**RSA编程实验报告**

20373819-赵睿智

**第一部分**

**实现RSA的密钥生成，加解密算法**

要求1：实现对数字的加解密，要求p,q的长度不小于1024bit。

要求2：用中国剩余定理优化以上RSA的解密速度。

要求3：对消息进行数字编码进行加解密，对生成公钥私钥的文件实现进行本地保存和读取（参考pem文件）

**实验准备部分**

**解决方案：**在看到这个问题时，我首先想到的是由于加密解密的运算基于大长整数的加减，乘除以及模幂操作，因此利用已有的知识，需要模拟大长整数的算法，因此写了如下的代码：

Tips：此部分仅仅是笔者在开始做实验时的一些想法，这部分的代码并没有实现什么，仅在指数<100000时进行的模幂等相关运算有较高的效率，对1024bit的大长整数的效率感人(也许20天也算不出来)。故可以略过…

1.结构体，宏定义：

#include<stdio.h>

#include<string.h>

#include<ctype.h>

#include<stdbool.h>

#include<stdlib.h>

#include<math.h>

#define LEN 1000000

typedef struct st {

char num[LEN];

}L, \*ptr;

typedef struct st1 {

int Int[LEN];

int len;

}I, Iptr;

2.加法：

L add(L a, L b)

{

I A, B, C;

A = rev(a);

B = rev(b);

memset(&C, 0, sizeof(C));

int x = 0, i;

for (i = 0; i <= (A.len > B.len ? A.len : B.len); i++)

{

C.Int[i] = A.Int[i] + B.Int[i] + x;

x = C.Int[i] / 10;

C.Int[i] %= 10;

}

C.len = (C.Int[i] == 0 ? i : i + 1);

L ans = rec(C);

return ans;

}

3.乘法：

L mul(L a, L b)

{

I A, B, C;

int i, j;

A = rev(a);

B = rev(b);

memset(&C, 0, sizeof(C));

for (i = 0; i < A.len + B.len; ++i)

{

// 这里直接计算结果中的从低到高第 i 位，且一并处理了进位

// 第 i 次循环为 c[i] 加上了所有满足 p + q = i 的 a[p] 与 b[q] 的乘积之和

// 这样做的效果和直接进行上图的运算最后求和是一样的，只是更加简短的一种实现方式

for (j = 0; j <= i; ++j)

C.Int[i] += A.Int[j] \* B.Int[i - j];

if (C.Int[i] >= 10)

{

C.Int[i + 1] += C.Int[i] / 10;

C.Int[i] %= 10;

}

}

for (i = A.len + B.len; C.Int[i] == 0; i--);

C.len = i + 1;

L ans = rec(C);

return ans;

}

4.除法：

// 被除数 a 以下标 last\_dg 为最低位，是否可以再减去除数 b 而保持非负

bool greater\_eq(I A, I B, int last\_dg)

{

// 有可能被除数剩余的部分比除数长，这个情况下最多多出 1 位，故如此判断即可

if (A.Int[last\_dg + B.len] != 0)

return true;

// 从高位到低位，逐位比较

for (int i = B.len - 1; i >= 0; --i)

{

if (A.Int[last\_dg + i] > B.Int[i])

return true;

if (A.Int[last\_dg + i] < B.Int[i])

return false;

}

// 相等的情形下也是可行的

return true;

}

L mod(L a, L b)

{

I A, B, C, D;

int i = 0, j = 0;

A = rev(a);

B = rev(b);

memset(&C, 0, sizeof(C));

memset(&D, 0, sizeof(D));

// c 是商

// d 是被除数的剩余部分，算法结束后自然成为余数

for (i = 0; i < A.len; ++i)

D.Int[i] = A.Int[i];

for (i = A.len - B.len; i >= 0; --i)

{

// 计算商的第 i 位

while (greater\_eq(D, B, i))

{

// 若可以减，则减

// 这一段是一个高精度减法

for (j = 0; j < B.len; ++j)

{

D.Int[i + j] -= B.Int[j];

if (D.Int[i + j] < 0)

{

D.Int[i + j + 1] -= 1;

D.Int[i + j] += 10;

}

}

// 使商的这一位增加 1

C.Int[i] += 1;

// 返回循环开头，重新检查

}

}

for (i = B.len - 1; D.Int[i] == 0; i--);

D.len = i + 1;

L ans = rec(D);

return ans;

}

5.求最大公约数：

//求最大公约数

L gcd(L a, L b)

{

L temp = b;

while (mod(a, b).num[0] != '\0')

{

temp = mod(a, b);

a = b;

b = temp;

}

return temp;

}

6.快速幂算法：

L Fp(L b, L p, L m)

{

I B, P;

P = revv(p);

B = rev(b);

int x, cnt, loopcnt = 0;

L temp, ans;

ans.num[0] = '1';

ans.num[1] = '\0';

temp = mod(b, m);

while (!isP0(P))

{

x = 0;

for (cnt = 0; cnt < P.len; cnt++)

{

P.Int[cnt] += x \* 10;

x = P.Int[cnt] % 2;

P.Int[cnt] /= 2;

}

//a为商，x为余数

if (loopcnt != 0)

temp = mod(mul(temp, temp), m);

if (x == 1)

ans = mod(mul(ans, temp), m);

loopcnt++;

}

return ans;

}

bool isP0(I P)

{

int cnt = 0;

bool judge = true;

for (cnt = 0; cnt < P.len; cnt++)

if (P.Int[cnt] != 0)

judge = false;

return judge;

}

但是在运行后，发现由于算法的缺陷，尤其是模拟减法采用的带余除法，速度很慢，于是不得已借用了mingw中自带的gmp大数库，并实现了以上的要求。见下面的实验改进部分。

