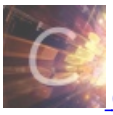
CTF Crypto RSA算法 入门总结（全）

原创

你们这样一点都不可耐 于 2020-08-06 18:48:44 发布 3145 收藏 58

分类专栏： CTF 文章标签： 算法 rsa 加密解密 CTF 安全

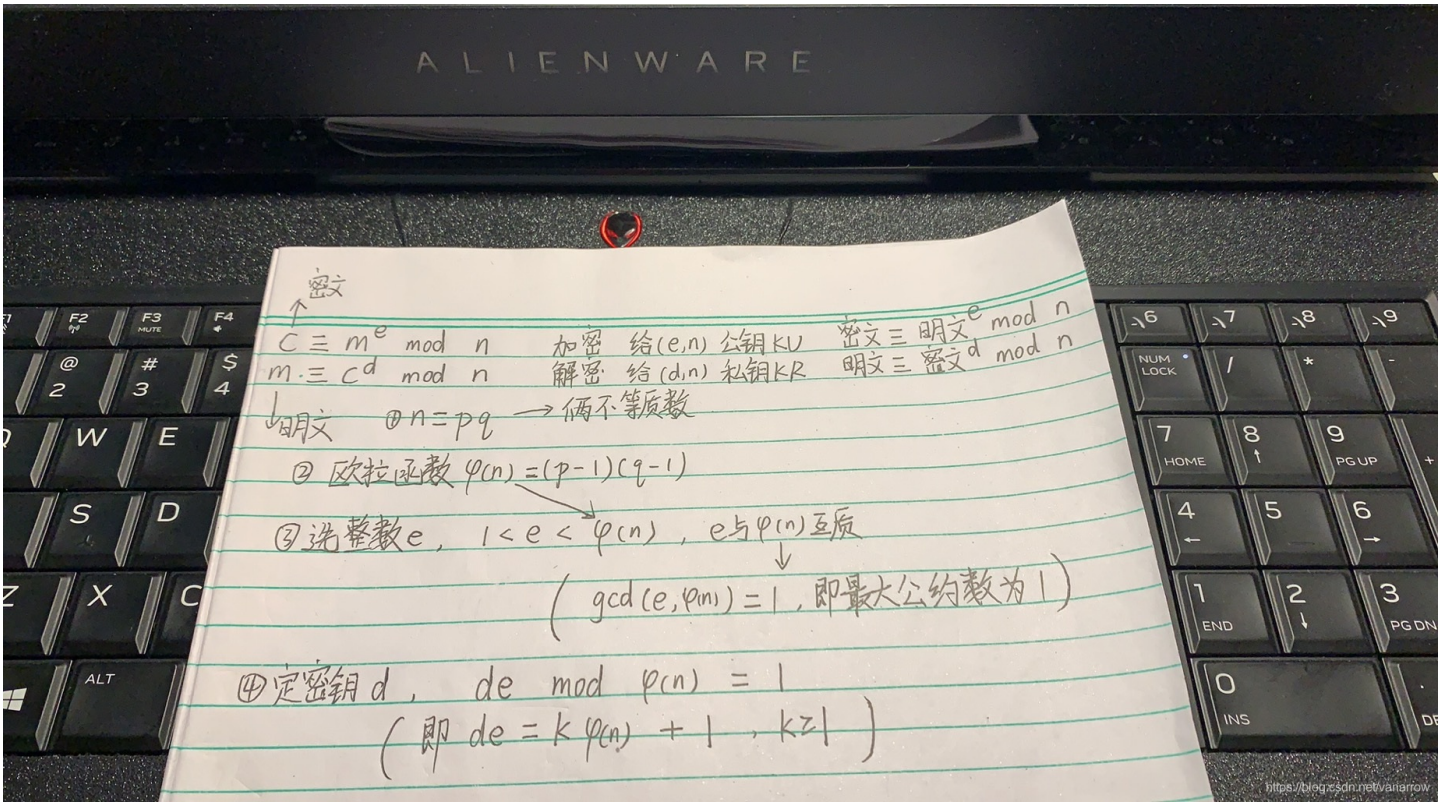
本文链接：https://blog.csdn.net/vanarrow/article/details/107846987



RSA算法—— CTF 新手入门 （看完就能做题）

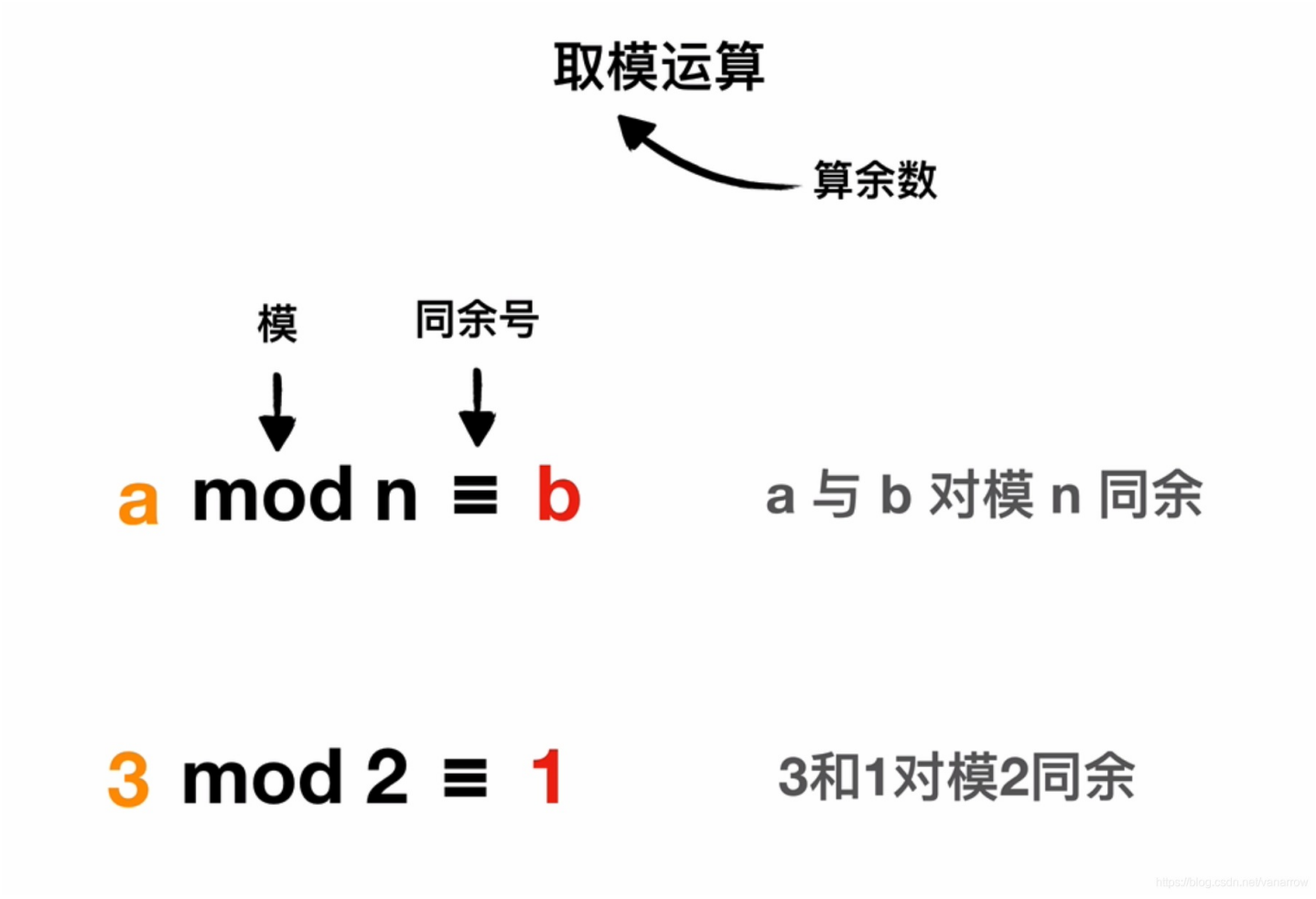
就像小时候刚学各种方程一样，一上来就很吓唬人，觉得晦涩难懂，懂了就觉得很简单，有个循序渐进，由浅入深的过程。

