72320377614967547827619

DEBUG: factor N: try past ctf primes
DEBUG: factor N: try Gimmicky Primes method
DEBUG: factor N: try Wiener's attack
DEBUG: d = 0x1245a2e4c321ada55905c249b7e09640f88a41cabd63c932b44e010d3788c977L
INFO:
p=288057917712602594868569027290204386866703544412962471482078628360646578497353436182070981
63901787287368569768472521344635567334299356760080507454640207003
q=159918469709932133220726269015607499326863257664034048640233418107353192490663709160906409
26219079368845510444031400322229147771682961132420481897362843199
d=8264667972294275017293339772371783322168822149471976834221082393409363691895

INFO: private key:
-----BEGIN RSA PRIVATE KEY-----
MIICOgIBAAKBgQKP/53T5v6XgWSet/5ekwPPaWNHxBELxLo5afCxFmmEDFHYGmhC
tt8rCQ8hzXbUNxqMDkcEjJZeyltGkTr7uNoFIHKgVm1wOcYYq6kGV1mwWeKeSF3F
BhoWrGMSlDjZNU5l31dHVGuF2z1pmBnEt3Mt+SfHCEpdUtbm1qrBRGI0JQKBgQH4
+6QQBS337aNGLxqs1p5AdgQzyjNXZ81zBaPQkIBaX9QF3W7qcOmPDKHhzyVHSGcb
8MmABsIO7h1ieQQ1Cf56mCOLQ5FgpWEtpx6QRRToEoBhfjB8PNMxP6TG/KMxWdBE
H7sY2DyvS9Rva5KXqAoULdab8aNXzLXkwgC22Q8VowIgEkWi5MMhraVZBcJJt+CW
QPiKQcq9Y8kytE4BDTeIyXcCQQIl/8b3xtiVlpZmPkzbkNSiUKmaISK/az2aRDGh
qNLmzXhIr045a5fsEQnqnPMM9nryIVzCy9OnVP8KpkB9+bSbAkEBMVaFDQVZVRk2
9CXoPsJL8JBVkfE+SgeFf4hJ82tPkEQxtQqkUJuvE2lYPjL/8rJa8Q453oaM6ncG
```

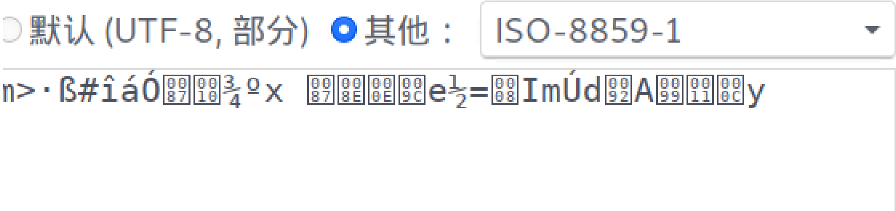
3：factordb.com

```
python2 solve.py --verbose  -k examples/jarvis_oj_mediumRSA/pubkey.pem --decrypt examples/jarvis_oj_mediumRSA/flag.enc
```



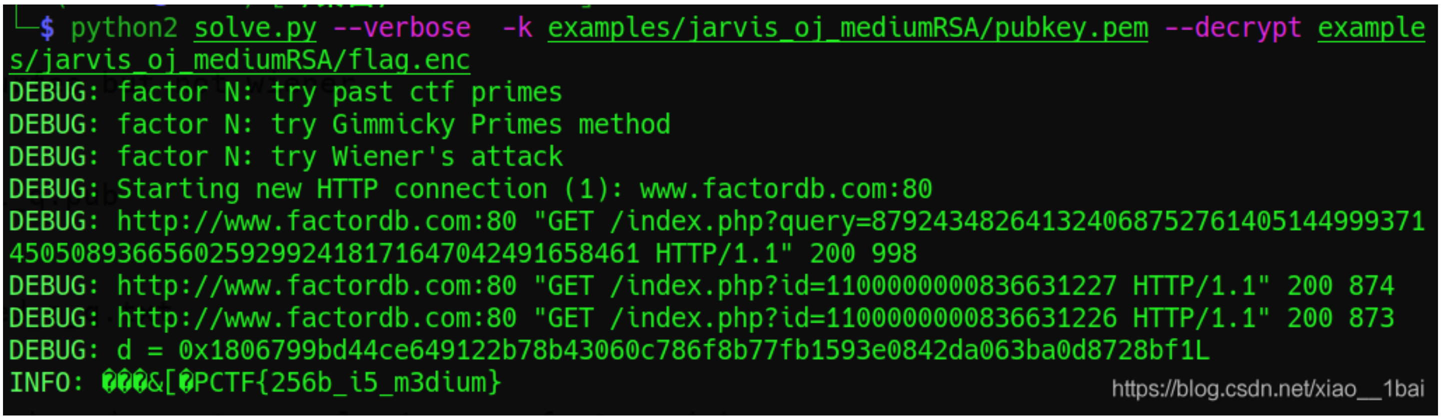


```
-----BEGIN PUBLIC KEY-----
MDowDQYJKoZIhvcNAQEBBQADKQAwJgIhAMJjauXD2OQ/+5erCQKPGqxsC/bNPXDr
yigb/+l/vjDdAgEC
-----END PUBLIC KEY-----
```



○默认 (UTF-8, 部分) ● 其他： ISO-8859-1

```
n>·ß#îáÓ\u0087\u0010¾ºx \u0087\u008e\u000e\u009ce½=\u0008ImÚd\u0092A\u0099\u0011\u000cy
```



```
└─$ python2 solve.py --verbose  -k examples/jarvis_oj_mediumRSA/pubkey.pem --decrypt example
s/jarvis_oj_mediumRSA/flag.enc
DEBUG: factor N: try past ctf primes
DEBUG: factor N: try Gimmicky Primes method
DEBUG: factor N: try Wiener's attack
DEBUG: Starting new HTTP connection (1): www.factordb.com:80
DEBUG: http://www.factordb.com:80 "GET /index.php?query=87924348264132406875276140514499937145050893665602592992418171647042491658461 HTTP/1.1" 200 998
DEBUG: http://www.factordb.com:80 "GET /index.php?id=1100000000836631227 HTTP/1.1" 200 874
DEBUG: http://www.factordb.com:80 "GET /index.php?id=1100000000836631226 HTTP/1.1" 200 873
DEBUG: d = 0x1806799bd44ce649122b78b43060c786f8b77fb1593e0842da063ba0d8728bf1L
INFO: ���&[�PCTF{256b_i5_m3dium}
```

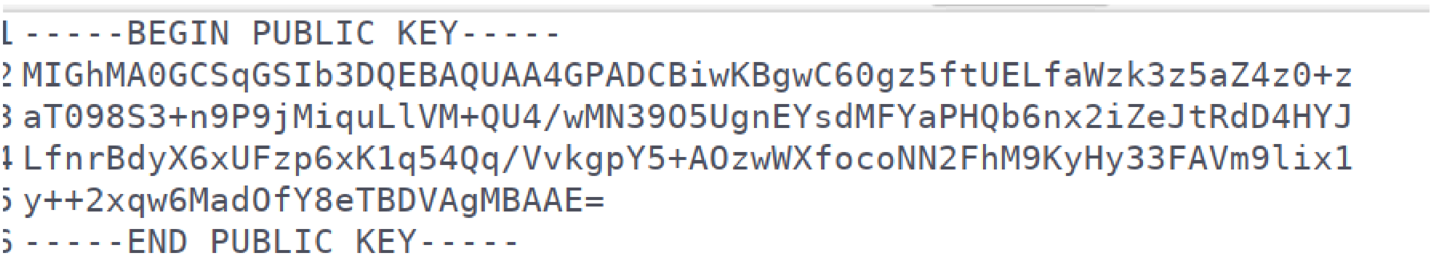
4：Boneh and Durfee attack

TODO: get an example public key solvable by boneh_durfee but not wiener

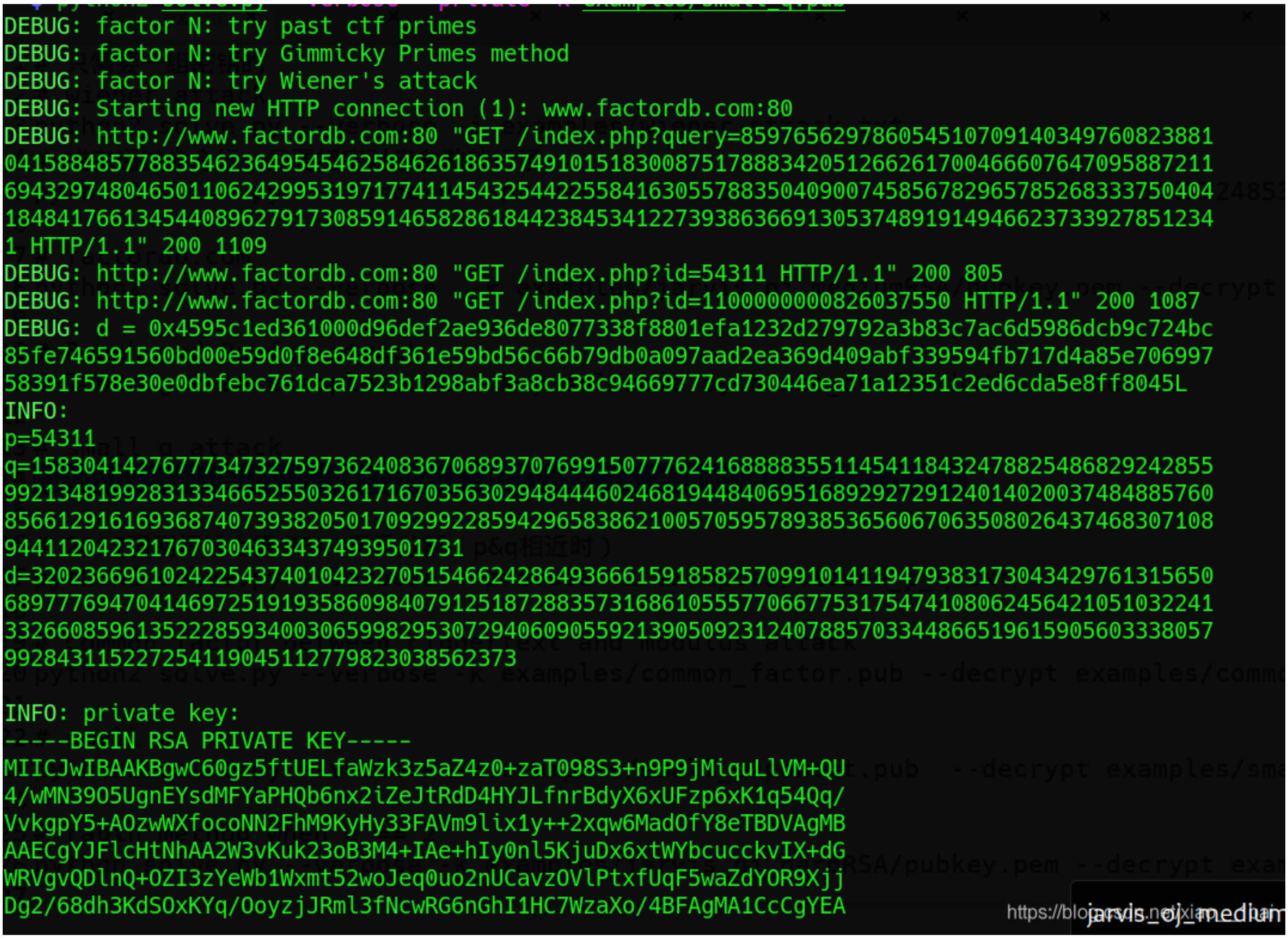
无解

5：small q attack