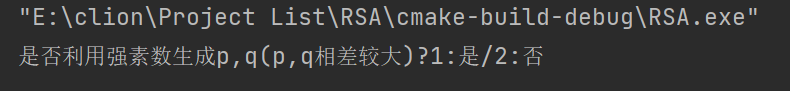
**实验改进部分(正式部分)**

一、代码部分：由于在word上查看代码并不方便，请助教查看附件中的RSA.c的源文件，里面有很详细的注释及代码解释。

二、代码校验部分：（使用的是clion编程软件）

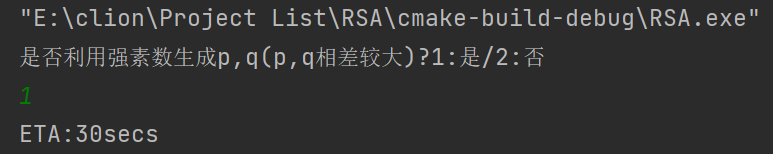
为方便助教检查代码，给程序增加了校验模式，具体的操作过程如下：

**1.**进入程序后，显示如下界面：

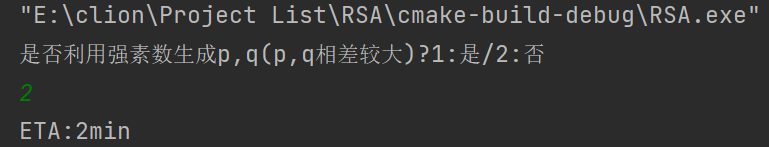


**2.** 选择模式1后，出现如下界面：

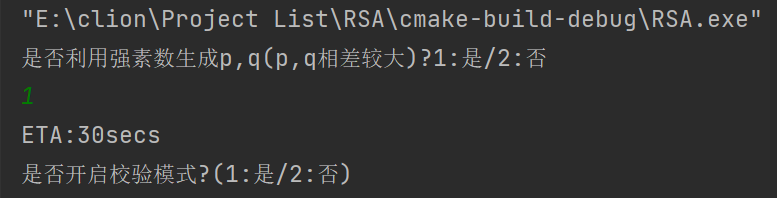
(ET是多次程序运行的平均时间，请助教耐心等待)



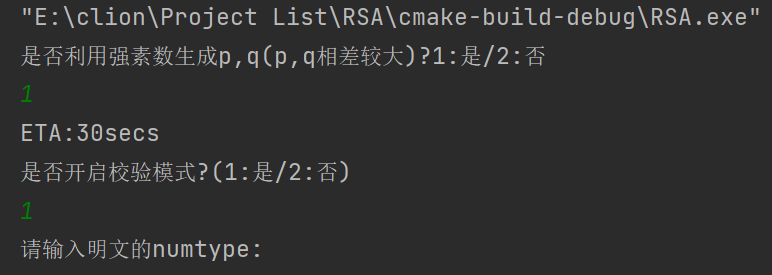
若选择模式2，出现如下界面：



**3.**待素数p,q生成后，程序会提示是否开启校验模式。由于采用的是一次一密的加密方式，建议开启校验模式。



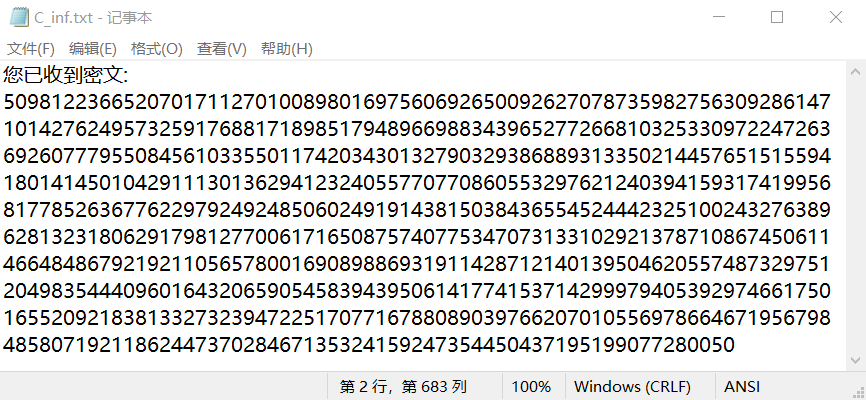
**4.**开启校验模式后，程序提示输入明文的numtype，这时输入要加密的明文即可。