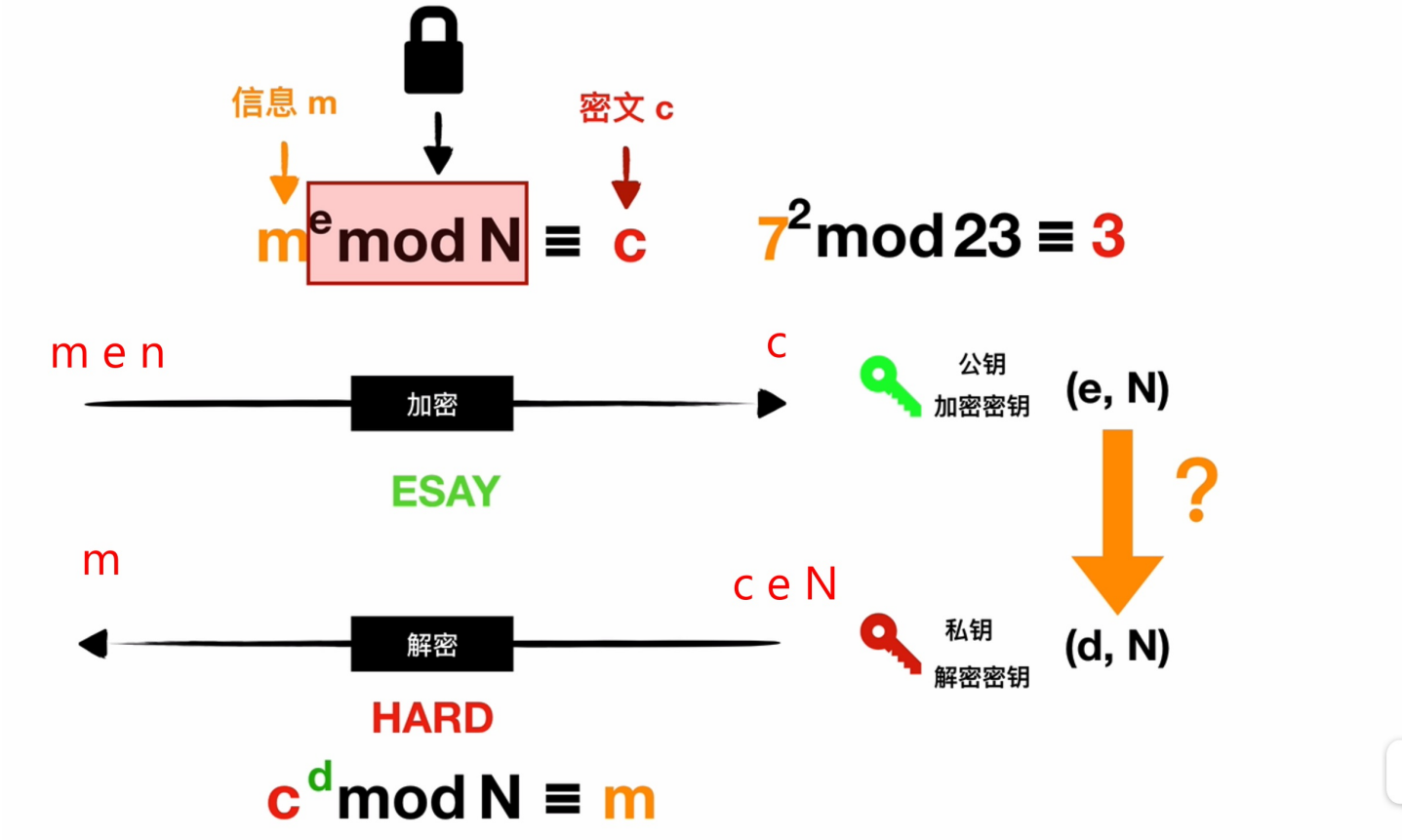
自己总结的笔记

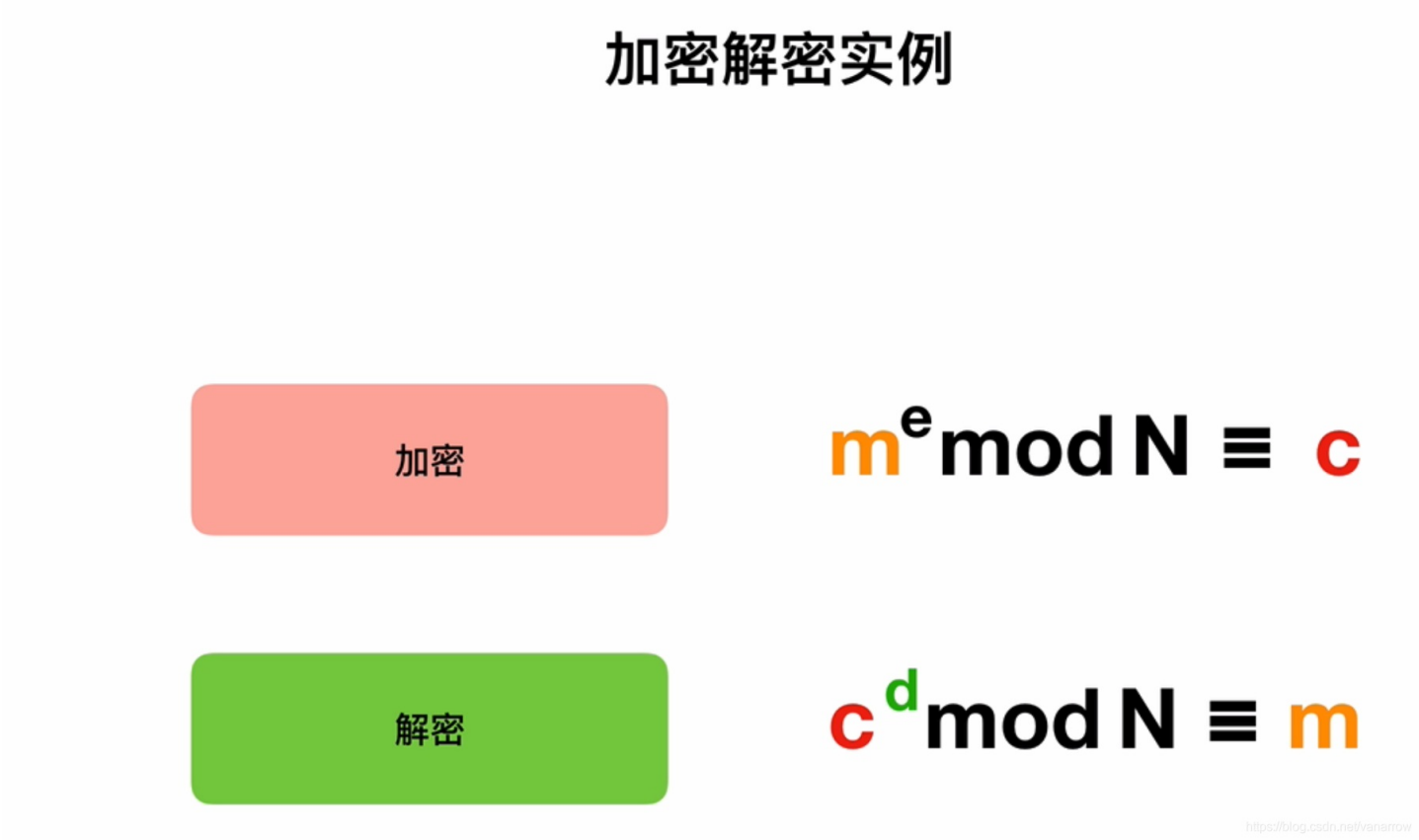


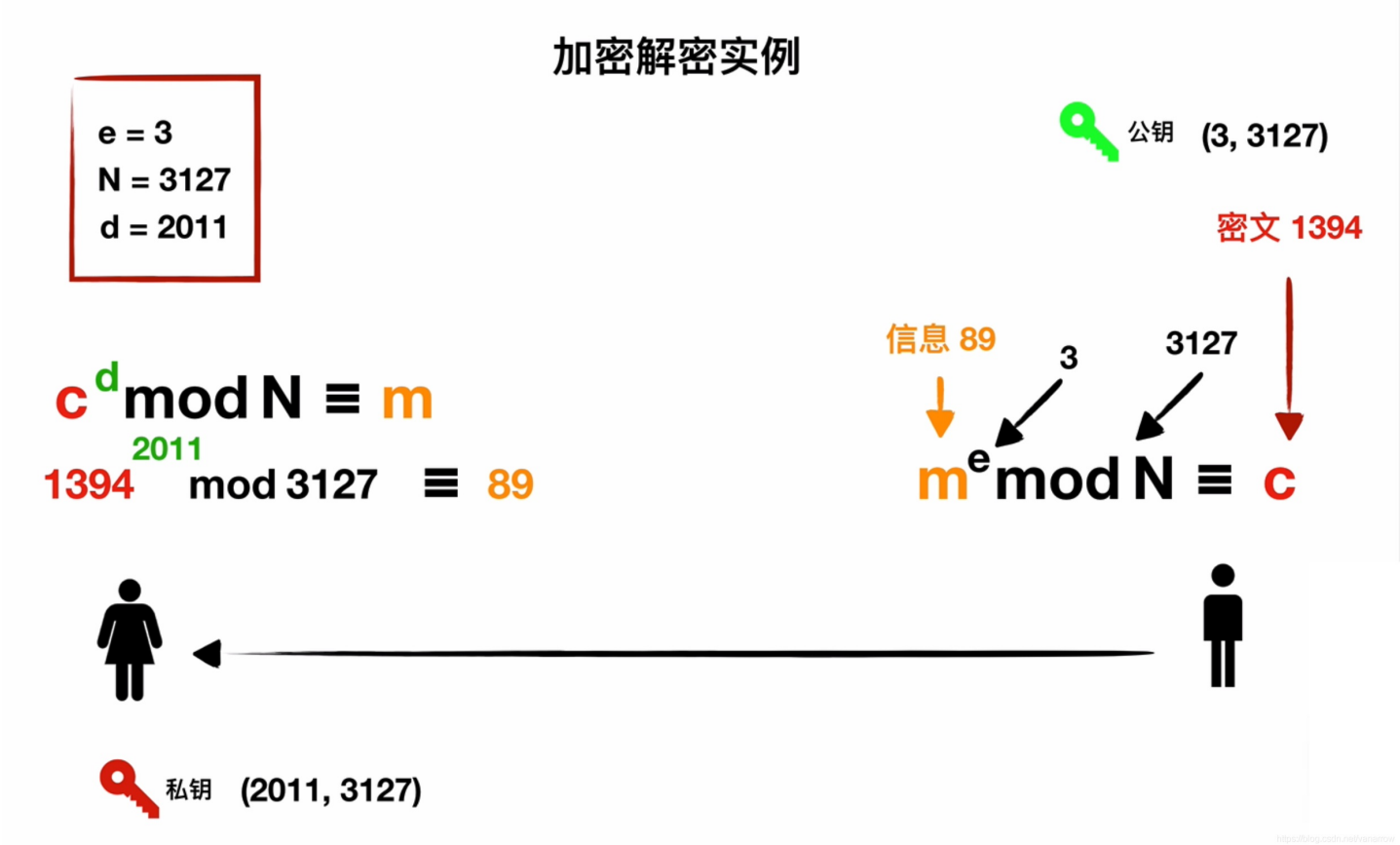
一、初步认识

（1）B站简单认识


取模运算
算余数
模
同余号
a mod n ≡ b
a 与 b 对模 n 同余
3 mod 2 ≡ 1
3和1对模2同余


信息 m
密文 c
m^e mod N ≡ c
7^2 mod 23 ≡ 3
m e n
加密
c
公钥
加密密钥
(e, N)
ESAY
?
m
c e N
解密
私钥
解密密钥
(d, N)
HARD
c^d mod N ≡ m





通俗理解：

选两个大质数p和q，且p!=q，计算N=p*q*

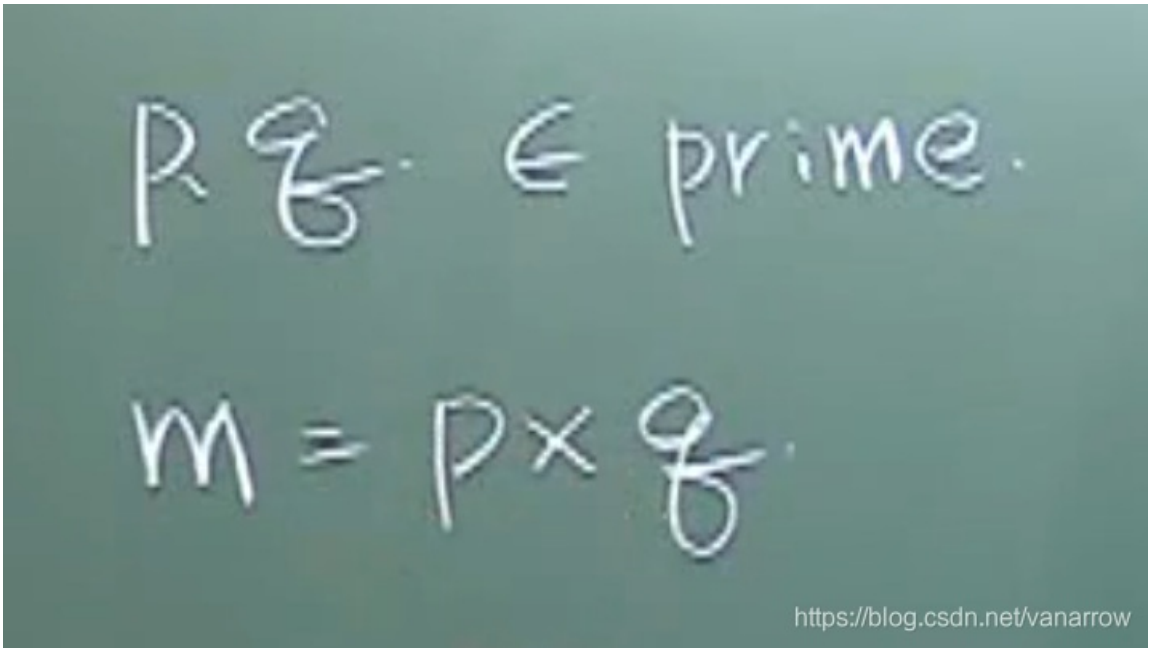
计算N的欧拉函数 ∮(n)=(p-1)(q-1)