```
python2 solve.py --verbose --private -k examples/small_q.pub
```



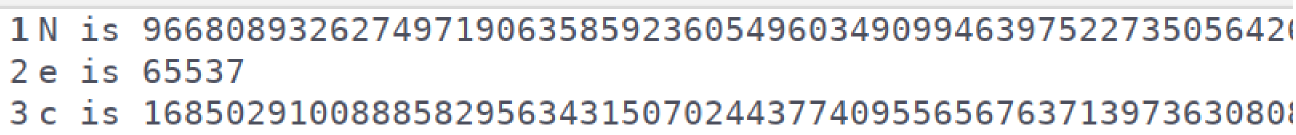
```
-----BEGIN PUBLIC KEY-----
MIGhMA0GCSqGSIb3DQEBAQUAA4GPADCBiwKBgwC60gz5ftUELfaWzk3z5aZ4z0+z
aT098S3+n9P9jMiquLlVM+QU4/wMN3905UgnEYsdMFYaPHQb6nx2iZeJtRdD4HYJ
LfnrBdyX6xUFzp6xK1q54Qq/VvkgpY5+AOzwWXfocoNN2FhM9KyHy33FAVm9lix1
y++2xqw6Mad0fY8eTBDVAgMBAAE=
-----END PUBLIC KEY-----
```



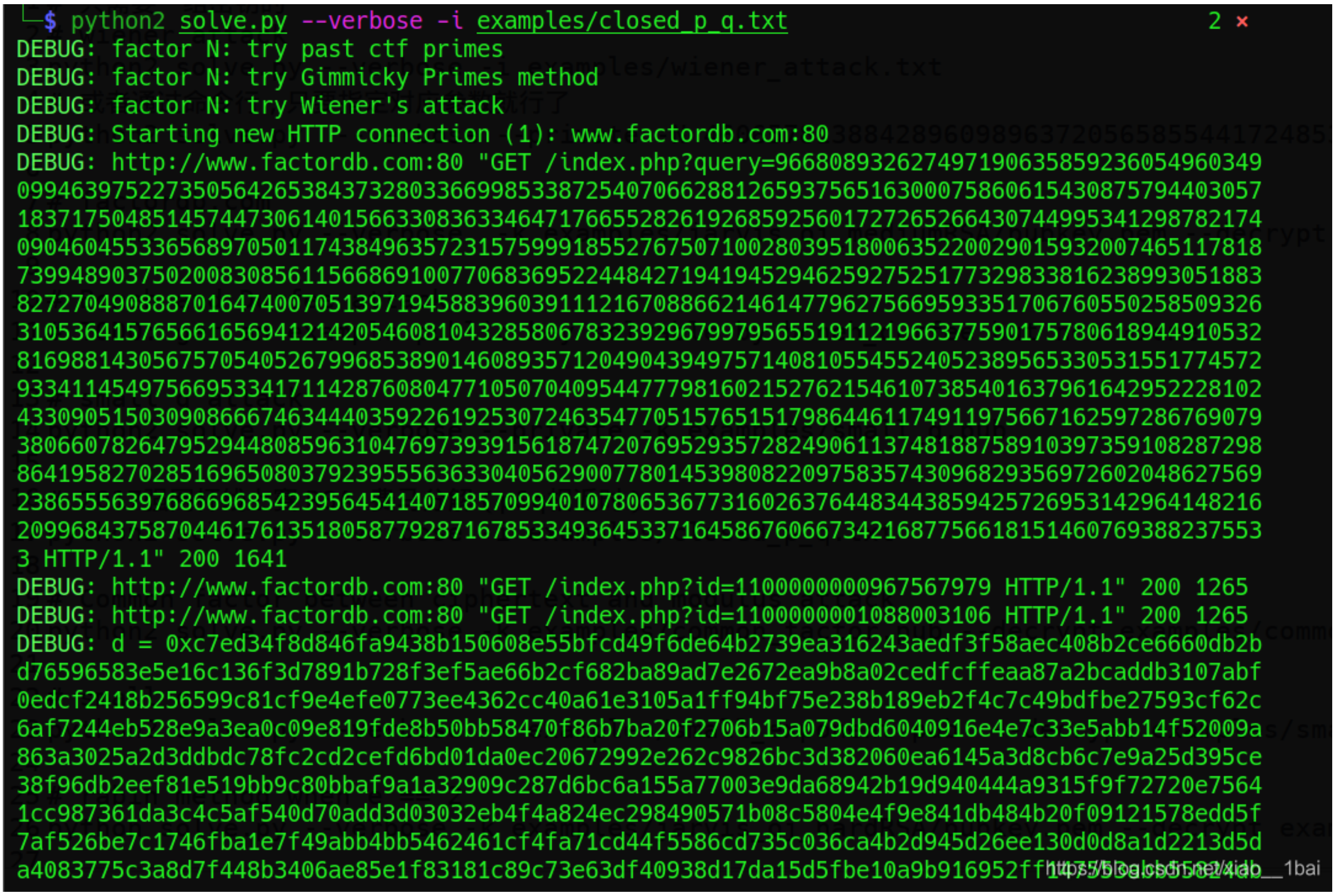
```
DEBUG: factor N: try past ctf primes
DEBUG: factor N: try Gimmicky Primes method
DEBUG: factor N: try Wiener's attack
DEBUG: Starting new HTTP connection (1): www.factordb.com:80
DEBUG: http://www.factordb.com:80 "GET /index.php?query=85976562978605451070914034976082388104158848577883546236495454625846261863574910151830087517888342051266261700466607647095887211694329748046501106242995319717741145432544225584163055788350409007458567829657852683337504041848417661345440896279173085914658286184423845341227393863669130537489191494662373392785123 41 HTTP/1.1" 200 1109
DEBUG: http://www.factordb.com:80 "GET /index.php?id=54311 HTTP/1.1" 200 805
DEBUG: http://www.factordb.com:80 "GET /index.php?id=1100000000826037550 HTTP/1.1" 200 1087
DEBUG: d = 0x4595c1ed361000d96def2ae936de8077338f8801efa1232d279792a3b83c7ac6d5986dcb9c724bc85fe746591560bd00e59d0f8e648df361e59bd56c66b79db0a097aad2ea369d409abf339594fb717d4a85e70699758391f578e30e0dbfebc761dca7523b1298abf3a8cb38c94669777cd730446ea71a12351c2ed6cda5e8ff8045L
INFO:
p=54311
q=158304142767773473275973624083670689370769915077762416888835511454118432478825486829242855992134819928313346652550326171670356302948444602468194484069516892927291240140200374848857608566129161693687407393820501709299228594296583862100570595789385365606706350802643746830710894411204232176703046334374939501731
d=320236696102422543740104232705154662428649366615918582570991014119479383173043429761315650689777694704146972519193586098407912518728835731686105557706677531754741080624564210510322413326608596135222859340030659982953072940609055921390509231240788570334486651961590560333805799284311522725411904511277982303856 2373