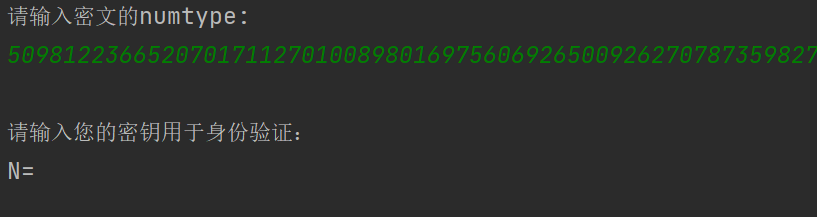


**5.**输入后，程序会输出公钥的相关信息。由于采用的是一次一密，选择校验模式后直接进入解密模式。

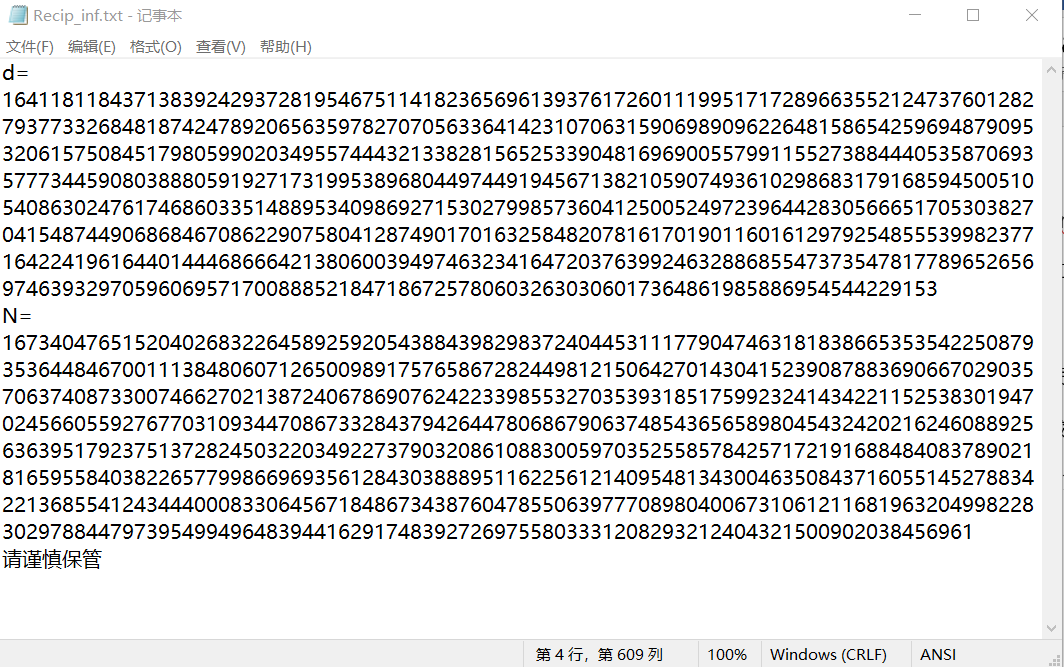


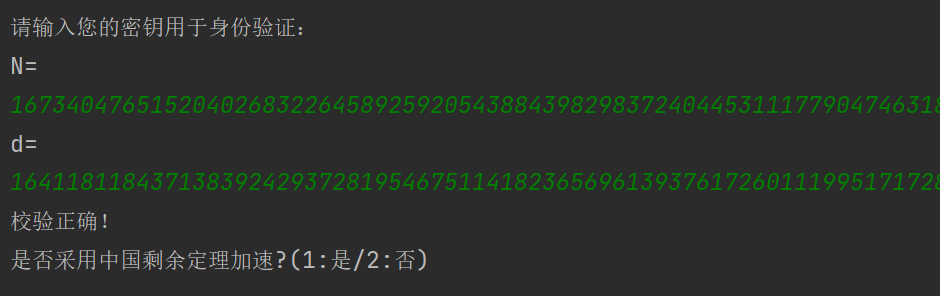
**6.**接下来程序会提示输入密文。在clion的对应工程文件夹的cmake-build-debug文件夹里找到C\_inf.txt文件，文件内是对应的密文，将其拷贝并输入。（可与之前在控制台上输出的密文做比对），输入后，界面如下：



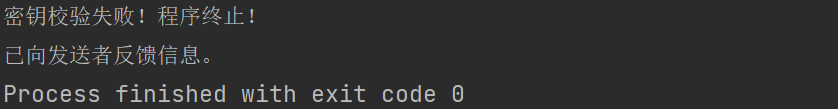


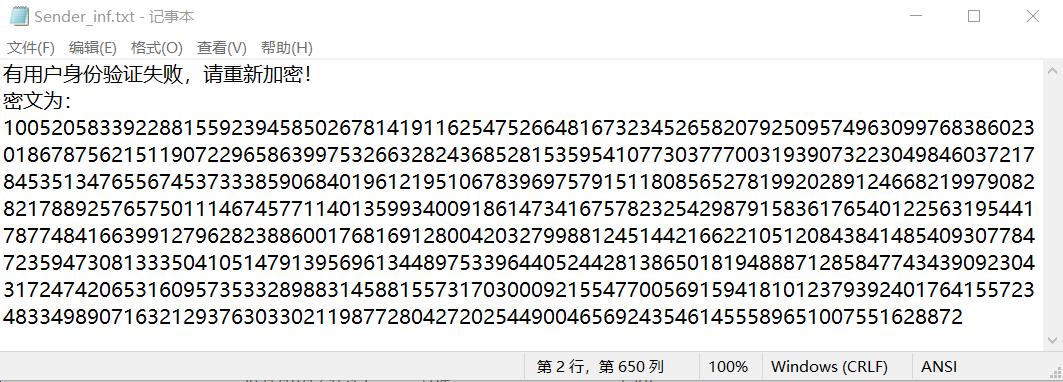
**7.**接下来，系统会提示需要私钥对(N,d)已验证解密者的身份。这时，仍从clion的对应工程文件夹的cmake-build-debug文件夹里找到Recip\_inf.txt文件，文件内是对应的私钥对，输入后便可验证身份正确：



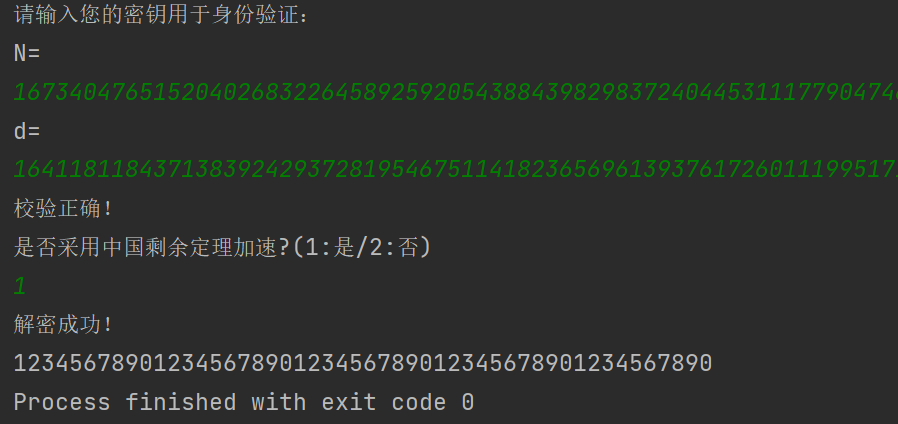


如若身份验证失败，程序会在clion的工程文件夹RSA的cmake-build-debug文件夹里的Sender\_inf.txt文件里找到如下反馈给加密者的信息：