选一个e满足 1与<e<∮(n)，且与∮(n)互质

找到d，使ed/ϕ(n)=x……1（x不重要，重要的是余数为1）
（n，e）为公钥，（n，d）为私钥
加密：$C = M^e \pmod n$
解密：$M = C^d$（mod n）

（2）不到20分钟让你学会RSA原理之数学质数与RSA密码算法

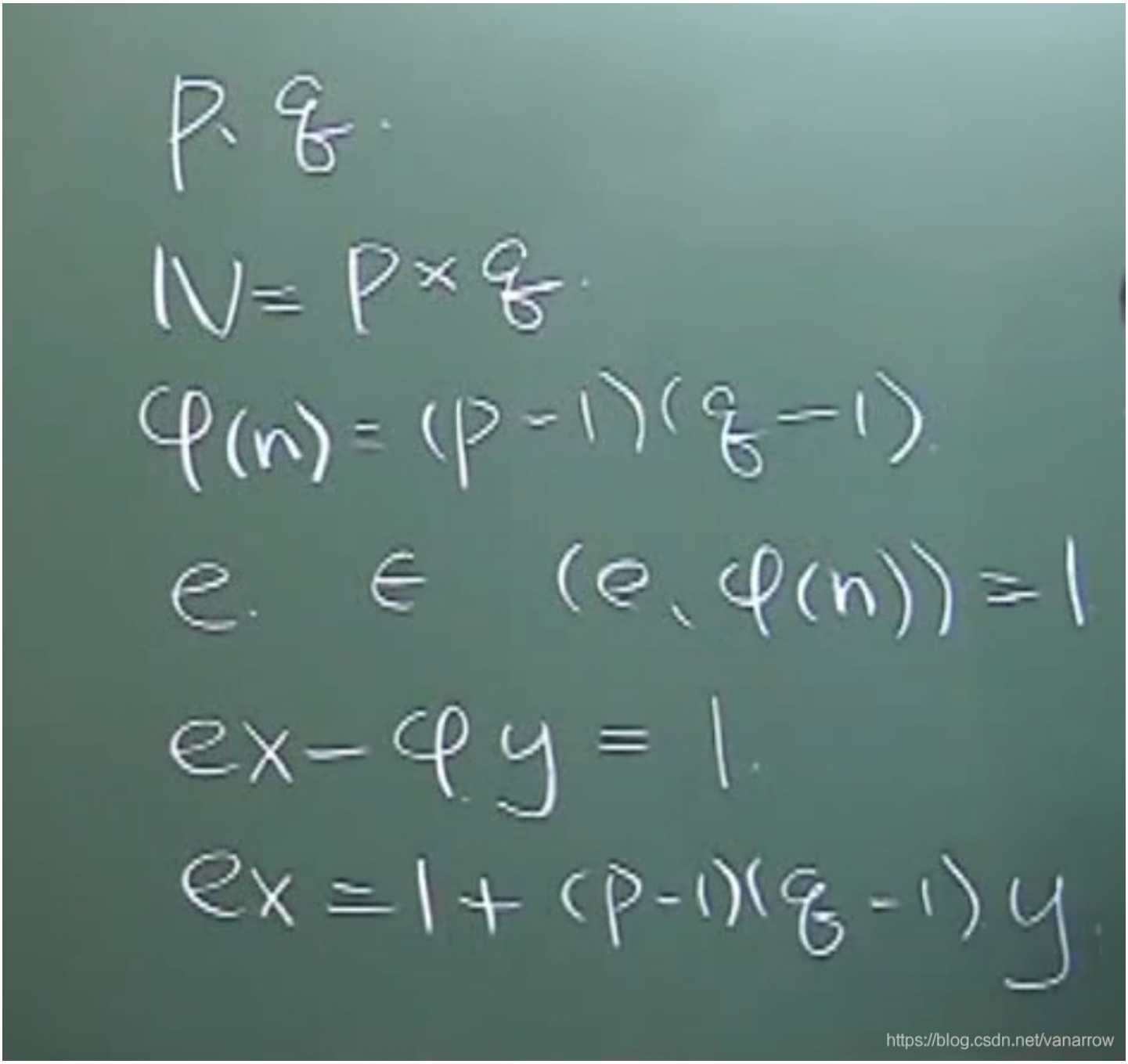
不到20分钟让你学会RSA原理之数学质数与RSA密码算法



用到数论中质数的性质
p、q是很大的质数，乘积m是更大的质数
p、q很容易算出m，但从m很难推出p、q
数越大，使得m分解出p、q的成本超出秘密本身的成本，就是安全的

传统密码演算法——映射（对应奇怪的符号）

公钥演算法——同余（更抽象，圆运算）



ϕ(n) : 不大于n，与n互质的所有整数的个数

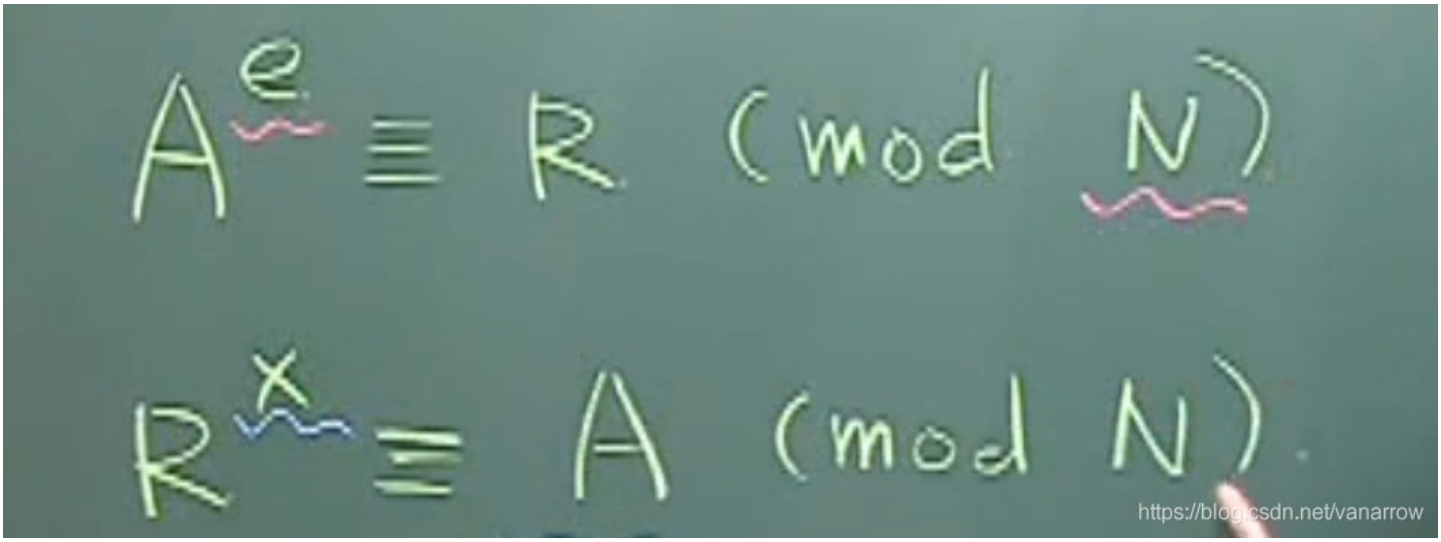
以N=6为例，
N=pxq
6=2x3
ϕ(n)=1x2=2，有两个，就是1，5