INFO: private key:
-----BEGIN RSA PRIVATE KEY-----
MIICJwIBAAKBgwC60gz5ftUELfaWzk3z5aZ4z0+zaT098S3+n9P9jMiquLlVM+QU
4/wMN3905UgnEYsdMFYaPHQb6nx2iZeJtRdD4HYJLfnrBdyX6xUFzp6xK1q54Qq/
VvkgpY5+AOzwWXfocoNN2FhM9KyHy33FAVm9lix1y++2xqw6Mad0fY8eTBDVAgMB
AAECgYJFlcHtNhAA2W3vKuk23oB3M4+IAe+hIy0nl5KjuDx6xtWYbcucckvIX+dG
WRVgvQDlnQ+OZI3zYeWb1Wxmt52woJeq0uo2nUCavzOVlPtxfUqF5waZdYOR9Xjj
Dg2/68dh3KdSOxKYq/OoyzjJRml3fNcwRG6nGhI1HC7WzaXo/4BFAgMA1CcCgYEA
```

6：2017强网杯线上赛 RSA 费马分解（p&q相近时）

```
python2 solve.py --verbose -i examples/closed_p_q.txt
```



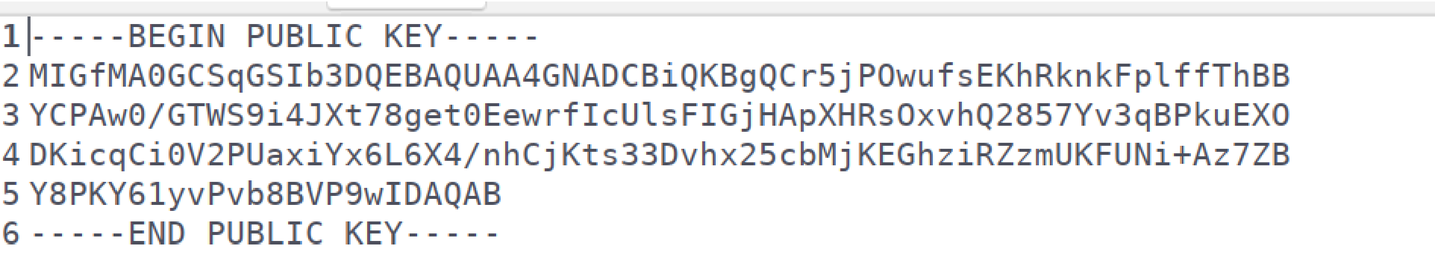
```
N is 96680893262749719063585923605496034909946397522735056426
e is 65537
c is 1685029100888582956343150702443774095565676371397363080
```



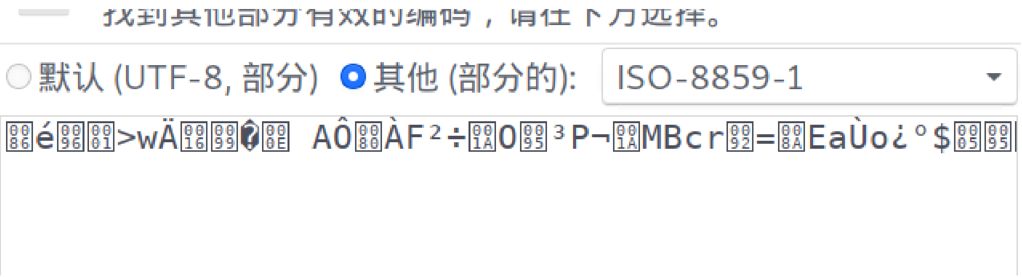
```
└─$ python2 solve.py --verbose -i examples/closed_p_q.txt                                  2 ×
DEBUG: factor N: try past ctf primes
DEBUG: factor N: try Gimmicky Primes method
DEBUG: factor N: try Wiener's attack
DEBUG: Starting new HTTP connection (1): www.factordb.com:80
DEBUG: http://www.factordb.com:80 "GET /index.php?query=966808932627497190635859236054960349
09946397522735056426538437328033669985338725407066288126593756516300075860615430875794403057
18371750485145744730614015663308363346471766552826192685925601727265266430744995341298782174
09046045533656897050117438496357231575999185527675071002803951800635220029015932007465117818
73994890375020083085611566869100770683695224484271941945294625927525177329833816238993051883
82727049088870164740070513971945883960391112167088662146147796275669593351706760550258509326
31053641576566165694121420546081043285806783239296799795655191121966377590175780618944910532
81698814305675705405267996853890146089357120490439497571408105545524052389565330531551774572
93341145497566953341711428760804771050704095447779816021527621546107385401637961642952228102
43309051503090866674634440359226192530724635477051576515179864461174911975667162597286769079
38066078264795294480859631047697393915618747207695293572824906113748188758910397359108287298
86419582702851696508037923955563633040562900778014539808220975835743096829356972602048627569
23865556397686696854239564541407185709940107806536773160263764483443859425726953142964148216
20996843758704461761351805877928716785334936453371645867606673421687756618151460769388237553
3 HTTP/1.1" 200 1641
DEBUG: http://www.factordb.com:80 "GET /index.php?id=1100000000967567979 HTTP/1.1" 200 1265
DEBUG: http://www.factordb.com:80 "GET /index.php?id=1100000001088003106 HTTP/1.1" 200 1265
DEBUG: d = 0xc7ed34f8d846fa9438b150608e55bfcd49f6de64b2739ea316243aedf3f58aec408b2ce6660db2b
d76596583e5e16c136f3d7891b728f3ef5ae66b2cf682ba89ad7e2672ea9b8a02cedfcffeaa87a2bcaddb3107abf
0edcf2418b256599c81cf9e4efe0773ee4362cc40a61e3105a1ff94bf75e238b189eb2f4c7c49bdfbe27593cf62c
6af7244eb528e9a3ea0c09e819fde8b50bb58470f86b7ba20f2706b15a079dbd6040916e4e7c33e5abb14f52009a
863a3025a2d3ddbdc78fc2cd2cefd6bd01da0ec20672992e262c9826bc3d382060ea6145a3d8cb6c7e9a25d395ce
38f96db2eef81e519bb9c80bbaf9a1a32909c287d6bc6a155a77003e9da68942b19d940444a9315f9f72720e7564
1cc987361da3c4c5af540d70add3d03032eb4f4a824ec298490571b08c5804e4f9e841db484b20f09121578edd5f
7af526be7c1746fba1e7f49abb4bb5462461cf4fa71cd44f5586cd735c036ca4b2d945d26ee130d0d8a1d2213d5d
a4083775c3a8d7f448b3406ae85e1f83181c89c73e63df40938d17da15d5fbe10a9b916952ff
```

7：Common factor between ciphertext and modulus attack