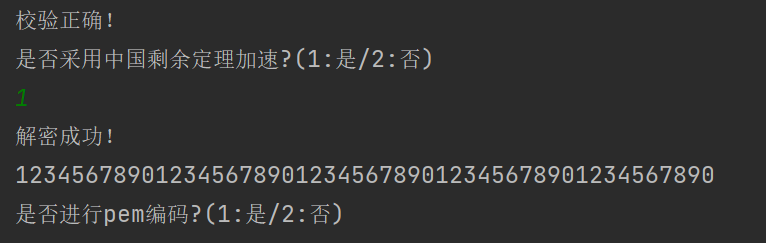




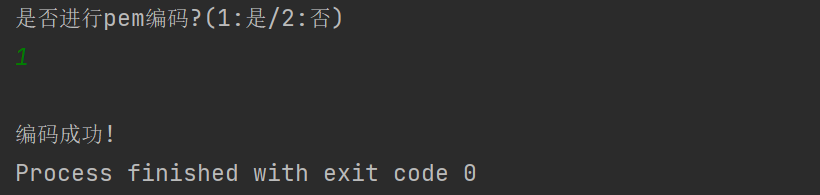
**8.**接下来程序会提示是否采用中国剩余定理加速。若使用会明显加快其解密速度，将解密文与开始时的明文比较即可校验程序的正确性。如下：（不采用中国剩余定理加速亦可得到正解）



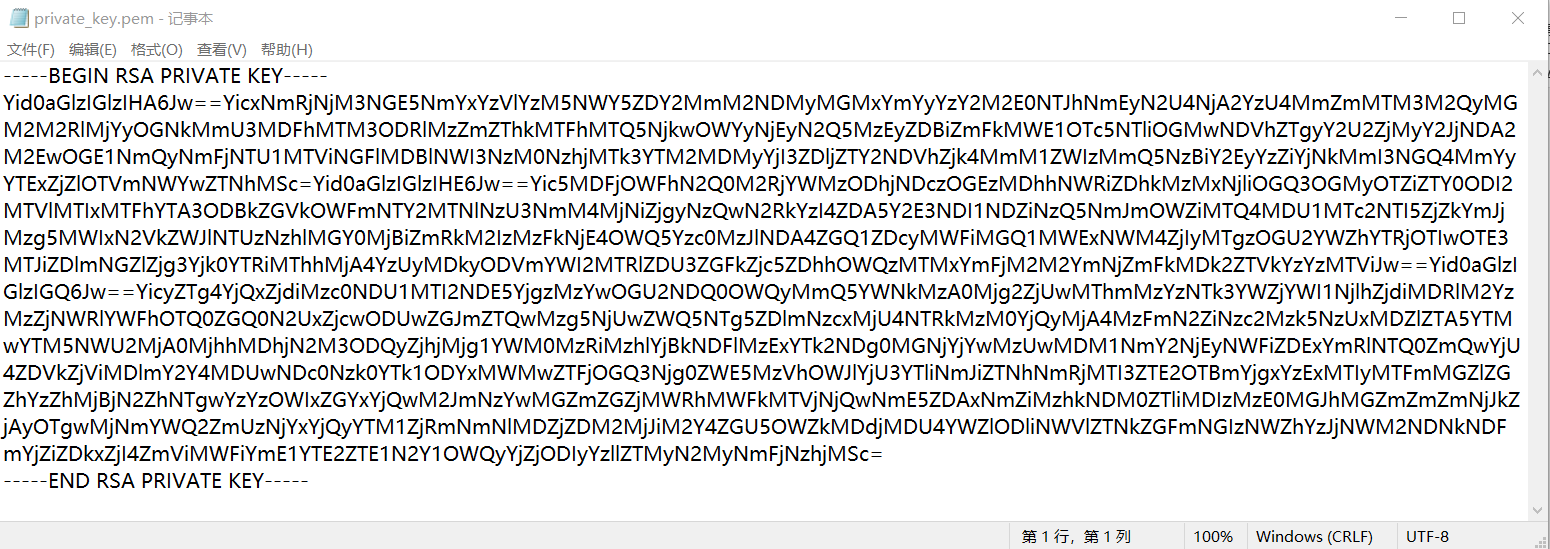
**9.**接下来程序会提示是否对密钥编码为pem格式并保存到本地(笔者水平有限，无法将hex用c转化为utf-8,因而pem的编码方式肯定存在问题，恳请助教指正并给出建议！！！)。

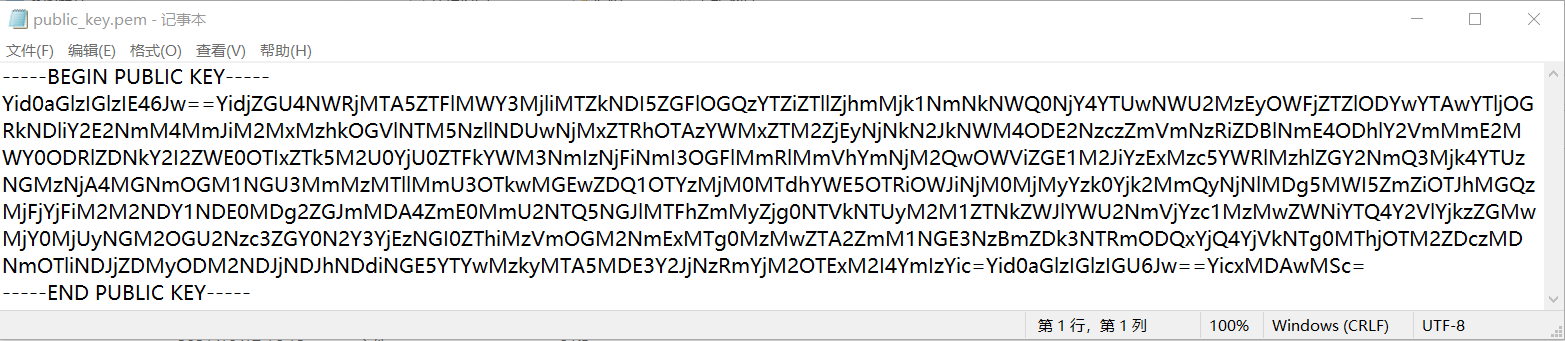


**10.**选择模式1后，程序提示编码成功并结束。



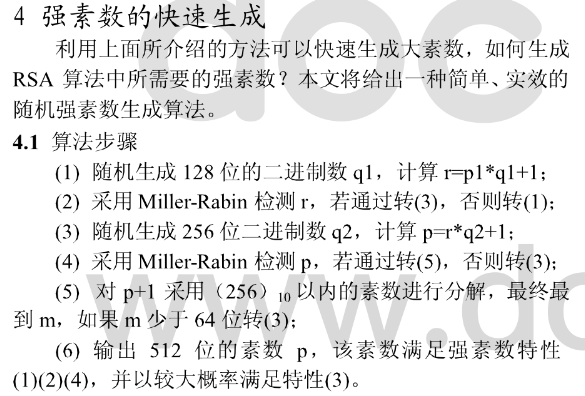
**11.**cmake-build-debug文件夹里的private\_key.pem和public\_key.pem里存放的便是编码后的私钥和公钥文件。