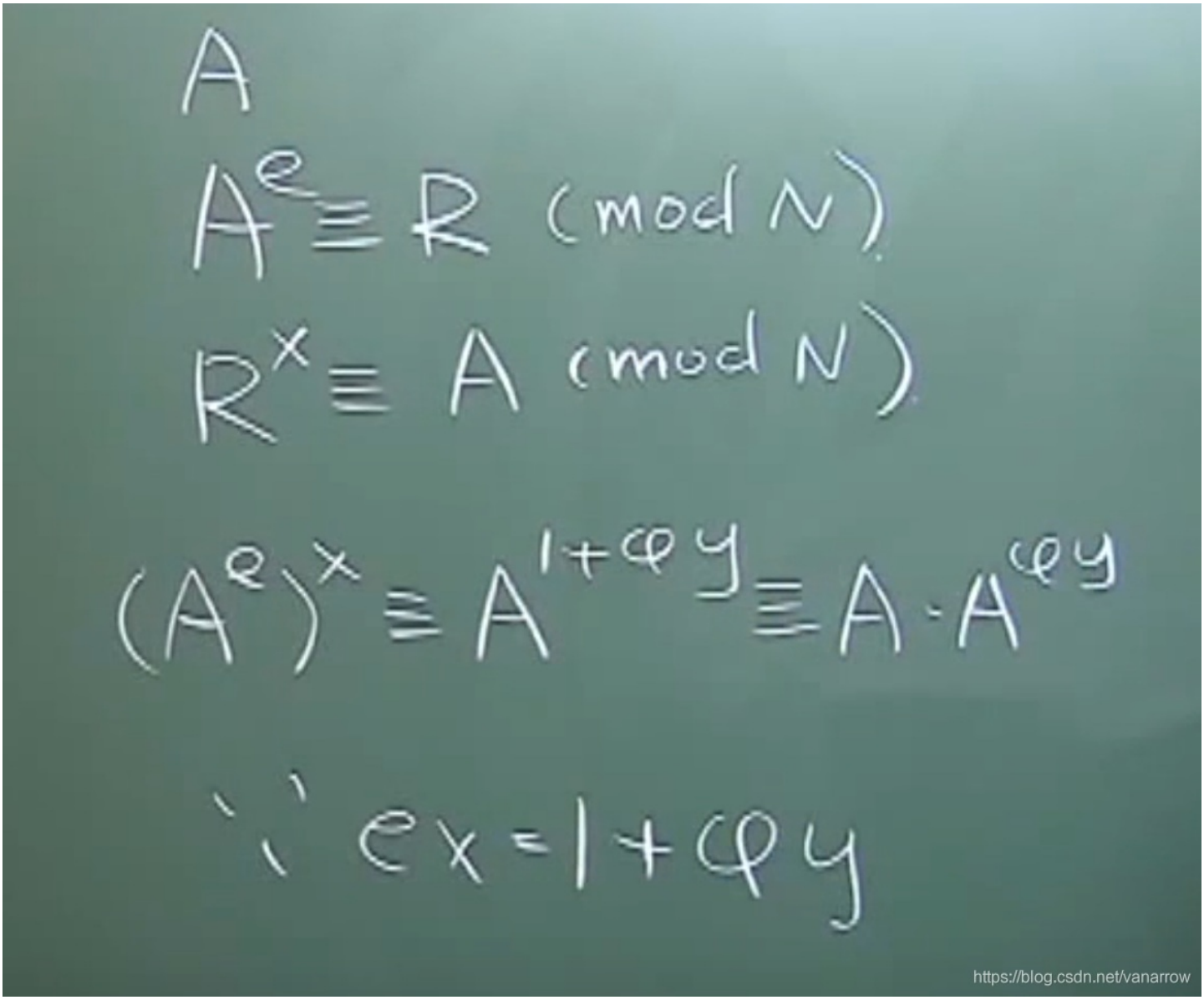
第四行，找比ϕ(n)小，且互质的的自然数

根据辗转相除法，写出最下面的两个式子，找到x，y

最后都丢掉，只留下N，e，x

对外界发布N，e，自己留私钥x





A为需要传输的明文，A的e次方除以N算出余数R
拿到余数R，知道x，R的x次方，除以N，算出明文A

Fermat's little theorem

$$a^{p-1} \equiv 1 \pmod{P}$$

费马小定理（先出现，欧拉再推广）

Euler-Fermat theorem

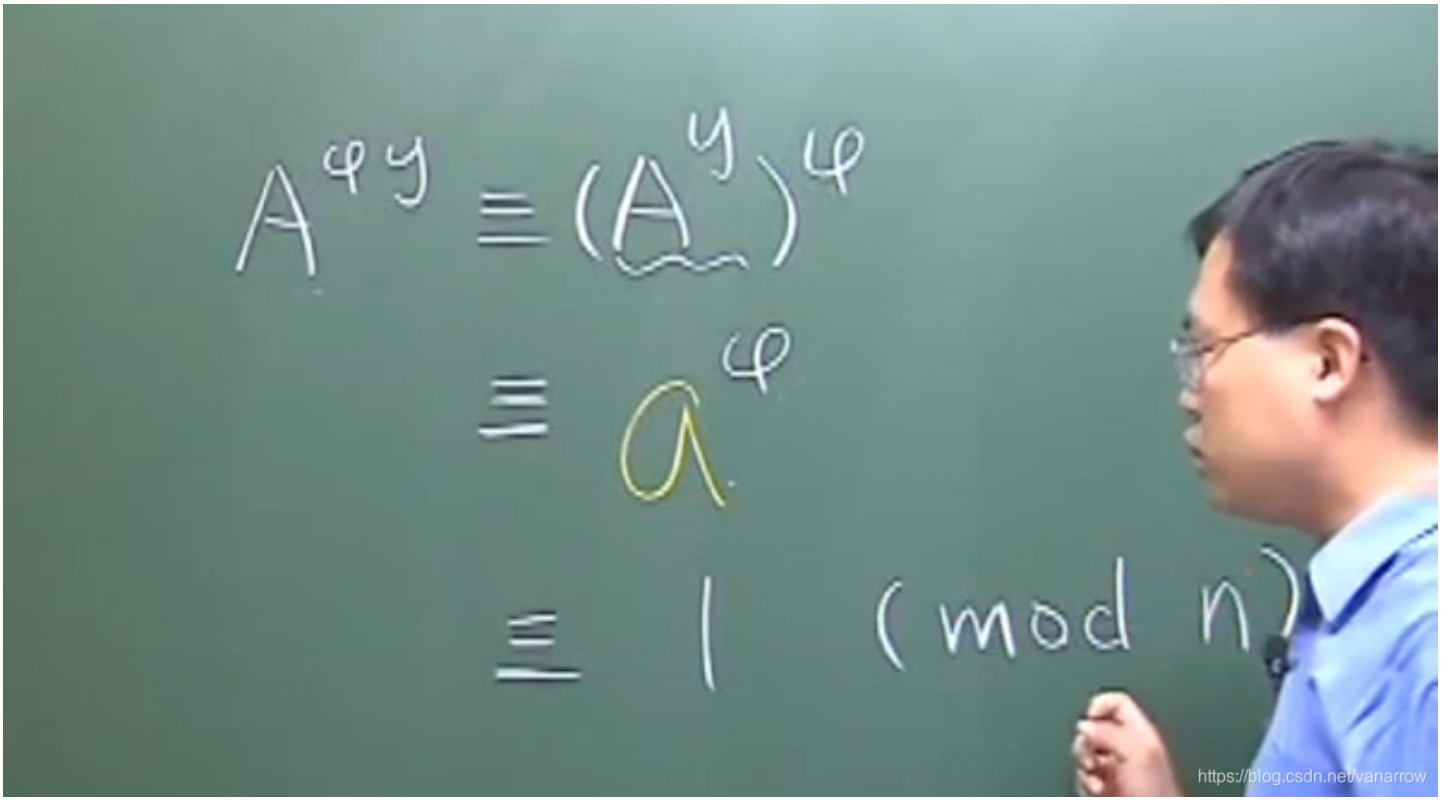
Euler totient function $\varphi(n)$

$$\gcd(a,n) = 1$$

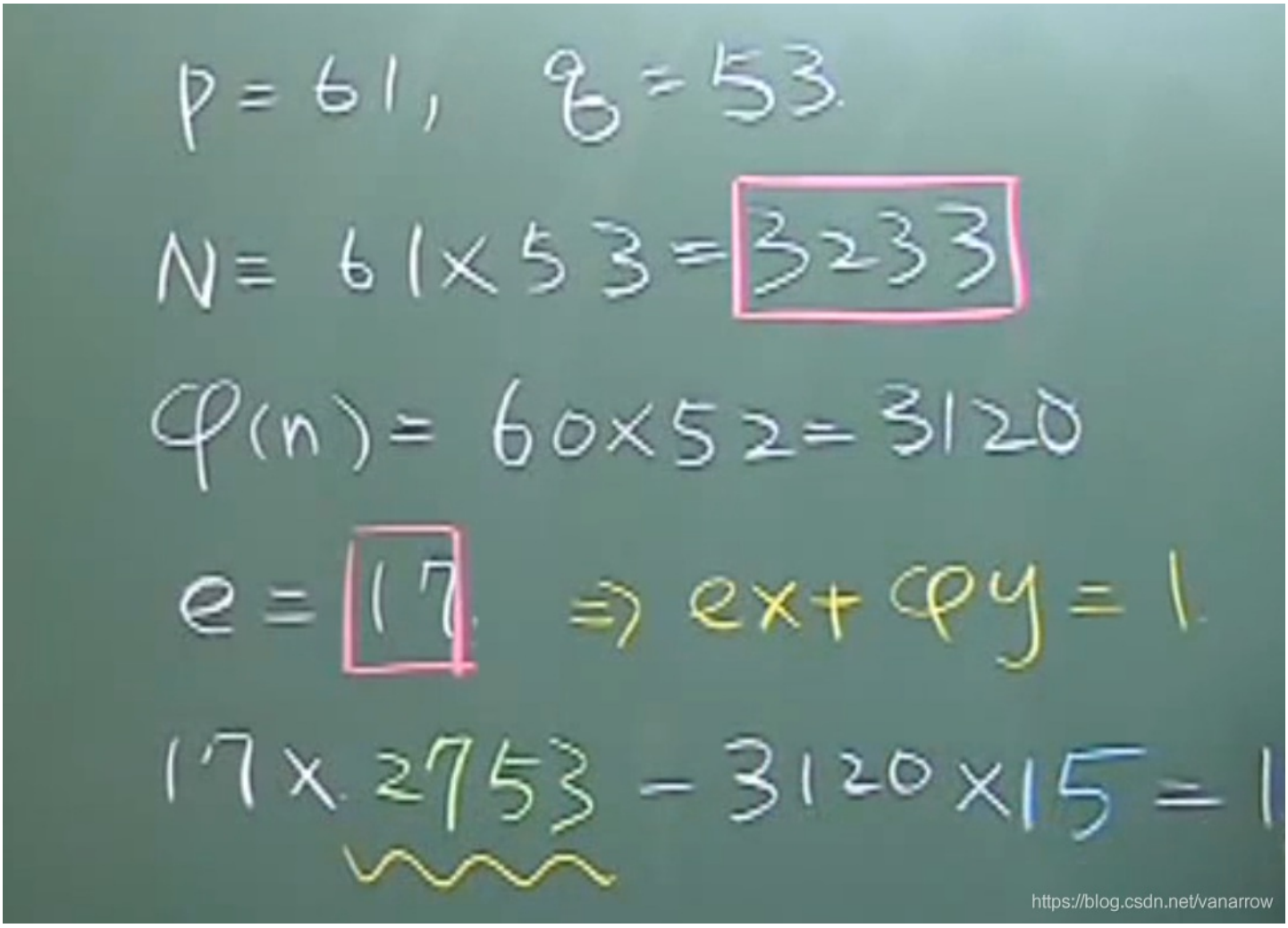
$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

A的φ(y)等于1（Euler totient function，即欧拉函数）

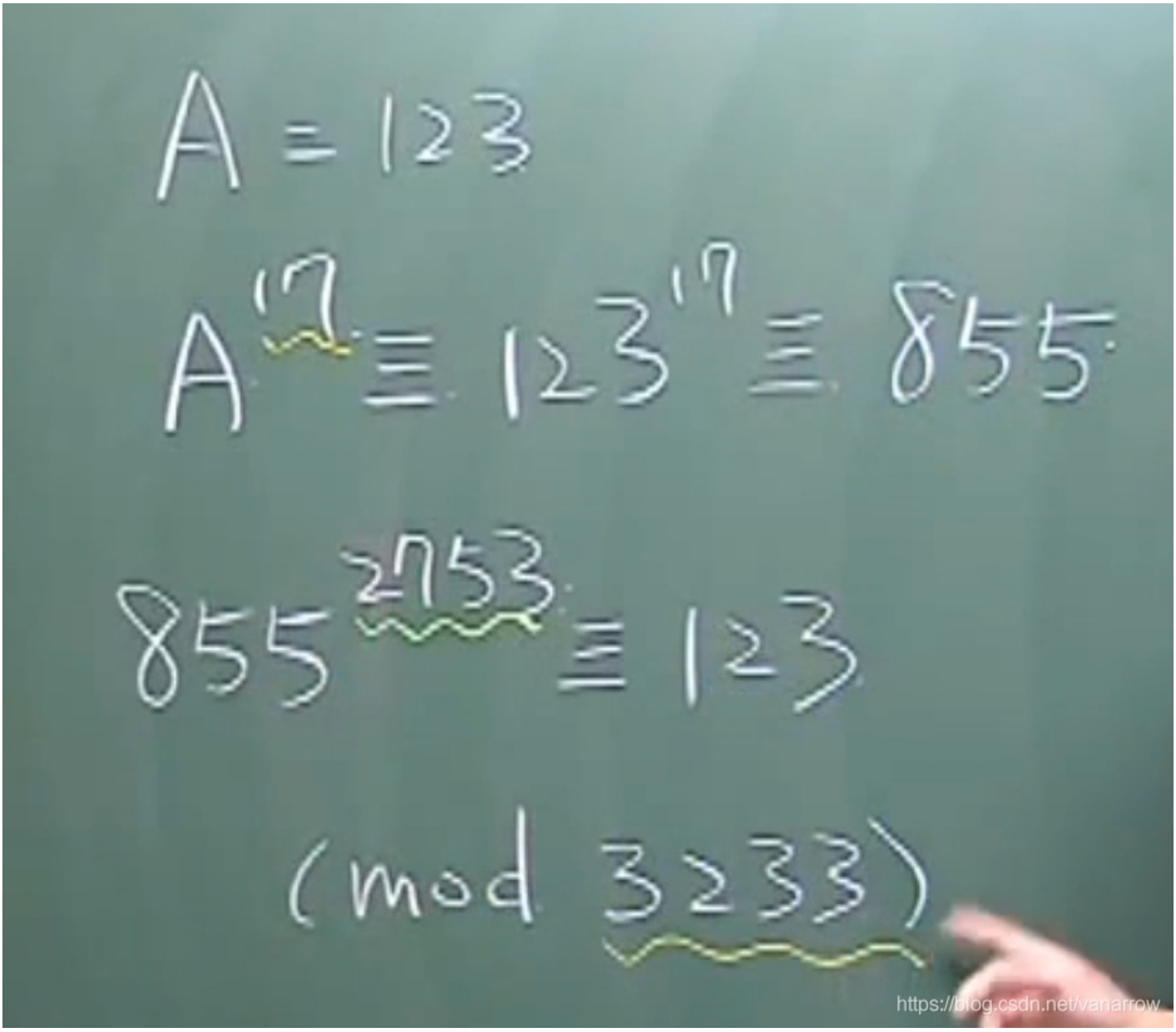
欧拉-费马定理：
不大于n，与n互质的自然数有多少个？



A和n互质，多少次结果还是互质



公开e，N；公钥是（e,N）即（17，3233）
私藏；d私钥是（d，N） 即（2753，3233）



二、简单理解

1. 百度解释

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]

$$c = E(m) = m^e mod n$$

（6）将密文c解密为明文m，解密算法为 [5]

$$m = D(c) = c^d mod n$$

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] 。

迷迷糊糊，懵懵懂懂，马马虎虎

密钥组成与加解密公式

公钥 KU	n：质数p和质数q的乘积（p和q必须保密） e：与$(p-1)\times(q-1)$互质
私钥KR	n：同公钥n d：$e^{-1}\pmod{(p-1)(q-1)}$
加密	$c = m^e \bmod n$
解密	$m = c^d \bmod n$