```
python2 solve.py --verbose -k examples/common_factor.pub --decrypt examples/common_factor.cipher
```



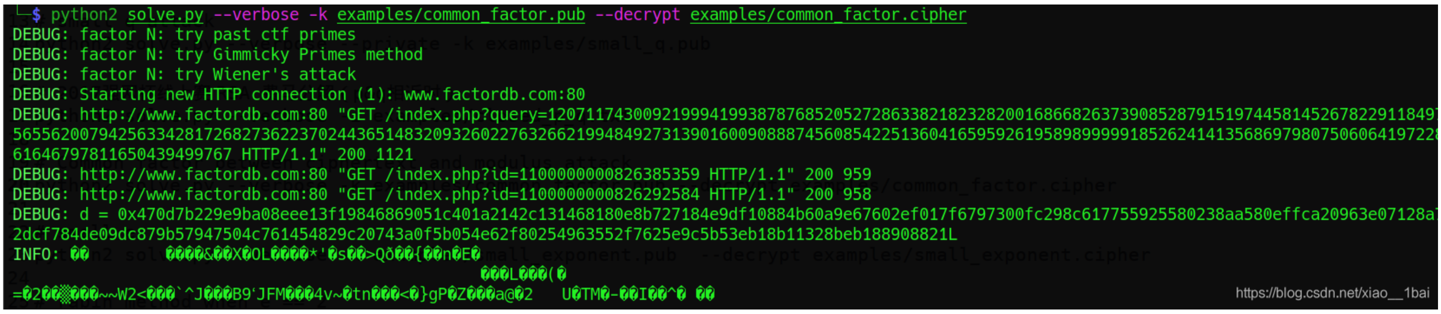
```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCr5jPOwufsEKhRknkFplffThBB
YCPAw0/GTWS9i4JXt78get0EewrfIcUlsFIGjHApXHRsOxvhQ2857Yv3qBPkuEXO
DKicqCi0V2PUaxiYx6L6X4/nhCjKts33Dvhx25cbMjKEGhziRZzmUKFUNi+Az7ZB
Y8PKY61yvPvb8BVP9wIDAQAB
-----END PUBLIC KEY-----
```



找到其他部分有效的编码，请在下方选择。

○ 默认 (UTF-8, 部分) ● 其他 (部分的): ISO-8859-1

□é□□>wÄ□□�□ AÔ□ÀF²÷□O□³P¬□MBcr□=□EaÙo¿°$□□

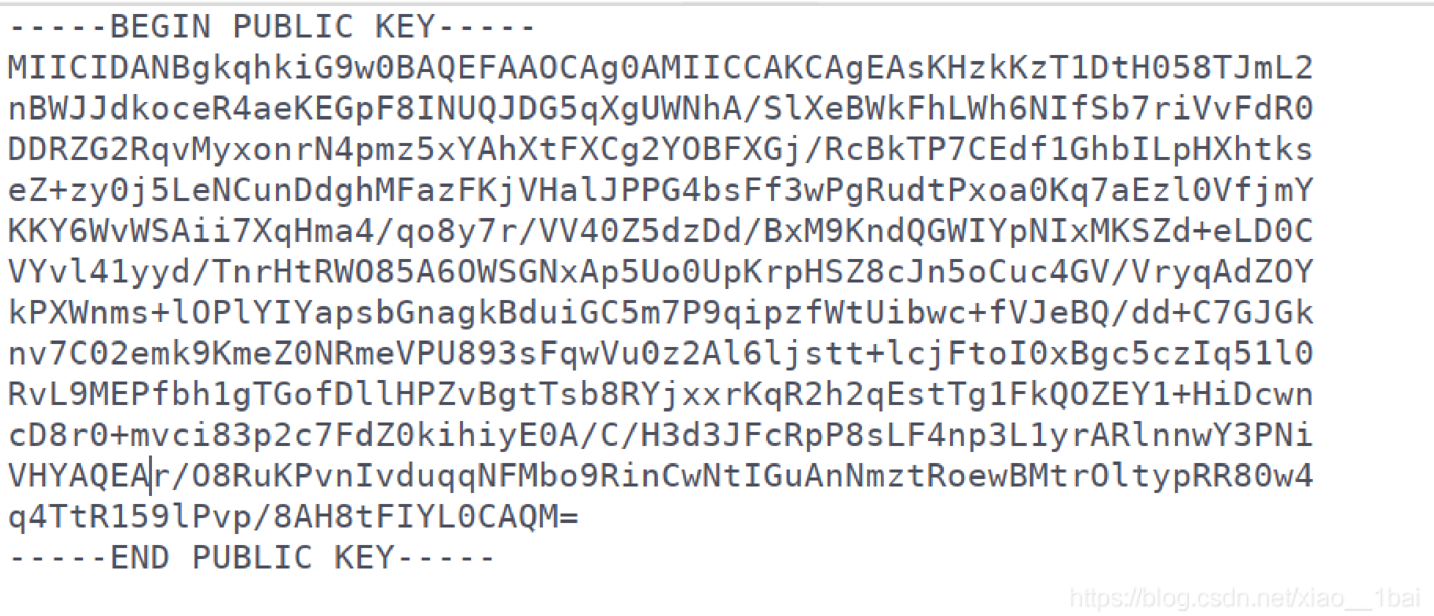


```
└─$ python2 solve.py --verbose -k examples/common_factor.pub --decrypt examples/common_factor.cipher
DEBUG: factor N: try past ctf primes
DEBUG: factor N: try Gimmicky Primes method
DEBUG: factor N: try Wiener's attack
DEBUG: Starting new HTTP connection (1): www.factordb.com:80
DEBUG: http://www.factordb.com:80 "GET /index.php?query=1207117430092199941993878768520527286338218232820016866826373908528791519744581452678229118497
56556200794256334281726827362237024436514832093260227632662199484927313901600908887456085422513604165959261958989999918526241413568697980750606419722
616467978116504394 99767 HTTP/1.1" 200 1121
DEBUG: http://www.factordb.com:80 "GET /index.php?id=1100000000826385359 HTTP/1.1" 200 959
DEBUG: http://www.factordb.com:80 "GET /index.php?id=1100000000826292584 HTTP/1.1" 200 958
DEBUG: d = 0x470d7b229e9ba08eee13f19846869051c401a2142c131468180e8b727184e9df10884b60a9e67602ef017f6797300fc298c617755925580238aa580effca20963e07128a
2dcf784de09dc879b57947504c761454829c20743a0f5b054e62f80254963552f7625e9c5b53eb18b11328beb188908821L
INFO: ��       ����&��X�OL����*'�s��>Qð��{��n�E�
                                                  ���L���(�
=�2��▒���~~W2<���`^J���B9'JFM���4v~�tn���<�}gP�Z���a@�2   U�TM�-��I��^� ��
```

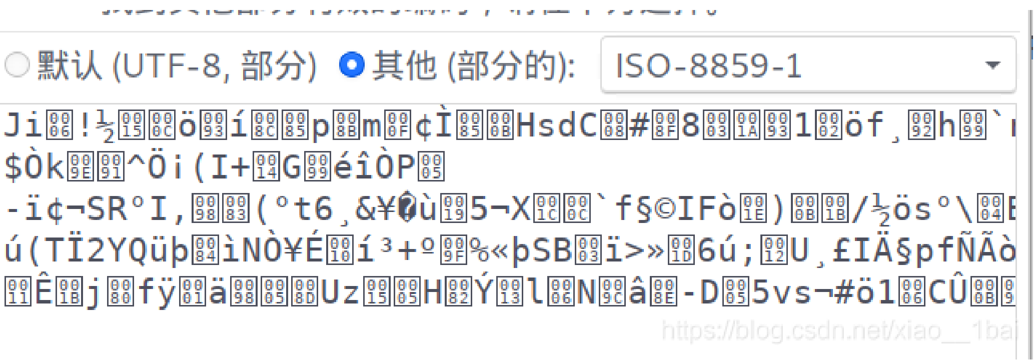
这里乱码了，后期还要再改源代码，思路是先十六进制输出来。

8：small e

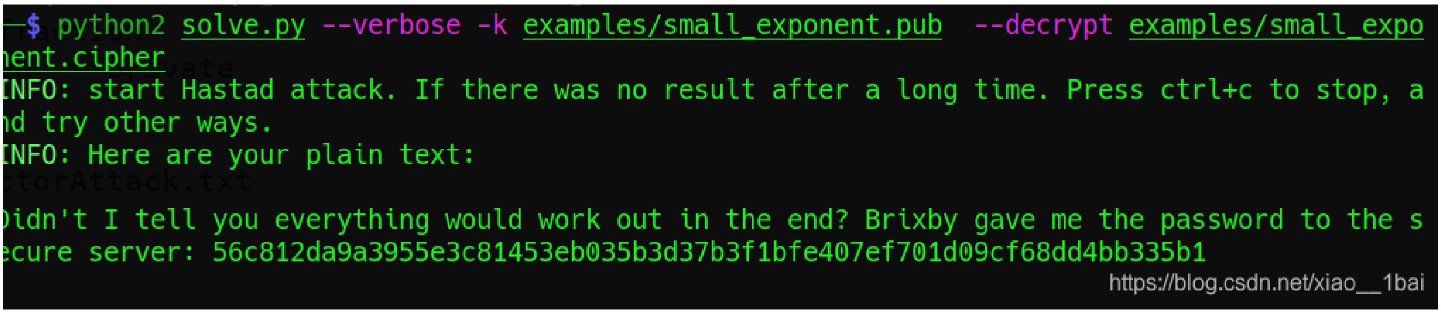
```
python2 solve.py --verbose -k examples/small_exponent.pub  --decrypt examples/small_exponent.cipher
```