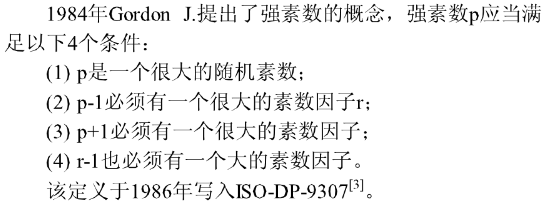




**实现密钥生成的过程中采用的技术：**

1. 一次一密。
2. p与q的随机化。在实现过程中，首先使用time.h库中的clock()函数，通过计算程序进程时间对随机种子grt实现随机化，之后利用此随机种子对p,q实现随机生成。
3. p,q采用快速生成强素数的算法





(《一种快速的强素数生成方法》游新娥，田华娟。发表于《通信技术2009年02期第42卷)

**密钥生成过程中涉及的安全细节：**

1.p,q采用了强素数。可防止黑客的暴力分解。

2.p,q相差较大，可防止黑客利用连分式理论对p，q进行破解。

3.p-1,q-1的因子很小。

3.生成的d很大，可防止黑客通过穷举法解出d进而直接破解密文。

4.在密钥生成过程中引入随机化，使得p，q的不可测性大大增强。

**第二部分**

**理解几种常见RSA攻击**

要求1：需要在实验报告中附上你的攻击原理和攻击代码

要求2：分析为何这种场景产生安全问题的本质原因。

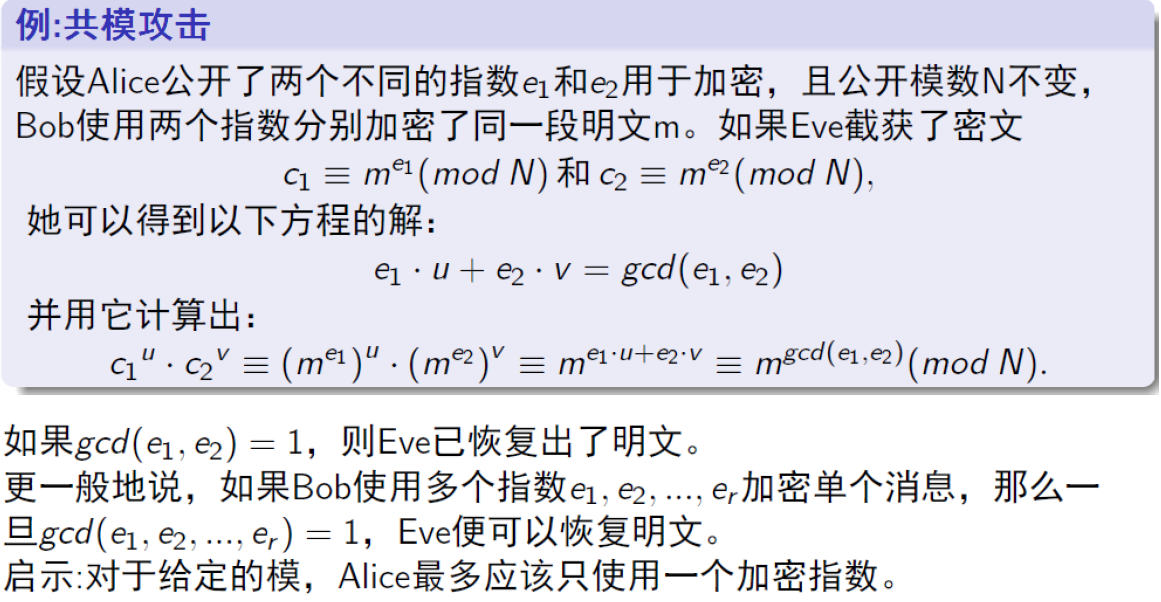
要求3：RSA-OAEP的思考题。

**Level1**

**1.分析：**

分析其加密方式可以发现：加密者对同一段明文利用相同的p,q以及不同的e进行了两次加密，这属于典型的共模攻击。通过gcd函数可求得(e1,e2)=(65537,963419)=1。由此可见，利用贝祖定理，直接对同一明文加密得到的两个密文进行相关模幂运算便可不用求p,q,d直接得到明文。

**2.具体算法如下：**



**3.这种安全问题出现的原因：**

引子：

假设有一家公司COMPANY，在员工通信系统中用RSA加密消息。COMPANY首先生成了两个大质数P,Q，取得PQ乘积N。并且以N为模数，生成多对不同的公钥及其相应的私钥。COMPANY将所有公钥公开。而不同的员工获得自己的私钥，比如，员工A获得了私钥d1.员工B获得了私钥d2.

现在，COMPANY将一条相同的消息，同时经过所有公钥加密，发送给所有员工。  
 此时便极有可能出现共模攻击。

可见，有时加密者为了方便，会用一个给定的模产生许多加密指数，但这样只要窃密者截获密文与其对应的加密指数，便极有可能直接破解出明文。故在实际情况中因严加防范这种情况的产生。

**4.攻击代码：**

int main(int argc, char \*\*argv)

{

Char e1[1024],e2[1024],c1[1024],c2[1024],n[1024];

gets(e1);

gets(e2);

mpz\_t re1,re2,E1,E2;

mpz\_init(re1);

mpz\_init(re2);

mpz\_init\_set\_str(E1,e1,10);

mpz\_init\_set\_str(E2,e2,10);

inversion(E1,E2,&re1,&re2);

gmp\_printf("%Zd\n%Zd\n",re1,re2);

gets(c1);

gets(c2);

gets(n);

mpz\_t N,c1FP,c2FP,ji,flag,C1,C2;

mpz\_init(c1FP); mpz\_init(c2FP);

mpz\_init(flag); mpz\_init(ji);

mpz\_init\_set\_str(C1,c1,10);

mpz\_init\_set\_str(C2,c2,10);

mpz\_init\_set\_str(N,n,10);