看看乙是如何计算密钥（公钥和私钥）的

1. 随机选择两个不相等的质数p和q（乙选择了61和53）
2. 计算p和q的乘积$n = p \times q = 61 \times 53 = 3233$
3. 根据本文"欧拉函数"介绍过的公式

$\varphi(n)=(p-1)(q-1)$

代入计算n的欧拉函数值

$\varphi(3233)=(61-1)\times(53-1)=60\times52=3120$

4. 随机选择一个整数e，条件是$1<e<\varphi(n)$，且e与$\varphi(n)$互质

乙就在1到3120之间，随机选择了17

5. 因为e与$\varphi(n)$互质，根据求模反元素的公式计算e，对于e的模反元素d有：

$ed\equiv1\ (\mathrm{mod}\ \varphi(n))$

这个式子等价于

$(ed-1)/\varphi(n)=k$（k为任意正整数）

即

$ed-k\varphi(n)=1$，代入数据得：

$17d-3120k=1$

实质上就是对以上这个二元一次方程求解

得到一组解为：$(d,k)=(2753,-15)$

6. 将n和e封装成公钥，n和d封装成私钥

$n=3233$，$e=17$，$d=2753$

所以公钥就是$(3233,17)$，私钥就是$(3233,2753)$

其中，n的长度就是密钥长度，3233写成二进制是110010100001

一共有12位，所以这个密钥就是12位

实际应用中，RSA密钥一般是1024位，重要场合则为2048位

大白话，通俗描述：

（1）选择一对不同的、足够大的素数p，q。

（2）计算n=pq。

（3）计算f(n)=(p-1)(q-1)，同时对p, q严加保密，不让任何人知道。

（4）找一个与f(n)互质的数e，且1<e<f(n)。

（5）计算d，使得de≡1 mod f(n)。

这个公式也可以表达为

原文这个是错的！！！ d ≡e-1 mod f(n) 应该是把e除到分母了

$d\equiv e^{-1}\ \mathrm{mod}\ f(n)$

这里要解释一下，≡是数论中表示同余的符号。公式中，≡符号的左边必须和符号右边同余，也就是两边模运算结果相同。显而易见，不管f(n)取什么值，符号右边1 mod f(n)的结果都等于1；符号的左边d与e的乘积做模运算后的结果也必须等于1。这就需要计算出d的值，让这个同余等式能够成立。
（6）公钥KU=(e,n)，私钥KR=(d,n)。
（7）加密时，先将明文变换成0至n-1的一个整数M。若明文较长，可先分割成适当的组，然后再进行交换。

Gamma(z) = \int_0^\infty t^{z-1}e{-t}dt,.
$$

2.带你彻底理解RSA算法原理

带你彻底理解RSA算法原理

(1)加密：

$$密文 = 明文^{E} mod N$$

$$公钥 = (E, N)$$

RSA加密是对明文的E次方后除以N后求余数的过程。

只要知道E和N任何人都可以进行RSA加密了，所以说E、N是RSA加密的密钥，也就是说E和N的组合就是公钥，用(E,N)来表示公钥。
（E、N有特别要求，暂时不管）

E：Encryption 加密
N：Number 数字

(2) 解密

$$明文 = 密文^{D} mod N$$

RSA解密，密文进行D次方后除以N的余数就是明文。

$$私钥 = (D, N)$$

知道D和N就能进行解密密文了，所以D和N的组合就是私钥

RSA的加密方式和解密方式是相同的

加密是求 E次方的mod N
解密是求 D次方的mod N

D：Decryption 解密
N：Number 数字

公钥	(E，N)
私钥	(D，N)
密钥对	(E，D，N)
加密	密文 = 明文$^{E}modN$
解密	明文 = 密文$^{D}modN$

后面作者说的很详细了

4. 生成密钥对

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

1. 求N
2. 求L（L为中间过程的中间数）
3. 求E
4. 求D

4.1 求N

准备两个质数p，q。这两个数不能太小，太小则会容易破解，将p乘以q就是N

$$N = p * q$$

4.2 求L

L 是 p - 1 和 q - 1的最小公倍数，可用如下表达式表示

$$L = lcm\ (\ p\ -\ 1\ ,\ q\ -\ 1\ )$$

4.3 求E

E必须满足两个条件：E是一个比1大比L小的数，E和L的最大公约数为1
用gcd(X,Y)来表示X，Y的最大公约数则E条件如下：

> **1 < E < L**
>
> **gcd（E，L）=1**

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

4.4 求D

数D是由数E计算出来的。D、E和L之间必须满足以下关系：

> **1 < D < L**
>
> **E * D mod L = 1**

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

求N	N= p * q；p，q为质数
求L	L=lcm（p - 1，q - 1）；L为p - 1、q - 1的最小公倍数
求E	1 < E < L，gcd（E，L）=1；E，L最大公约数为1（E和L互质）
求D	1 < D < L，E * D mod L = 1

5 实践下吧

我们用具体的数字来实践下RSA的密钥对对生成，及其加解密对全过程。为方便我们使用较小数字来模拟。

5.1 求N

我们准备两个很小对质数，
p = 17
q = 19
N = p * q = 323

5.2 求L

L = lcm（p - 1， q - 1）= lcm(16，18) = 144
144为16和18对最小公倍数

5.3 求E

求E必须要满足2个条件：1 < E < L，gcd（E，L）=1
即1 < E < 144，gcd（E，144） = 1
E和144互为质数，5显然满足上述2个条件
故E = 5

此时**公钥=(E，N) = （5，323）**

5.4 求D

求D也必须满足2个条件：1 < D < L，E * D mod L = 1
即1 < D < 144，5 * D mod 144 = 1
显然当D = 29 时满足上述两个条件
1 < 29 < 144
5 * 29 mod 144 = 145 mod 144 = 1
此时**私钥 = （D，N） = （29，323）**

5.5 加密

准备的明文必须时小于N的数，因为加密或者解密都要mod N其结果必须小于N
假设明文 = 123
则 密文 = 明文$^{E} mod N = 123^{5} mod 323 = 225$

5.6 解密

明文 = 密文$^{D} mod N = 225^{29} mod 323 = 123$
解密后的明文为123。

好了至此RSA的算法原理已经讲解完毕，是不是很简单?

学的差不多了？来深入了解一下！

三、深入理解

就讲的很好！

1.阮一峰——RSA算法原理（一）

阮一峰——RSA算法原理（一）

2.阮一峰——RSA算法原理（二）

阮一峰——RSA算法原理（二）

四、附必要的数学基础：

基础整合
黄映焜的博客园 (基础知识整合)

公式图表

公钥 KU	n：两素数 p 和 q 的乘积（p 和 q 必须保密） e：与（p-1）(q-1) 互质
私钥 KR	d：e^{-1} (mod(p-1)(q-1)) n：
加密	$C \equiv m^{e} \bmod n$
解密	$m \equiv c^{d} \bmod n$

质数（素数）：
一个大于1的自然数，除了1和它本身外，不能被其他自然数整除（除0以外）的数称之为质数（素数）；否则称为合数。

互质数：公约数只有1的两个数，叫做互质数。

判别方法主要有以下几种（不限于此）：
（1）两个质数一定是互质数。例如，2与7、13与19。
（2）一个质数如果不能整除另一个合数，这两个数为互质数。例如，3与10、5与 26。
（3）1不是质数也不是合数，它和任何一个自然数在一起都是互质数。如1和9908。
（4）相邻的两个自然数是互质数。如 15与 16。
（5）相邻的两个奇数是互质数。如 49与 51。
（6）大数是质数的两个数是互质数。如97与88。
（7）小数是质数，大数不是小数的倍数的两个数是互质数。如 7和 16。
（8）两个数都是合数（二数差又较大），小数所有的质因数，都不是大数的约数，这两个数是互质数。如357与715，357=3×7×17，而3、7和17都不是715的约数，这两个数为互质数。等等。

5. 模运算：模运算是整数运算，有一个整数m，以n为模做模运算，即m mod n。用m去被n整除，只取所得的余数作为结果，就叫做模运算。例如，10 mod 3=1；26 mod 6=2；28 mod 2 =0。

a ≡ b (mod n)

~~原文表述错误：给定一个正整数m，如果两个整数a和b满足a-b能被m整除，即(a-b)modm=0，那么就称整数a与b对模m同余，记作a=b(modm)，同时可成立amodm=b ，再次提醒注意，同余与模运算是不同的~~

查了百度，不是非要0，而且不能那么说吧。

数论中的重要概念。给定一个正整数m，如果两个整数a和b满足a-b能够被m整除，即(a-b)/m得到一个整数，那么就称整数a与b对模m同余，记作a≡b(mod m)。对模m同余是整数的一个等价关系。

≡ 表示同余，即MOD的符号，表示模。

a和b除以n后余数相同，读作a与b同余,mod为n。

如果我们以3为模，则4与7就是关于3同余的。记作7≡4 (mod 3)，8≡5 (mod 3)，26≡5 (mod 7)。

简单理解，当取余，python立面自己试试看，1%20=1，千万不要当作是20

```
>>> 1%20
1
```

~~同余涉及：欧拉定理，费马小定理，中国剩余定理（孙子定理）~~

假设用户A需要将明文“key”通过RSA加密后传递给用户B，过程如下：

（1）设计公私密钥(e,n)和(d,n)。

令p=3，q=11，得出n=p×q=3×11=33；f(n)=(p-1)(q-1)=2×10=20；取e=3，（3与20互质）则e×d≡1 mod f(n)，即3×d≡1 mod 20。

d怎样取值呢？可以用试算的办法来寻找。

试算结果见下表：

d	e×d = 3×d	(e×d) mod (p-1)(q-1) = (3×d) mod 20
1	3	3
2	6	6
3	9	9
4	12	12
5	15	15
6	18	18
7	21	1
8	24	3
9	27	6

当d=7时，3x7 mod 20 = 1 mod 20 = 1，即 e×d≡1 mod f(n)同余等式成立。因此，可令d=7。从而我们可以设计出一对公私密钥，加密密钥（公钥）为：KU =(e,n)=(3,33)，解密密钥（私钥）为：KR =(d,n)=(7,33)。