```
-----BEGIN PUBLIC KEY-----
MIICIDANBgkqhkiG9w0BAQEFAAOCAg0AMIICCAKCAgEAsKHzkKzT1DtH058TJmL2
nBWJJdkoceR4aeKEGpF8INUQJDG5qXgUWNhA/SlXeBWkFhLWh6NIfSb7riVvFdR0
DDRZG2RqvMyxonrN4pmz5xYAhXtFXCg2YOBFXGj/RcBkTP7CEdf1GhbILpHXhtks
eZ+zy0j5LeNCunDdghMFazFKjVHalJPPG4bsFf3wPgRudtPxoa0Kq7aEzl0VfjmY
KKY6WvWSAii7XqHma4/qo8y7r/VV40Z5dzDd/BxM9KndQGWIYpNIxMKSZd+eLD0C
VYvl41yyd/TnrHtRWO85A6OWSGNxAp5Uo0UpKrpHSZ8cJn5oCuc4GV/VryqAdZOY
kPXWnms+lOPlYIYapsbGnagkBduiGC5m7P9qipzfWtUibwc+fVJeBQ/dd+C7GJGk
nv7C02emk9KmeZ0NRmeVPU893sFqwVu0z2Al6ljstt+lcjFtoI0xBgc5czIq51l0
RvL9MEPfbh1gTGofDllHPZvBgtTsb8RYjxxrKqR2h2qEstTg1FkQOZEY1+HiDcwn
cD8r0+mvci83p2c7FdZ0kihiyE0A/C/H3d3JFcRpP8sLF4np3L1yrARlnnwY3PNi
VHYAQEAr/O8RuKPvnIvduqqNFMbo9RinCwNtIGuAnNmztRoewBMtrOltypRR80w4
q4TtR159lPvp/8AH8tFIYL0CAQM=
-----END PUBLIC KEY-----
```



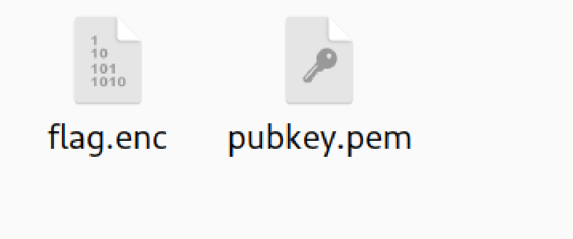
```
Ji□!½□□ö□í□□p□m□¢Ì□□HsdC□#□8□□□1□öf¸□h□`ı
$Òk□□^Ö¡(I+□G□éîÒP□
-ï¢¬SR°I,□□(°t6¸&¥�ù□5¬X□□`f§©IFò□)□□/½ös°\□E
ú(TÏ2YQüþ□ìNÒ¥É□í³+º□%«þSB□ï>»□6ú;□U¸£IÄ§pfÑÃò
□Ê□j□fÿ□ä□□□Uz□□H□Ý□l□N□â□-D□5vs¬#ö1□CÛ□□
```

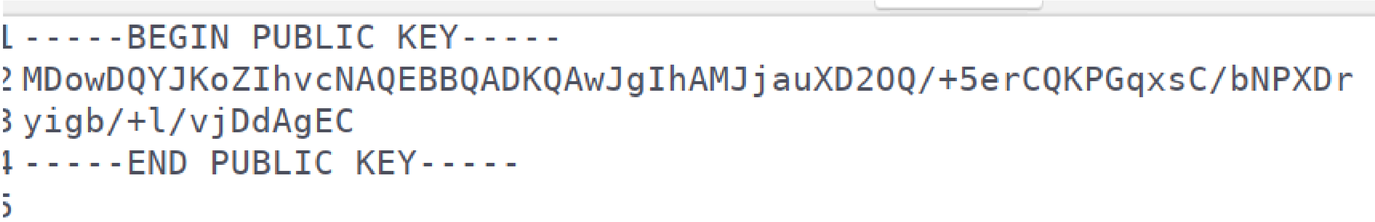


```
—$ python2 solve.py --verbose -k examples/small_exponent.pub  --decrypt examples/small_expo
nent.cipher
INFO: start Hastad attack. If there was no result after a long time. Press ctrl+c to stop, a
nd try other ways.
INFO: Here are your plain text:

Didn't I tell you everything would work out in the end? Brixby gave me the password to the s
ecure server: 56c812da9a3955e3c81453eb035b3d37b3f1bfe407ef701d09cf68dd4bb335b1
```

8：rabin method when e == 2

```
python2 solve.py --verbose -k examples/jarvis_oj_hardRSA/pubkey.pem --decrypt examples/jarvis_oj_hardRSA/flag.enc
```



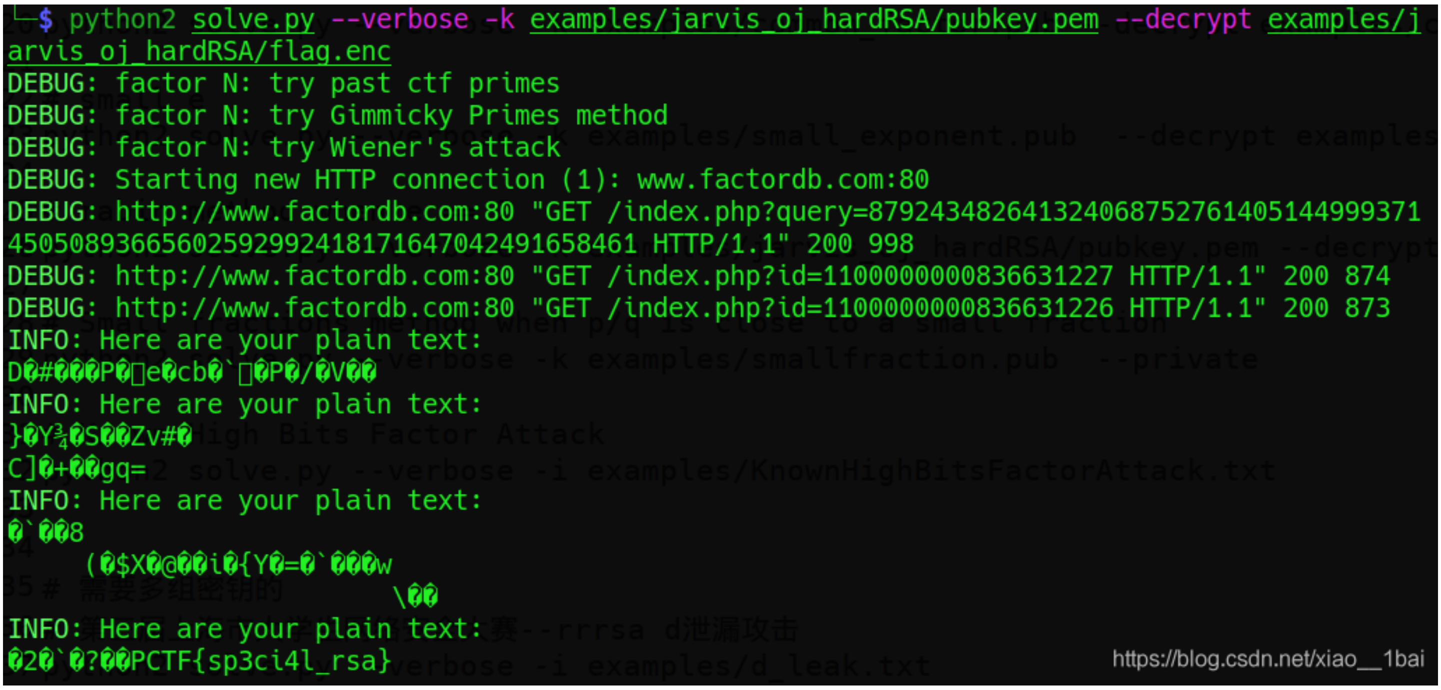


```
-----BEGIN PUBLIC KEY-----
MDowDQYJKoZIhvcNAQEBBQADKQAwJgIhAMJjauXD2OQ/+5erCQKPGqxsC/bNPXDr
yigb/+l/vjDdAgEC
-----END PUBLIC KEY-----
```

已找到其他有效编码，请在下面选择。

○默认 (UTF-8, 部分) ● 其他： ISO-8859-1

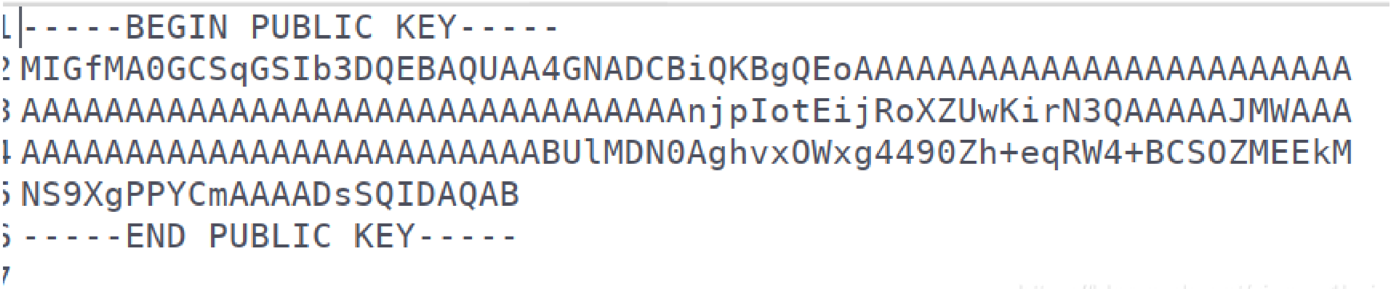
```
9Þ[03]mã[13]'Wè[19]÷iêÖK´[87]î?GæxC¯·7Hý[9E][97][9B]à
```