Fp(&C1,&re1,&N,&c1FP);

Fp(&C2,&re2,&N,&c2FP);

mpz\_mul(ji,c1FP,c2FP);

mpz\_mod(flag,ji,N);

gmp\_printf("%Zd",flag);

return 0;

}

(使用了自己写的Inversion函数，具体算法请看源代码注释)

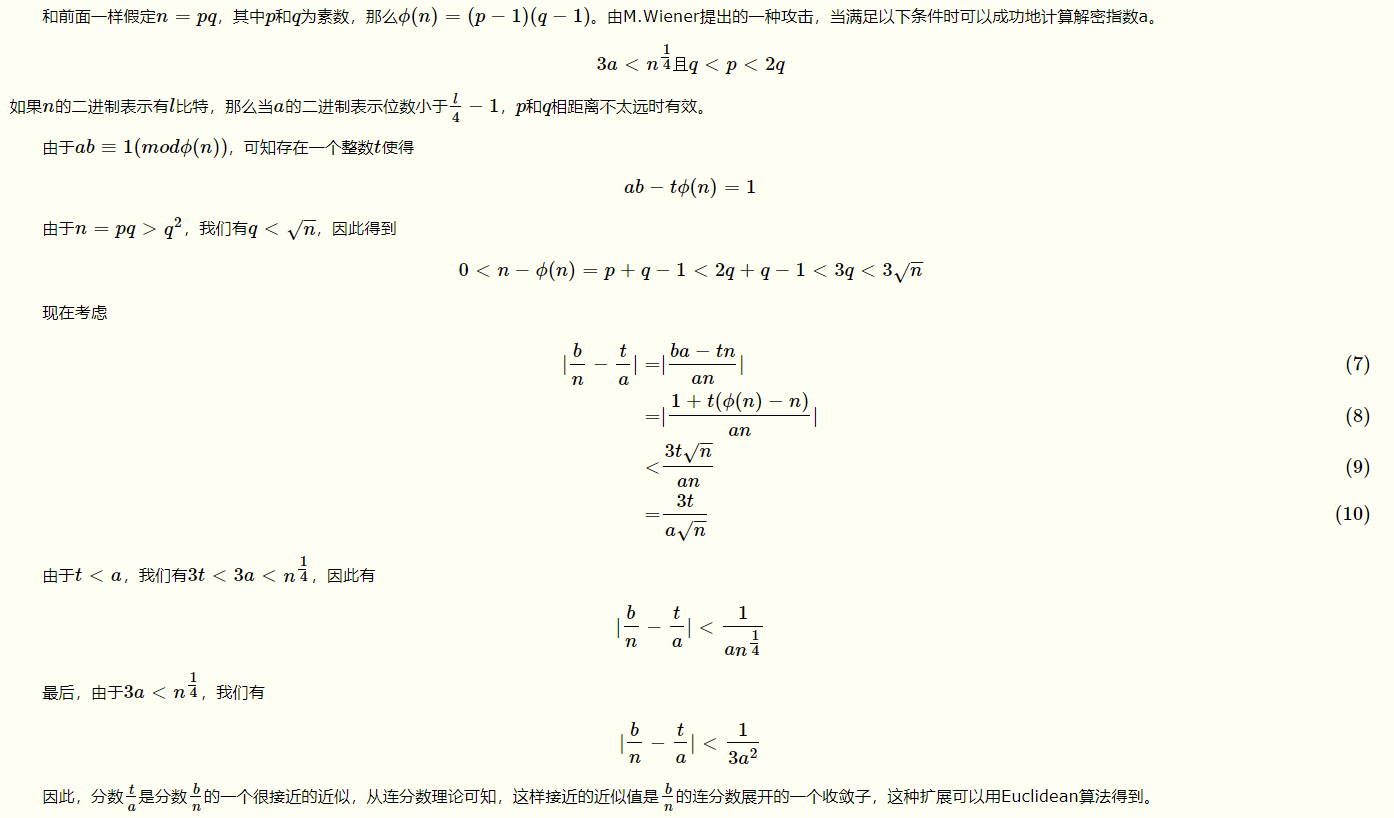
以下为python的攻击代码：

e1 = 65537  
e2 = 963419  
def getre(c,d,m):  
 for i in range(d):  
 c=c\*c%m  
 return c  
  
def ex(a,b):  
 if b==0:  
 return 1,0,a  
 else:  
 x,y,q =ex(b,a%b)  
 x,y=y,(x-a//b\*y)  
 return x,y,q  
  
def modre(a,p):  
 x,y,q=ex(a,p)  
 if q!=1:  
 print(q)  
 return;  
 else:  
 return (x+p)%p  
  
def isgcd(e1,e2):  
 x,y,q=ex(e1,e2)  
 if q==1:  
 print('(e1,e2)=1')  
 else:  
 print(q)  
  
isgcd(e1,e2)  
  
N=17163032619982233797602854042535897808957544664163502752959679407960994373854760206297968907142946865169636014343101514757592589055073100803372760587477491687355100752037881885917667944758145846074200498743386144000393288441891166709028576265088585148498627261895180687660278054727240581766147179779601969818590625606678354767031723251454205568955826817832805943418355294936666555120295531741185597534453186623769972269184001251391960342621123566974695978134851635513386415263633691232944879169781947123740362695584850142265761587364238120062371812091468926377123154635632871202767196558735750143916321833823881494333  
c1=9471942065493862990661170159845915489453625823279744308699548606361517384655044942222389975278380499191588181142475062350256438128181814686424656608346807458152248561602838756986128448773937958427549007291082363241712328183458825570161139954168559467752202744664465894248798665897449486058311202648314973301468980876237196207875676288087842330063200520625251478138851441267676696155600315002372663293949283468603327295586942366793998889614316324960698784056751393594714183913278378237920513974425198494098391157727720211475198457305694512618786562592987519575694942470380861353222454747848479957877378881098787145710  
c2=4407023295581609264353466112013445130210766867576283061345324656448555264891267246867056063614585134820309792854113650396508186942876572093445700588310238300450857147109085981373345202843806788446802172247573054744276852612531302622201063765188636014208356921464972869349970200957274709065979095570449508392358104603617989020435924270364788851607099947709123520072076672568314618562616668971004242718709082553060960639686254873198911281218628514456842578051940672084276890893794230523959924759177672811267709624274535017328785065805760252070838981312818186690835115614734476082254510700307341317055063797768869437211  
x,y,q=ex(e1,e2)  
m=pow(c1,x,N)\*pow(c2,y,N)%N  
print(m)  
flag=n2s(m)  
print(flag)