（2）英文数字化。

将明文信息数字化，并将每块两个数字分组。假定明文英文字母编码表为按字母顺序排列数值，即：

字母	a	b	c	d	e	f	g	h	i	j	k	l	m
码值	01	02	03	04	05	06	07	08	09	10	11	12	13
字母	n	o	p	q	r	s	t	u	v	w	x	y	z
码值	14	15	16	17	18	19	20	21	22	23	24	25	26

则得到分组后的key的明文信息为：11，05，25。

（3）明文加密

用户加密密钥(3,33) 将数字化明文分组信息加密成密文。由C≡Me(mod n)得：

$$M1 \equiv (C1)^d (\mathrm{mod}\ n) = 11^7 (\mathrm{mod}\ 33) = 11$$
$$M2 \equiv (C2)^d (\mathrm{mod}\ n) = 31^7 (\mathrm{mod}\ 33) = 05$$
$$M3 \equiv (C3)^d (\mathrm{mod}\ n) = 16^7 (\mathrm{mod}\ 33) = 25$$

因此，得到相应的密文信息为：11，31，16。

（4）密文解密。

用户 B 收到密文，若将其解密，只需要计算$M \equiv C^d (\mathrm{mod}\ n)$，即：

$$M1 \equiv (C1)^d (\mathrm{mod}\ n) = 11^7 (\mathrm{mod}\ 33) = 11$$
$$M2 \equiv (C2)^d (\mathrm{mod}\ n) = 31^7 (\mathrm{mod}\ 33) = 05$$
$$M3 \equiv (C3)^d (\mathrm{mod}\ n) = 16^7 (\mathrm{mod}\ 33) = 25$$

用户 B 得到明文信息为：11，05，25。根据上面的编码表将其转换为英文，我们又得到了恢复后的原文“key”。

6. 欧拉函数

```
任意给定正整数n，计算在小于等于n的正整数之中，有多少个与n构成互质关系？
计算这个值的方法就叫做欧拉函数，以ϕ(n)表示.
```

例如，在1到8之中，与8形成互质关系的是1、3、5、7，所以φ(n)=4

在RSA算法中，我们需要明白欧拉函数对以下定理成立

```
如果n可以分解成两个互质的整数之积，即n=p×q，则有：ϕ(n)=ϕ(pq)=ϕ(p)ϕ(q);
根据“大数是质数的两个数一定是互质数”可以知道：
一个数如果是质数，则小于它的所有正整数与它都是互质数；
所以如果一个数p是质数，则有：ϕ(p)=p-1
```

由上易得，若我们知道一个数n可以分解为两个质数p和q的乘积，则有

φ(n)=(p-1)(q-1)

五、CTF例题

一起来做做你曾经望而生畏的简单题

1. ctf.show crypto4

题目
https://ctf.show/challenges#crypto4
解题视频
https://www.bilibili.com/video/BV12t4y1y75J?from=search&seid=16581163553083825362

> p=447685307 q=2037 e=17
> flag{d}

> https://buuoj.cn/challenges#RSA
> 同BUUCTF Crypto RSA
> 在一次RSA密钥对生成中，假设p=473398607161，q=4511491，e=17
> 求解出d作为flga提交

求私钥d

```python
import gmpy2

p = 447685307
q = 2037
e = 17

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

> 53616899001

```
'''
import gmpy2
n=pq
phi =(p-1)(q-1)
ed=1 mod phi

常用的库
import libnum
libnum.n2s(n) 16进制转字符串
libnum.s2n(s) 字符串转16进制
gmpy2.mpz(n)初始化一个大整数
n=invert(m,phi)求mod phi的逆元
pow(m,e,n)求c^d mod n
gmpy2.is_prime(n) 素性检测
gmpy2.gcd(a,b)  欧几里得算法，最大公约数
gmpy2.gcdext(a,b)  扩展欧几里得算法
gmpy2.iroot(x,n) x开n次根
'''

p=473398607161
q=4511491
phi=(p-1)*(q-1)
e=17
d=gmpy2.invert(e,phi)
print(d)
```

2. ctf.show crypto5

> https://ctf.show/challenges#crypto5
> p=447685307 q=2037 e=17 c=704796792
> 提交flag{m}
> //
> 同 BUUCTF rsarsa
> Math is cool! Use the RSA algorithm to decode the secret message, c, p, q, and e are parameters for the RSA algorithm.
> p =
> 964842302901051567659055174001042653494573763923573980064398935203985250729849139956103500916342705037010757073363335091169128029777716020062528166537848 3
> q =
> 118748438379802970320924058486536568527609101545433809076500401907042833589092085782510630477324439922306479038875100655479473135432993032619860534865694 07
> e = 65537 c =
> 832082989951746041747735902982036393605400248712561268928896613457424033149298619391004926666056473166465764865262174570063768422808697285817267464015837058999417682141387422596893348407356335530538876418476511737762518202930872128856701803674068074067659236389731613758173927377478327627516901044238690190 34
> Use RSA to find the secret message

```
import gmpy2

p=447685307
q=2037
e=17
c=704796792

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

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

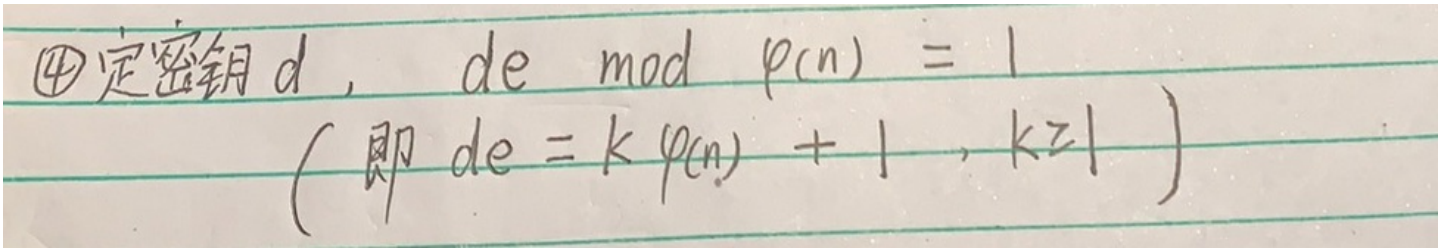
```
import gmpy2

p = 964842302901051567659055174001042653494573763923573980064398935203985250729849139956103500916342705037010757
073363335091169128029777716020062528166537848 3
q = 118748438379802970320924058486536568527609101545433809076500401907042833589092085782510630477324439922306479
038875100655479473135432993032619860534865694 07
e = 65537
c = 832082989951746041747735902982036393605400248712561268928896613457424033149298619391004926660564731664657648652621745700637684228086972858172674640158370589994176821413874225968933484073563355305388764184765117377625182029308721288567018036740680740676592363897316137581739273774783276275169010442386901903 4

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

m = pow(c,d,n)

print(m)
```

> 904332399012

3. XCTF easy_RSA

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

```
import gmpy2

p=473398607161
q=4511491
e=17

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

print(d)
```

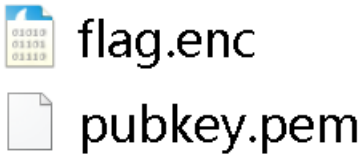


(ϕ (n)+1) / e = d

```
p=473398607161
q=4511491
e=17

n = p*q
phi_n = (p-1)*(q-1)
d = (phi_n+1)//17

print(d)
```

> 125631357777427553

4. XCTF Normal_RSA



enc: flag的密文

pem: 公钥 public key

PEM是OpenSSL和许多其他SSL工具的标准格式，OpenSSL 使用PEM 文件格式存储证书和密钥。这种格式被设计用来安全的包含在ascii甚至富文本文档中，如电子邮件。这意味着您可以简单的复制和粘贴pem文件的内容到另一个文档中。

PEM文件是Base64编码的证书。PEM证书通常用于web服务器，因为他们可以通过一个简单的文本编辑器，很容易地转换成可读的数据。通常当一个PEM编码在文本编辑器中打开文件,它会包含不同的页眉和页脚。


```
openssl rsa -pubin -text -modulus -in warmup -in pubkey.pem
openssl rsa -pubin -text -modulus -in warmup -in pubkey.pem > 1.txt
```

```
    @kali:~/下载$ openssl rsa -pubin -text -modulus -in warmup -in pubkey.pem
RSA Public-Key: (256 bit)
Modulus:
    00:c2:63:6a:e5:c3:d8:e4:3f:fb:97:ab:09:02:8f:
    1a:ac:6c:0b:f6:cd:3d:70:eb:ca:28:1b:ff:e9:7f:
    be:30:dd
Exponent: 65537 (0x10001)
Modulus=C2636AE5C3D8E43FFB97AB09028F1AAC6C0BF6CD3D70EBCA281BFFE97FBE30DD
writing RSA key
-----BEGIN PUBLIC KEY-----
MDwwDQYJKoZIhvcNAQEBBQADKwAwKAIhAMJjauXD2OQ/+5erCQKPGqxsC/bNPXDr
yigb/+l/vjDdAgMBAAE=
-----END PUBLIC KEY-----
    @kali:~/下载$
```

```
RSA Public-Key: (256 bit)
Modulus:
    00:c2:63:6a:e5:c3:d8:e4:3f:fb:97:ab:09:02:8f:
    1a:ac:6c:0b:f6:cd:3d:70:eb:ca:28:1b:ff:e9:7f:
    be:30:dd
Exponent: 65537 (0x10001)
Modulus=C2636AE5C3D8E43FFB97AB09028F1AAC6C0BF6CD3D70EBCA281BFFE97FBE30DD
-----BEGIN PUBLIC KEY-----
MDwwDQYJKoZIhvcNAQEBBQADKwAwKAIhAMJjauXD2OQ/+5erCQKPGqxsC/bNPXDr
yigb/+l/vjDdAgMBAAE=
-----END PUBLIC KEY-----
```

modulus：N

exponent：e

16进制数转10进制

```
a = "C2636AE5C3D8E43FFB97AB09028F1AAC6C0BF6CD3D70EBCA281BFFE97FBE30DD"
print(int(a,16))
```

> 87924348264132406875276140514499937145050893665602592992418171647042491658461

大数分解出质因数，得到p、q

http://www.factordb.com/

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

87924348264132406875276140514499937145050893665602592992418171647042491658461 Factorize! (?)

Result:

status (?)	digits	number
FF	77 (show)	8792434826...61<77> = 275127860351348928173285174381581152299<39> · 319576316814478949870590164193048041239<39>