```
└─$ python2 solve.py --verbose -k examples/jarvis_oj_hardRSA/pubkey.pem --decrypt examples/j
arvis_oj_hardRSA/flag.enc
DEBUG: factor N: try past ctf primes
DEBUG: factor N: try Gimmicky Primes method
DEBUG: factor N: try Wiener's attack
DEBUG: Starting new HTTP connection (1): www.factordb.com:80
DEBUG: http://www.factordb.com:80 "GET /index.php?query=879243482641324068752761405144999371
45050893665602592992418171647042491658461 HTTP/1.1" 200 998
DEBUG: http://www.factordb.com:80 "GET /index.php?id=1100000000836631227 HTTP/1.1" 200 874
DEBUG: http://www.factordb.com:80 "GET /index.php?id=1100000000836631226 HTTP/1.1" 200 873
INFO: Here are your plain text:
D�#���P�e�cb�`�P�/�V��
INFO: Here are your plain text:
}�Y¾�S��Zv#�
C]�+��gq=
INFO: Here are your plain text:
�`��8
    (�$X�@��i�{Y�=�`���w
                  \��
INFO: Here are your plain text:
�2�`�?��PCTF{sp3ci4l_rsa}
```

9：Small fractions method when p/q is close to a small fraction

```
python2 solve.py --verbose -k examples/smallfraction.pub  --private
```

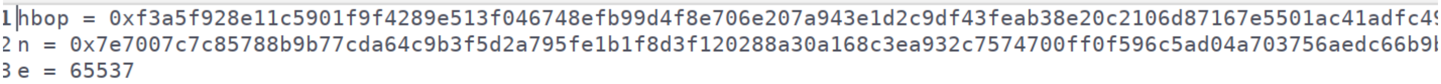


```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQEoAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAnjpIotEijRoXZUwKirN3QAAAAAJMWAAA
AAAAAAAAAAAAAAAAAAAAAAAAAABUlMDN0AghvxOWxg4490Zh+eqRW4+BCSOZMEEkM
NS9XgPPYCmAAAADsSQIDAQAB
-----END PUBLIC KEY-----
```

呃，我这里报错了，这需要sage，我说了我搞不定。

10：Known High Bits Factor Attack

```
python2 solve.py --verbose -i examples/KnownHighBitsFactorAttack.txt
```

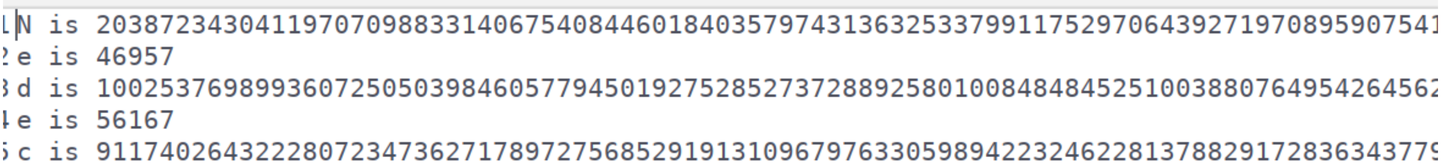


```
hbop = 0xf3a5f928e11c5901f9f4289e513f046748efb99d4f8e706e207a943e1d2c9df43feab38e20c2106d87167e5501ac41adfc4
n = 0x7e7007c7c85788b9b77cda64c9b3f5d2a795fe1b1f8d3f120288a30a168c3ea932c7574700ff0f596c5ad04a703756aedc66b9
e = 65537
```

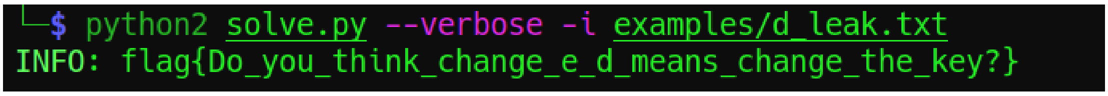
这个带0x的高指数更是需要sage，我也搞不定，复现不了。

11：需要多组密钥的
第三届上海市大学生网络安全大赛–rrrsa d泄漏攻击

```
python2 solve.py --verbose -i examples/d_leak.txt
```



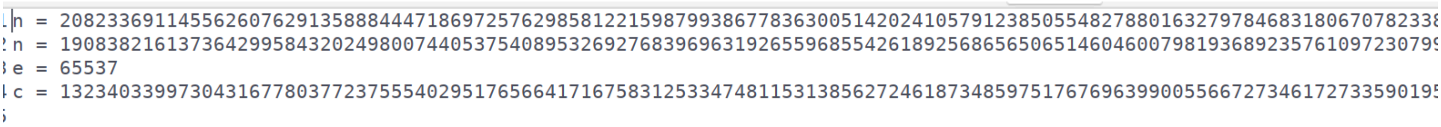
```
N is 2038723430411970709883314067540844601840357974313632533799117529706439271970895907541
e is 46957
d is 1002537698993607250503984605779450192752852737288925801008484845251003880764954264562
e is 56167
c is 911740264322280723473627178972756852919131096797633059894223246228137882917283634377
```



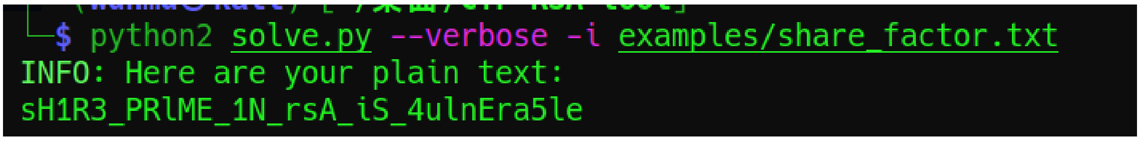
```
└─$ python2 solve.py --verbose -i examples/d_leak.txt
INFO: flag{Do_you_think_change_e_d_means_change_the_key?}
```

12：模不互素

```
python2 solve.py --verbose -i examples/share_factor.txt
```



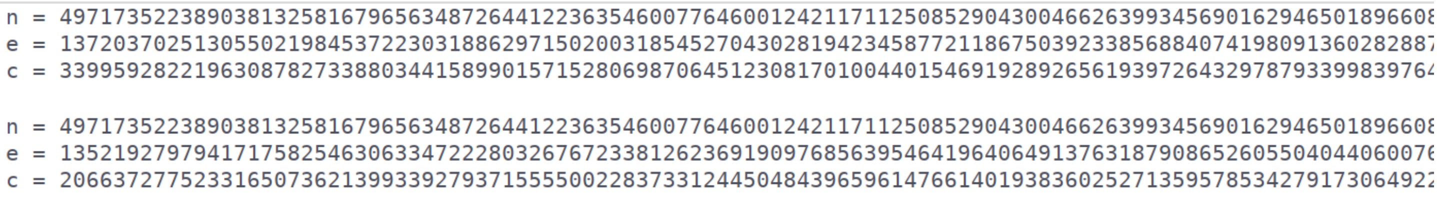
```
n = 2082336911455626076291358884447186972576298581221598799386778363005142024105791238505548278801632797846831806707823338
n = 1908382161373642995843202498007440537540895326927683969631926559685542618925686565065146046007981936892357610972307999
e = 65537
c = 1323403399730431677803772375554029517656641716758312533474811531385627246187348597517676963990055667273461727335901955
```



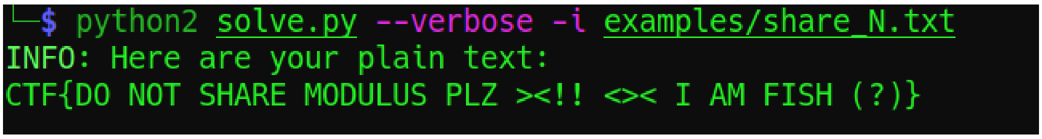
```
└─$ python2 solve.py --verbose -i examples/share_factor.txt
INFO: Here are your plain text:
sH1R3_PRlME_1N_rsA_iS_4ulnEra5le
```

13：共模攻击