**Level2**

**1.分析：**

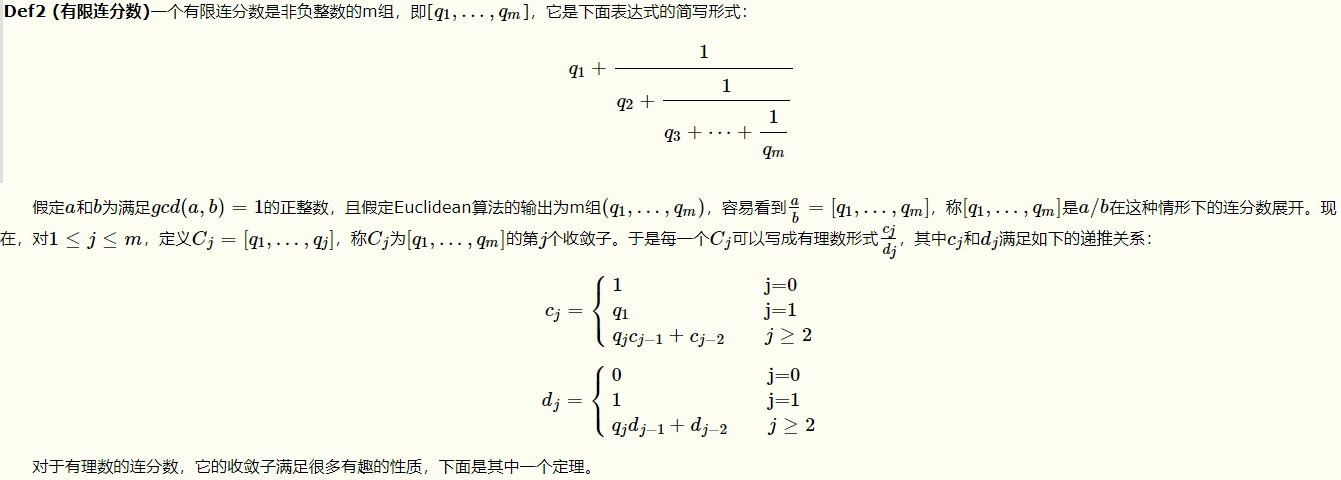
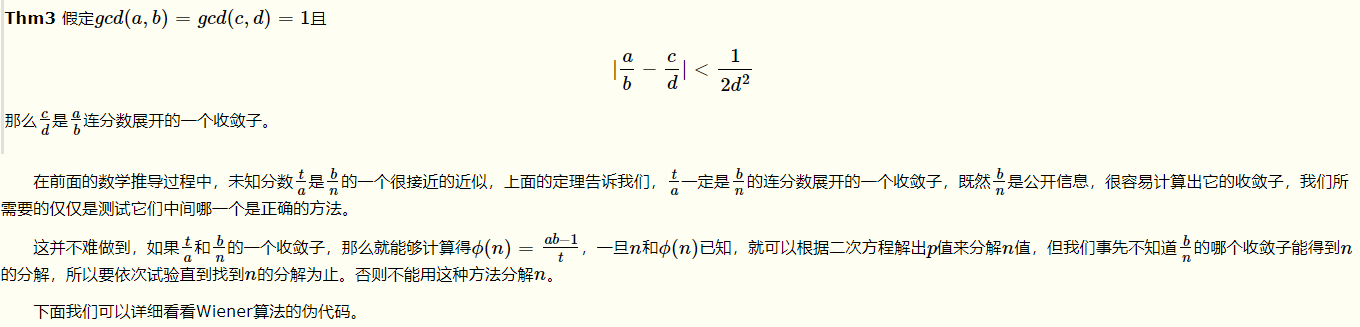
分析其加密方式可以发现：加密者先生成了一个介于0到N的四分之一次方的d，再利用此生成的e对明文加密。由于此d非常的小(大约为512位)，利用连分式理论便可破解。

**2.具体算法及思路如下：**

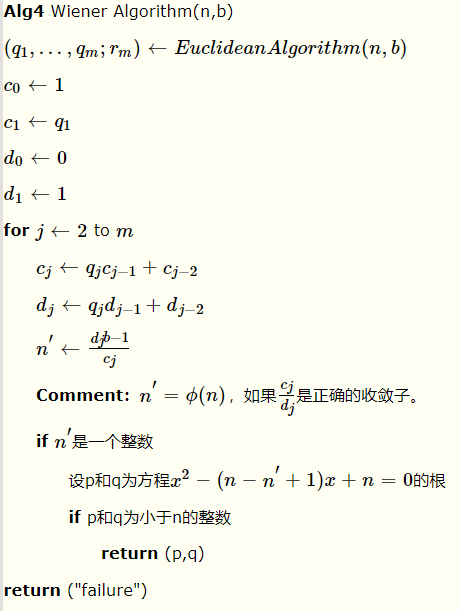
由上面的介绍可知，破解的关键在于

d = randint(0, int(iroot(N, 4)[0]) // 3)

这个关键的约束条件。通过利用此条件，我们便可得到两个分数十分相近的结论。下面介绍连分数的相关知识：

下面介绍这个定理：

通过如上的介绍，我们便可得到wiener算法的伪代码：

 便可由此写出攻击脚本。

**3.这种安全问题出现的原因：**

这种加密方式可以很明显的看到e很大，可能是考虑到解密者的便捷与高效性先设置了这个较小的d，(其实e和d在这里是地位对称的)，从而使得这个e变得非常大。但是这与低加密指数相同，低解密指数可以加快解密的过程，但是也带来了很大的安全问题。