More information

ECM

factordb.com - 14 queries to generate this page (0.01 seconds) (limits) (Imprint) (Privacy Policy)

p=275127860351348928173285174381581152299

q=319576316814478949870590164193048041239

e=65537

@@@用rsatool

https://github.com/ius/rsatool

```
Failed to build gmpy
Installing collected packages: gmpy
    Running setup.py install for gmpy ... done
Successfully installed gmpy-1.17
haha@kali:~/下载/rsatool$ python rsatool.py -f PEM -o private.pem -p 27512786035
1348928173285174381581152299 -q 319576316814478949870590164193048041239 -e 65537
Traceback (most recent call last):
  File "rsatool.py", line 11, in <module>
    from pyasn1.codec.der import encoder
ImportError: No module named pyasn1.codec.der
haha@kali:~/下载/rsatool$ pip install pyasn1
```

```
/usr/share/python-wheels/pkg_resources-0.0.0-py3-none-any.whl/pkg_resources/py2_
warn.py:21: UserWarning: Setuptools will stop working on Python 2
************************************************************
You are running Setuptools on Python 2, which is no longer
supported and
>>> SETUPTOOLS WILL STOP WORKING <<<
in a subsequent release (no sooner than 2020-04-20).
Please ensure you are installing
Setuptools using pip 9.x or later or pin to `setuptools<45`
in your environment.
If you have done those things and are still encountering
this message, please follow up at
https://bit.ly/setuptools-py2-warning.
************************************************************
WARNING: pip is being invoked by an old script wrapper. This will fail in a futu
re version of pip.
Please see https://github.com/pypa/pip/issues/5599 for advice on fixing the unde
rlying issue.
To avoid this problem you can invoke Python with '-m pip' instead of running pip
```


haha@kali: ~/下载/rsatool

```
haha@kali:~/下载/rsatool$ python rsatool.py -f PEM -o private.pem -p 27512786035
13489281732851743815811522 99 -q 319576316814478949870590164193048041239 -e 65537
Using (p, q) to initialise RSA instance

n =
c2636ae5c3d8e43ffb97ab09028f1aac6c0bf6cd3d70ebca281bffe97fbe30dd

e = 65537 (0x10001)

d =
1806799bd44ce649122b78b43060c786f8b77fb1593e0842da063ba0d8728bf1

p = 27512786035134892817328517438158115229 9 (0xcefbb2cf7e18a98ebedc36e3e7c3b02b)

q = 319576316814478949870590164193048041239 (0xf06c28e91c8922b9c236e23560c09717)

Saving PEM as private.pem
Traceback (most recent call last):
  File "rsatool.py", line 164, in <module>
    data = rsa.to_pem()
  File "rsatool.py", line 98, in to_pem
    return (PEM_TEMPLATE % base64.encodestring(self.to_der()).decode()).encode()
  File "rsatool.py", line 107, in to_der
    seq.setComponentByPosition(len(seq), Integer(x))
  File "/home/haha/.local/lib/python2.7/site-packages/pyasn1/type/univ.py", line
 2267, in __len__
    return len(self._componentValues)
  File "/home/haha/.local/lib/python2.7/site-packages/pyasn1/type/base.py", line
 214, in plug
    raise error.PyAsn1Error('Attempted "%s" operation on ASN.1 schema object' %
```

```
C:\Users\Nah\Desktop>python2 rsatool.py -o private.pem -e 65537 -p 275127860351348928173285174381581152299 -q 31
9576316814478949870590164193048041239
Using (p, q) to initialise RSA instance

n =
c2636ae5c3d8e43ffb97ab09028f1aac6c0bf6cd3d70ebca281bffe97fbe30dd

e = 65537 (0x10001)

d =
1806799bd44ce649122b78b43060c786f8b77fb1593e0842da063ba0d8728bf1

p = 275127860351348928173285174381581152299 (0xcefbb2cf7e18a98ebedc36e3e7c3b02b)

q = 319576316814478949870590164193048041239 (0xf06c28e91c8922b9c236e23560c09717)

Saving PEM as private.pem
Traceback (most recent call last):
  File "rsatool.py", line 154, in <module>
    data = rsa.to_pem()
  File "rsatool.py", line 88, in to_pem
    return (PEM_TEMPLATE % base64.encodestring(self.to_der()).decode()).encode()
  File "rsatool.py", line 97, in to_der
    seq.setComponentByPosition(len(seq), Integer(x))
  File "C:\Python27\lib\site-packages\pyasn1\type\univ.py", line 2267, in __len__
    return len(self._componentValues)
  File "C:\Python27\lib\site-packages\pyasn1\type\base.py", line 214, in plug
    raise error.PyAsn1Error('Attempted "%s" operation on ASN.1 schema object' % name)
pyasn1.error.PyAsn1Error: Attempted "__len__" operation on ASN.1 schema object
```

kali，windows双失败

正常操作，生成private.pem文件

```
python rsatool.py -f PEM -o private.pem -p 275127860351348928173285174381581152299 -q 319576316814478949870590164193048041239 -e 65537
```

用openssl用private.pem解密flag.enc文件并将明文生成txt文件

```
rsautl -decrypt -in flag.enc -inkey private.pem -out flag.txt
```

@@@用RsaCtfTool
https://github.com/Ganapati/RsaCtfTool
https://github.com/3summer/CTF-RSA-tool

```
haha@kali:~/下载/RsaCtfTool-master$ python RsaCtfTool.py --publickey pubkey.pem
--uncipherfile flag.enc
Traceback (most recent call last):
  File "RsaCtfTool.py", line 21, in <module>
    from lib.rsa_attack import RSAAttack
  File "/home/haha/下载/RsaCtfTool-master/lib/rsa_attack.py", line 7, in <module
>
    from lib.keys_wrapper import PublicKey
  File "/home/haha/下载/RsaCtfTool-master/lib/keys_wrapper.py", line 11, in <mod
ule>
    from cryptography.hazmat.primitives import serialization
ImportError: No module named cryptography.hazmat.primitives
haha@kali:~/下载/RsaCtfTool-master$ pip install cryptography
/usr/share/python-wheels/pkg_resources-0.0.0-py3-none-any.whl/pkg_resources/py2_
warn.py:21: UserWarning: Setuptools will stop working on Python 2
***********************************************************
You are running Setuptools on Python 2, which is no longer
supported and
>>> SETUPTOOLS WILL STOP WORKING <<<
in a subsequent release (no sooner than 2020-04-20).
Please ensure you are installing
Setuptools using pip 9.x or later or pin to `setuptools<45`
in your environment.
If you have done those things and are still encountering
this message, please follow up at
https://bit.ly/setuptools-py2-warning.
***********************************************************
WARNING: pip is being invoked by an old script wrapper. This will fail in a futu
re version of pip.
Please see https://github.com/pypa/pip/issues/5599 for advice on fixing the unde
```

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

/home/haha/.local/lib/python2.7/site-packages/cryptography/__init__.py:39: CryptographyDeprecationWarning: Pytho
n 2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, and will b
e removed in a future release.
  CryptographyDeprecationWarning,

[*] Testing key pubkey.pem.
Traceback (most recent call last):
  File "RsaCtfTool.py", line 261, in <module>
    attackobj.attack_single_key(publickey, attacks_list)
  File "/home/haha/下载/RsaCtfTool-master/lib/rsa_attack.py", line 174, in attack_single_key
    self.load_attacks(attacks_list)
  File "/home/haha/下载/RsaCtfTool-master/lib/rsa_attack.py", line 123, in load_attacks
    except ModuleNotFoundError:
NameError: global name 'ModuleNotFoundError' is not defined
```

那个模块不支持py2

```
hahaa@kali:~/下载/RsaCtfTool-master$ python3 RsaCtfTool.py --publickey pubkey.pem
 --uncipherfile flag.enc

[*] Testing key pubkey.pem.
Can't load smallfraction because sage is not installed
Can't load ecm2 because sage is not installed
Can't load qicheng because sage is not installed
Can't load ecm because sage is not installed
Can't load boneh_durfee because sage is not installed
[*] Performing comfact_cn attack on pubkey.pem.
[*] Performing factordb attack on pubkey.pem.

Results for pubkey.pem:

Unciphered data :
b'\x00\x02\xc0\xfe\x04\xe3&\x0e[\x87\x00PCTF{256b_i5_m3dium}\n'
```



```
用法一：已知公钥(自动求私钥)
python3 RsaCtfTool.py --publickey pubkey.pem --uncipherfile flag.enc
python RsaCtfTool.py --publickey 公钥文件 --uncipherfile 加密的文件

用法二：已知公钥求私钥。
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
把n,e转换为PEM格式
python RsaCtfTool.py --createpub -n 782837482376192871287312987398172312837182 -e 65537
```

```
Results for pubkey. pem:
Unciphered data:
b'\x00\x02\ xc0\ xfe\x04\ xe38\x0e[\x87\x00PCTF{256b_i5_m3dium}\n'
```

> PCTF{256b_i5_m3dium}

或者直接使用python脚本

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_v1_5
from gmpy2 import invert
with open('flag.enc','r') as f:
    cip=f. read()
n = 879243482641324068752761405144999371450508936656025929924181716470424916584611
e = 65537L
p = 27512786035134892817328517438158115229 9L
q = 319576316814478949870590164193048041239L
phi=(p-1)*(q-1)
d=invert(e, phi)
privkey=RSA.construct((n,e,long(d),p,q))
key1=PKCS1_v1_5. new(privkey)
print key1.decrypt(cip,'')[:-1]
```

py × test_1.py × test_2.py × test_3.py ×

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_v1_5
from gmpy2 import invert
with open('flag.enc','r') as f:
    cip=f. read()
n = 879243482641324068752761405144999371450508936656025929924181 7
e = 65537L
p = 27512786035134892817328517438158115229 9L
q = 31957631681447894987059016419304804123 9L
phi=(p-1)*(q-1)
d=invert(e, phi)
privkey=RSA.construct((n,e,long(d),p,q))
key1=PKCS1_v1_5. new(privkey)
print key1.decrypt(cip,'')[:-1]
```

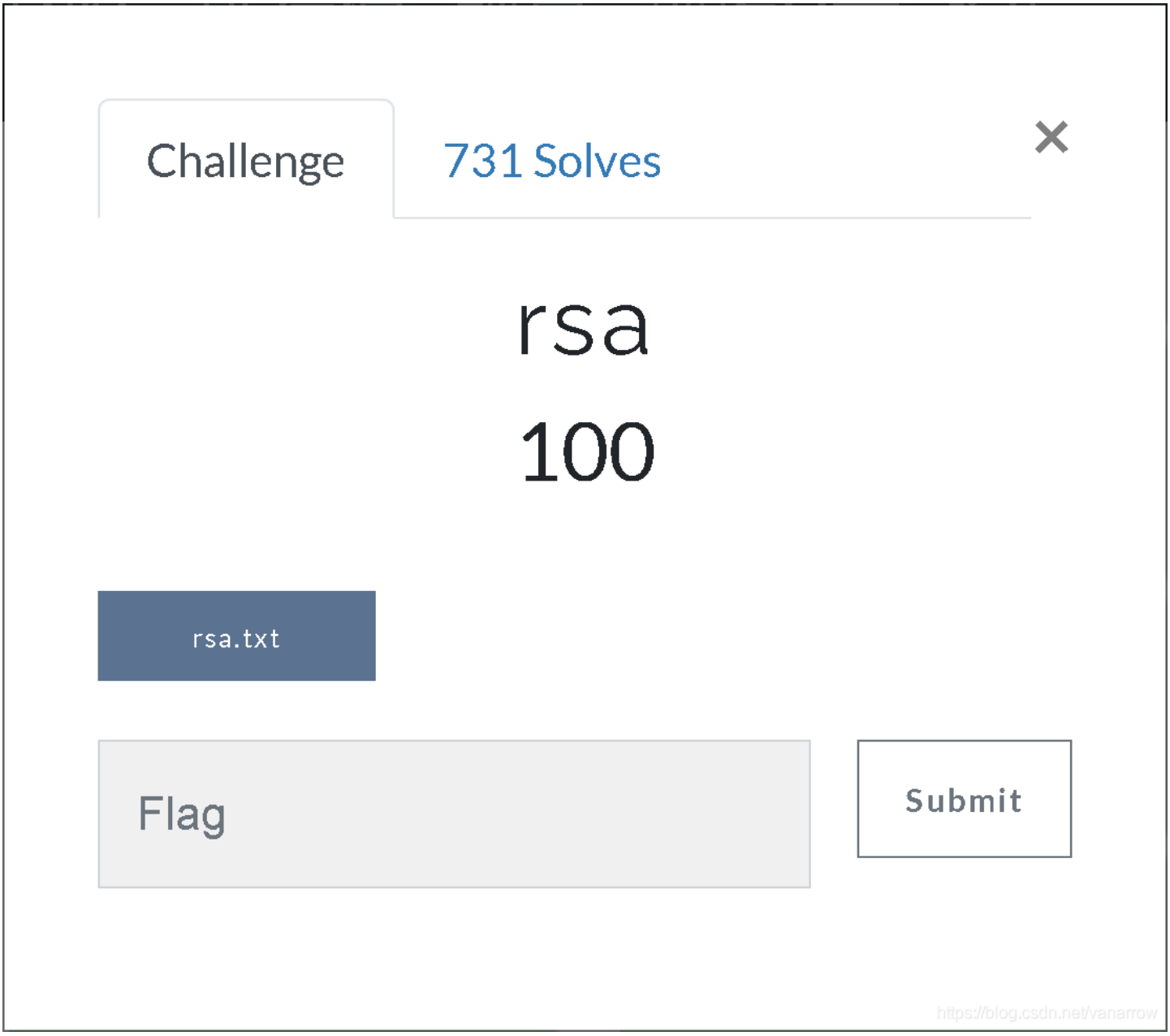
test_3 ×