```
python2 solve.py --verbose -i examples/share_N.txt
```



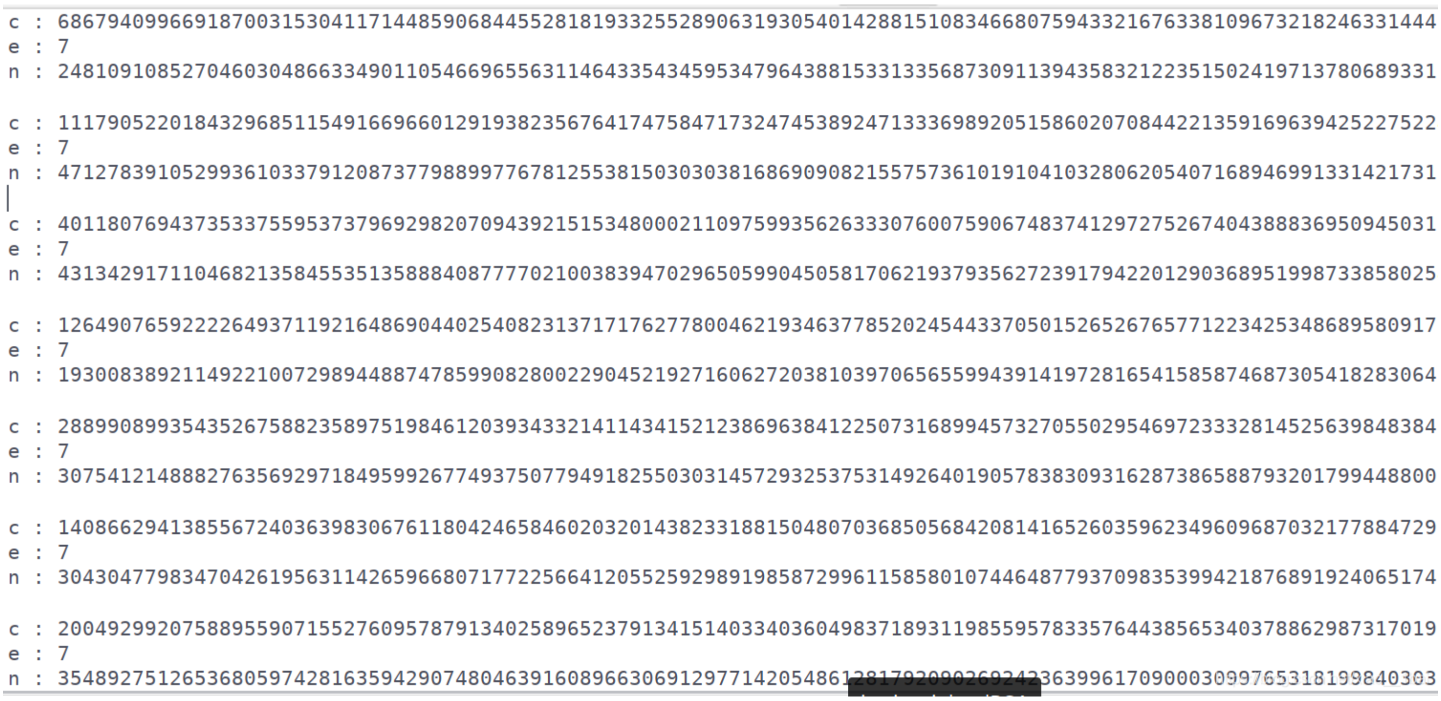
```
n = 4971735223890381325816796563487264412236354600776460012421171125085290430046626399345690162946501896608
e = 1372037025130550219845372230318862971502003185452704302819423458772118675039233856884074198091360282887
c = 3399592822196308782733880344158990157152806987064512308170100440154691928926561939726432978793399839764

n = 4971735223890381325816796563487264412236354600776460012421171125085290430046626399345690162946501896608
e = 1352192797941717582546306334722280326767233812623691909768563954641964064913763187908652605504044060076
c = 2066372775233165073621399339279371555500228373312445048439659614766140193836025271359578534279173064922
```



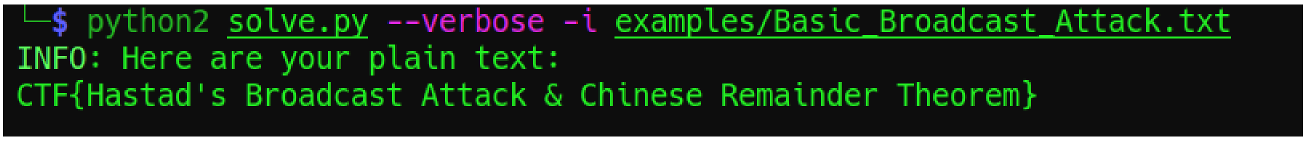
```
└─$ python2 solve.py --verbose -i examples/share_N.txt
INFO: Here are your plain text:
CTF{DO NOT SHARE MODULUS PLZ ><!! <>< I AM FISH (?)}
```

14：Basic Broadcast Attack

```
python2 solve.py --verbose -i examples/Basic_Broadcast_Attack.txt
```



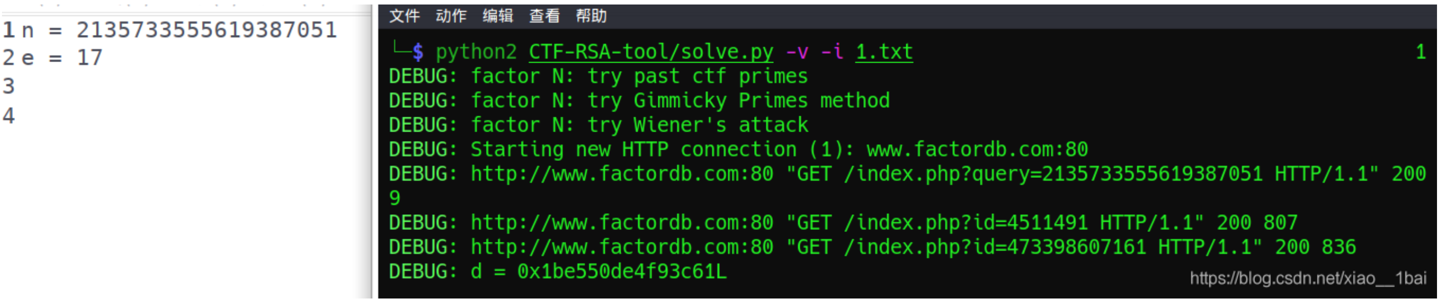
```
c : 6867940996691870031530411714485906844552818193325528906319305401428815108346680759433216763381096732182463314444
e : 7
n : 2481091085270460304866334901105466965563114643354345953479643881533133568730911394358321223515024197137806893331

c : 1117905220184329685115491669660129193823567641747584717324745389247133369892051586020708442213591696394252275222
e : 7
n : 4712783910529936103379120873779889977678125538150303038168690908215575736101910410328062054071689469913314217311

c : 4011807694373533755953737969298207094392151534800021109759935626333076007590674837412972752674043888369509450311
e : 7
n : 4313429171104682135845535135888408777702100383947029650599045058170621937935627239179422012903689519987338580255

c : 1264907659222264937119216486904402540823137171762778004621934637785202454433705015265267657712234253486895809177
e : 7
n : 1930083892114922100729894488747859908280022904521927160627203810397065655994391419728165415858746873054182830644

c : 2889908993543526758823589751984612039343321411434152123869638412250731689945732705502954697233328145256398483844
e : 7
n : 3075412148882763569297184959926774937507794918255030314572932537531492640190578383093162873865887932017994488000

c : 1408662941385567240363983067611804246584602032014382331881504807036850568420814165260359623496096870321778847299
e : 7
n : 3043047798347042619563114265966807177225664120552592989198587299611585801074464877937098353994218768919240651744

c : 2004929920758895590715527609578791340258965237913415140334036049837189311985595783357644385653403788629873170199
e : 7
n : 3548927512653680597428163594290748046391608966306912977142054861281792090269242363996170900030997653181998403033
```



```
└─$ python2 solve.py --verbose -i examples/Basic_Broadcast_Attack.txt
INFO: Here are your plain text:
CTF{Hastad's Broadcast Attack & Chinese Remainder Theorem}
```

到这里我的工作基本就完成了，现在回头看看攻防世界的easy_RSA真的用脚本跑一下就行了，不过学问可当然不是这样做的，但是本人真的对密码学没啥子兴趣，所以只能研究下解题脚本了。

补充：单个n,e,c,q,p,的时候最好用单个参数输入的方式，不要用文本读取的方式，因为文本读取的时候DEBUG显示的十六进制的d有时并不是我们想要的，比如攻防世界这道easy_RSA：
文本输入(N是pq乘积)：

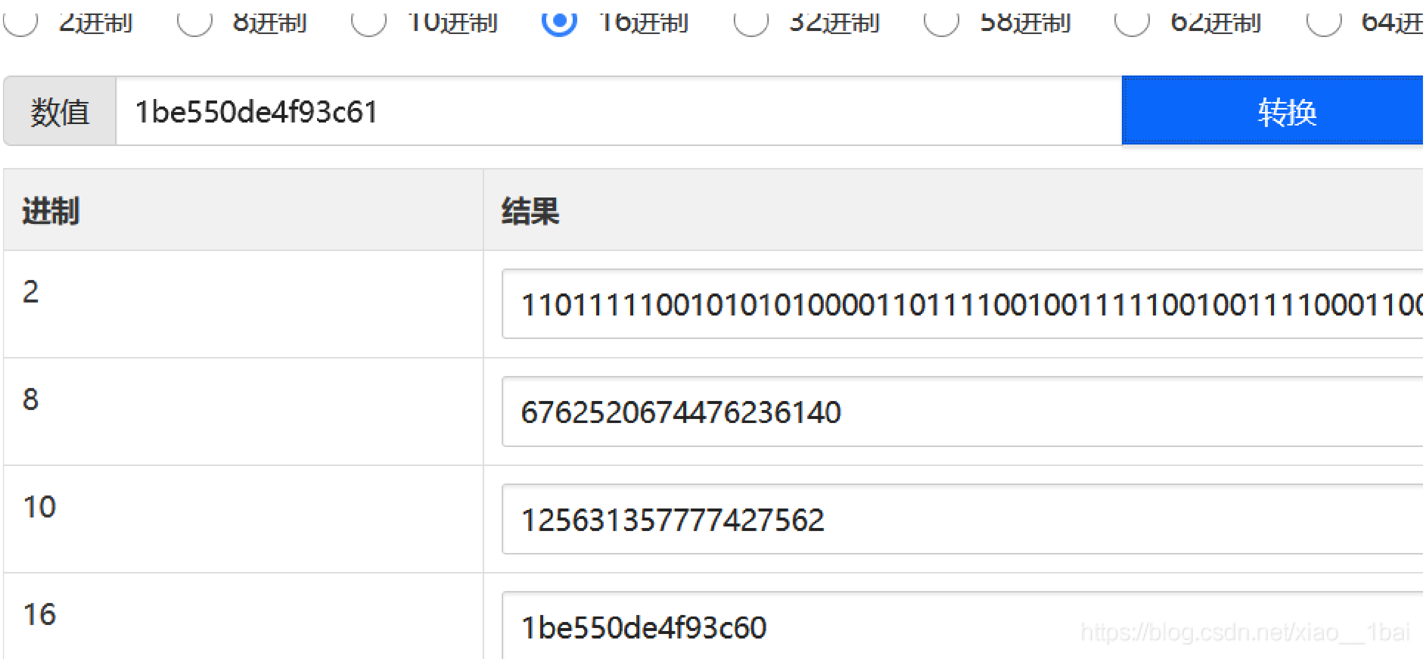


```
n = 2135733555619387051
e = 17
```

```
└─$ python2 CTF-RSA-tool/solve.py -v -i 1.txt                                   1
DEBUG: factor N: try past ctf primes
DEBUG: factor N: try Gimmicky Primes method
DEBUG: factor N: try Wiener's attack
DEBUG: Starting new HTTP connection (1): www.factordb.com:80
DEBUG: http://www.factordb.com:80 "GET /index.php?query=2135733555619387051 HTTP/1.1" 200
9
DEBUG: http://www.factordb.com:80 "GET /index.php?id=4511491 HTTP/1.1" 200 807
DEBUG: http://www.factordb.com:80 "GET /index.php?id=473398607161 HTTP/1.1" 200 836
DEBUG: d = 0x1be550de4f93c61L
```

https://blog.csdn.net/xiao__1bai

而十六进制d：0x1be550de4f93c61L转换十进制出来却多出一个1，(标准答案应该是十进制的125631357777427553才对)



○ 2进制 ○ 8进制 ○ 10进制 ● 16进制 ○ 32进制 ○ 58进制 ○ 62进制 ○ 64进

数值 1be550de4f93c61 转换

进制	结果
2	110111110010101010000110111100100111110010011110001100
8	6762520674476236140
10	125631357777427562
16	1be550de4f93c60

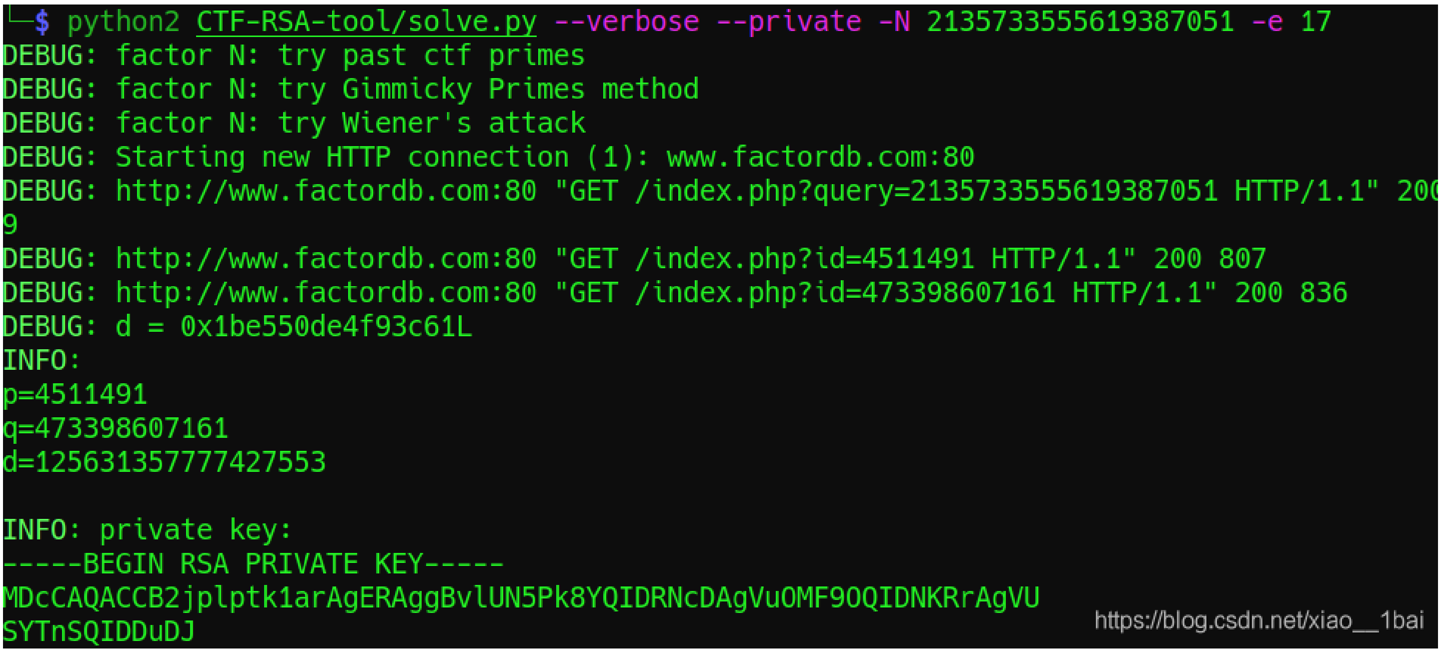
https://blog.csdn.net/xiao__1bai

这里我也搞不懂，还需要深究脚本源码，不过如果直接参数输入就没有这个DEBUG d问题，因为会直接显示出INFO信息，下面两个命令都可用，毕竟N=q*p，只用N也是可以的：

```
python2 CTF-RSA-tool/solve.py --verbose --private -N 2135733555619387051 -e 17
```

```
python2 CTF-RSA-tool/solve.py --verbose --private -N 2135733555619387051 -e 17 -p 473398607161 -q 4511491
```

结果：



```
└─$ python2 CTF-RSA-tool/solve.py --verbose --private -N 2135733555619387051 -e 17
DEBUG: factor N: try past ctf primes
DEBUG: factor N: try Gimmicky Primes method
DEBUG: factor N: try Wiener's attack
DEBUG: Starting new HTTP connection (1): www.factordb.com:80
DEBUG: http://www.factordb.com:80 "GET /index.php?query=2135733555619387051 HTTP/1.1" 200
9
DEBUG: http://www.factordb.com:80 "GET /index.php?id=4511491 HTTP/1.1" 200 807
DEBUG: http://www.factordb.com:80 "GET /index.php?id=473398607161 HTTP/1.1" 200 836
DEBUG: d = 0x1be550de4f93c61L
INFO:
p=4511491
q=473398607161
d=1256313577777427553

INFO: private key:
-----BEGIN RSA PRIVATE KEY-----
MDcCAQACCB2jplptk1arAgERAggBvlUN5Pk8YQIDRNcDAgVuOMF9OQIDNKRrAgVU
SYTnSQIDDuDJ
```

这篇博客对你有什么用？其实对你没什么用，是写给我自己看的，当然你要是能从中提取到对自己有帮助的信息我也很欣慰。