```
C:\Python27\python2.exe C:/Users/Nah/Desktop/test_3.py
PCTF{256b_i5_m3dium}

Process finished with exit code 0
```



```
import math
import sys
from Crypto.PublicKey import RSA

keypair = RSA.generate(1024)
keypair.p = 27512786035134892817328517438158115229 9
keypair.q = 31957631681447894987059016419304804123 9
keypair.e = 65537
keypair.n = keypair.p*keypair.q
Qn=((keypair.p-1)*(keypair.q-1))
i = 1
while(True):
    x = (Qn*i)+1
    if(x % keypair.e == 0):
        keypair.d = x / keypair.e
        break
    i += 1
private = open('private.pem', 'w')
private.write(keypair.exportKey())
private.close()
```

5. Bugku

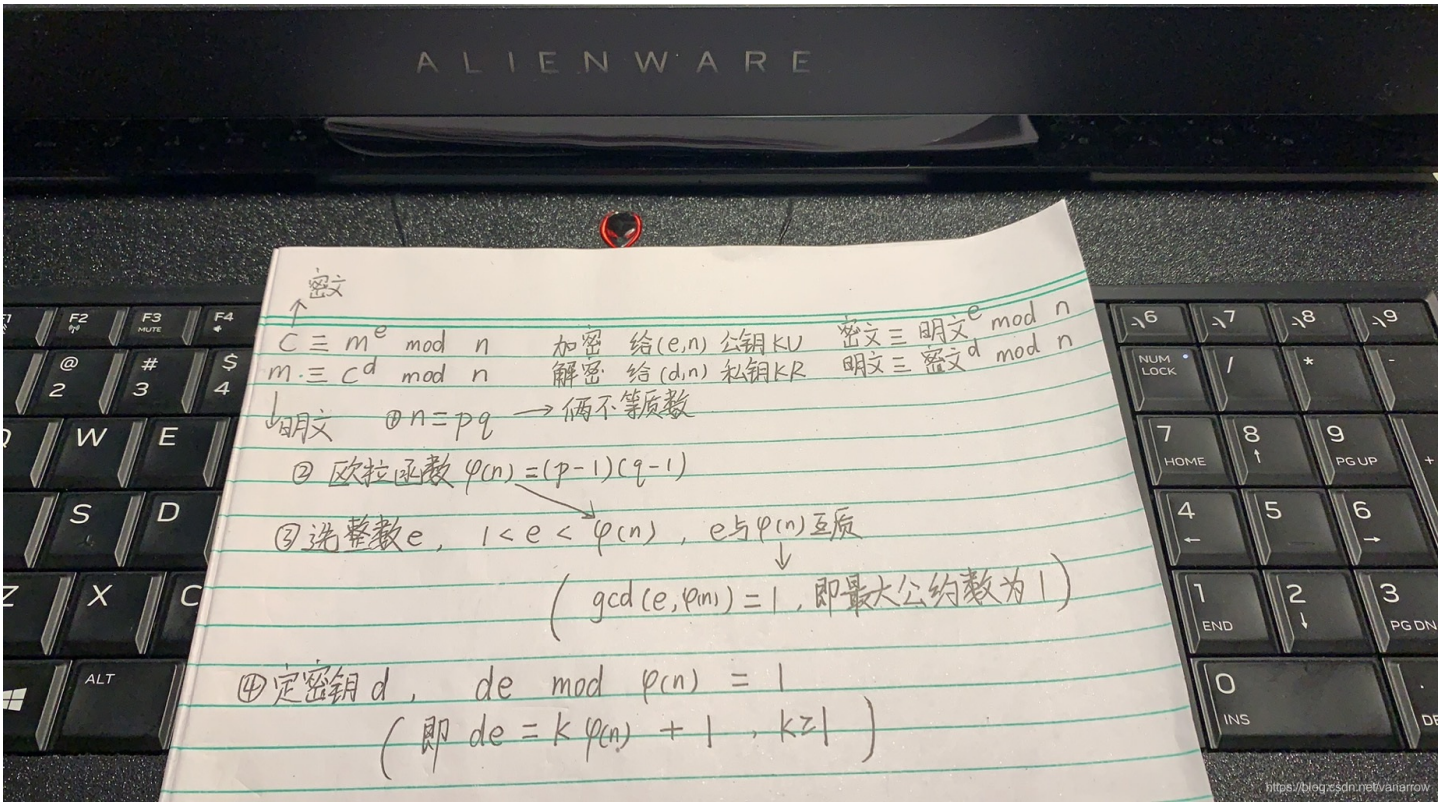


rsa.txt

https://ctf.bugku.com/files/98e8f374f63ee3ef4818621ceafcb78f/rsa.txt

```
N : 46065781388428960989637205658554417248531811702624626389974432923749270182062721955600778820059011913617389598900138215153600685382332638289236314360431451868638878600298924880081486124859507532627709964533869497709745916853089877600729369572810197606942397169652423775522718706141820284991147912479399072259 7
e : 35461110244130757205657218182792589919834535022875373093108939327546391654445662689424541509610783446577840953237318712531855461472259930179152891621283936812106603554100880826153450058602365276771227162578520428096468800468032830012484968047710530251937737009257810782711682139182621097232037761496754782761 9

enc : 38230991316229399651823567590692301060044620412191737764632384680546256228451518238842965221394711848337832459443844468894683621541882148407367446578858589438101776758719911114666531582571911396056999163473082949956645302808168504827405306022545591237591211063383592202426377759190269335633260694494243911 92
```

给了n，e，c（c就是密文，就是给的enc，你可能会猜这是什么变量，encode加密编码，decode解密解码），求明文m



对着之前的图想想，n分离出 p、q

可以用Wiener's attack脚本分解质数

```
def continued_fractions_expansion(numerator,denominator):#(e,N)
    result=[]

    divident=numerator%denominator
    quotient=numerator/denominator
    result.append(quotient)

    while divident!=0:
        numerator=numerator-quotient*denominator

        tmp=denominator
        denominator=numerator
        numerator=tmp

        divident=numerator%denominator
        quotient=numerator/denominator
        result.append(quotient)

    return result

def convergents(expansion):
    convergents=[(expansion[0],1)]
    for i in range(1,len(expansion)):
        numerator=1
        denominator=expansion[i]
        for j in range(i-1,-1,-1):
            numerator+=expansion[j]*denominator
            if j==0:
                break
            tmp=denominator
            denominator=numerator
```

```
            denominator=numerator
            numerator=tmp
        convergents.append((numerator,denominator))#(k,d)
    return convergents

def newtonSqrt(n):
    approx = n/2
    better = (approx + n/approx)/2
    while better != approx:
        approx = better
        better = (approx + n/approx)/2
    return approx

def wiener_attack(cons,e,N):
    for cs in cons:
        k,d=cs
        if k==0:
            continue
        phi_N=(e*d-1)/k
        #x**2-((N-phi_N)+1)*x+N=0
        a=1
        b=-((N-phi_N)+1)
        c=N
        delta = b*b - 4*a*c
        if delta<=0:
            continue
        x1= (newtonSqrt(delta)-b)/(2*a)
        x2=-(newtonSqrt(delta)+b)/(2*a)
        if x1*x2==N:
            return [x1,x2,k,d]


N=46065781388428960989637205658554417248531811702624626389974432923749270182062721955600778820059011913617389598
90013821515360068538233263828923631436043145186863887860029892488008148612485950753262770996453386949770974591685308987760072936957281019760694239716965242377552271870614182028499114791247939907225 97
e=35461110244130757205657218182792589919834535022875373093108939327546391654445662689424541509610783446577840953237318712531855461472259930179152891621283936812106603554100880826153450058602365276771227162578520428096468800468032830012484968047710530251937737009257810782711682139182621097232037761496754782761 9

expansion=continued_fractions_expansion(e,N)
cons=convergents(expansion)

p,q,k,d=wiener_attack(cons,e,N)
print 'p\n', p
print 'q\n', q
```

```
p
28805791771260259486856902729020438686670354441296247148207862836064657849735343618207098163901787287368569768472521344635567334299356760080507454640207003
q
15991846970993213322072626901560749932686325766403404864023341810735319249066370916090640926219079368845510444031400322229147771682961132420481897362843199
```

求出了p、q，那就能求出来 （p-1)(q-1) = ϕ

已知了n、e、c、p、q、ϕ、求m

```
import binascii
import sys
sys.setrecursionlimit(1000000)
def ByteToHex(bins):
    return ''.join(["%02X" % x for x in bins]).strip()
def n2s(num):
    t = hex(num)[2:-1]  # python
    if len(t) % 2 == 1:
        t = '0' + t
    #print(t)
    return(binascii.a2b_hex(t).decode('latin1'))
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:
        print('modular inverse does not exist')
        return 'null'
    else:
        return x % m
c = 38230991316229399651823567590692301060044620412191737764632384680546256228451518238842965221394711848337832459443844446889468362154188214840736744657885858943810177675871991111466653158257191139605699916347308294995664530280816850482740530602254559123759121106338359220242637775919026933563326069449424391192
p = 28805791771260259486856902729020438686670354441296247148207862836064657849735343618207098163901787287368569768472521344635567334299356760080507454640207003
q = 15991846970993213322072626901560749932686325766403404864023341810735319249066370916090640926219079368845510444031400322229147771682961132420481897362843199
e = 35461110244130757205657218182792589919834535022875373093108939327546391654445662689424541509610783446577840953237318712531855461472259930179152891621283936812106603554100880826153450058602365276771227162578520428096468800468032830012484968047710530251937737009257810782711682139182621097232037761496754782761 9
n = p * q
d = modinv(e, (p - 1) * (q - 1))
m = pow(c, d, n)
print 'm \n', m
```

flag{Wien3r_4tt@ck_1s_3AsY}

> 本文为个人学习笔记，仅供学习使用，由于本人基础知识薄弱，欢迎各位大佬批评指正！！！