**4.攻击代码：**

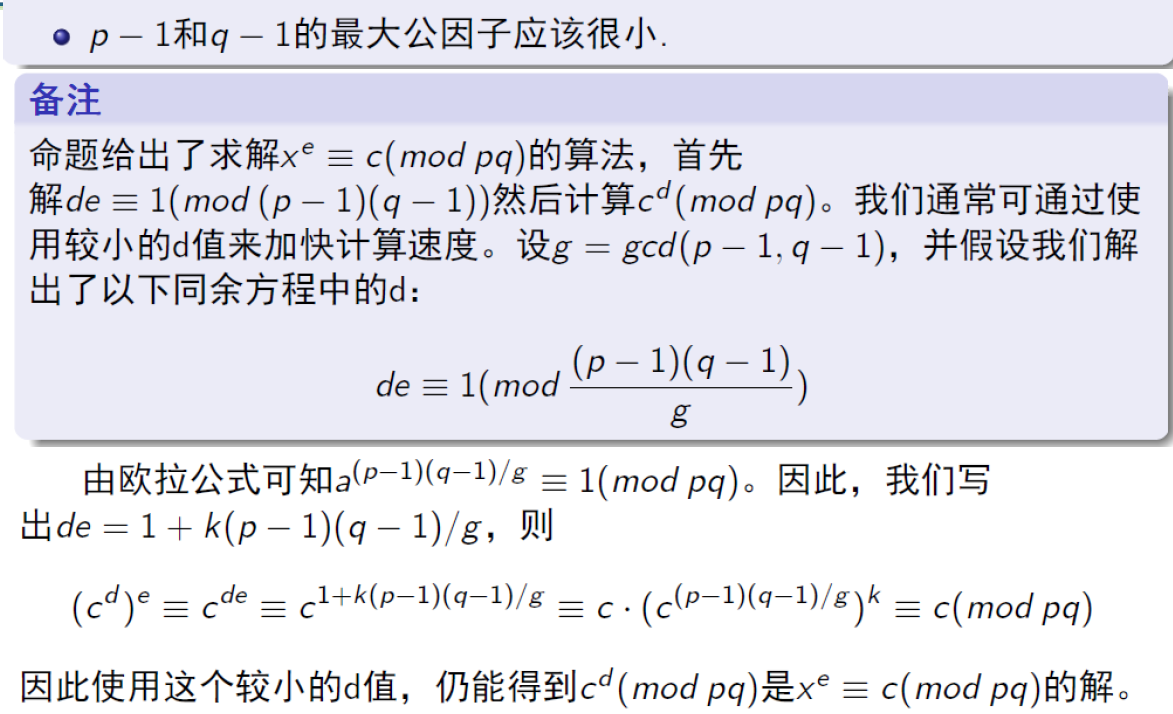
'''连分数算法'''  
def con\_frac(n,d):  
 buf = []  
 while d:  
 buf.append(n // d)  
 n, d = d, n % d  
 return buf  
  
def board(sub\_res):  
 n, d = 1, 0  
 for i in sub\_res[::-1]:  
 d, n = n, i\*n+d  
 return d, n  
  
def list\_frac(x, y):  
 res = con\_frac(x, y)  
 res = list(map(board, (res[0:i]for i in range(1, len(res)))))  
 return res  
  
def get\_pq(b,c):  
 judge = gmpy2.isqrt(b\*b-4\*c)  
 x1, x2 = (-b+judge)//2, (-b-judge)//2  
 return x1, x2  
  
def getkey\_d(e,n):  
 for (d, k) in list\_frac(e,n):  
 if k==0:  
 continue  
 if (e \* d - 1) % k != 0:  
 continue  
 phi=(e\*d-1)//k  
 px,qy=get\_pq(n-phi+1,n)  
 if px\*qy==n:  
 p,q=abs(int(px)),abs(int(qy))  
 d=gmpy2.invert(e,phi)  
 return d  
 print('fail')  
  
d=getkey\_d(e,N)  
print(d)  
flag=pow(c,d,N)  
print(flag)  
m=70802764959962272935287291867539727642893086193535508850980767449442943573660256430563313774743607814949701215649976305035090367943504765  
Flag = n2s(m)  
print(Flag)

(限于水平时间有限，我真的写不动这个攻击的c代码了，于是就上网爬了一段python代码自己理解整理了一下\_\_\_\_苦涩……)

**Level3**

**1.分析：**

分析发现，提供的参数较多，且破解的关键在于求出phi。但问题在于我们只能知道phi的倍数为e\*d-1，且由于加密文有限，无法通过求(e1\*d1-1,e2\*d2-1,…)来计算最大公约数。但事实上，我们知道在实际加密过程中，我们总是用(p-1,q-1)而不是phi作为模数，这就启示我其实不需要算phi的具体值，而是用并不大的e\*d-1代替phi作为模数即可。



**2.具体算法与思路：**

这道题的思路就很简单：直接用e\*d-1作为phi，利用其算出d即可解密。

**3.这种安全问题出现的原因：**

加密算法的缺陷与漏洞导致p-1与q-1的公因子较大，且为了方便生成，提高生成效率导致e较小，使得即使使用phi=ed-1这个相较于(p-1,q-1)很大的数算出的d仍能在有限的时间内求得d。

**4.攻击代码：**

C攻击代码：

int main(int argc, char \*\*argv)

{

char e[1024],d[1024],n[1024],c2[1024],e2[1024];

gets(e);

gets(d);

gets(n);

gets(c2);

gets(e2);

mpz\_t E,D,N,C2,E2,unit;

mpz\_init\_set\_str(E,e,10);

mpz\_init\_set\_str(D,d,10);

mpz\_init\_set\_str(N,n,10);

mpz\_init\_set\_str(C2,c2,10);

mpz\_init\_set\_str(E2,e2,10);

mpz\_init\_set\_si(unit,1);

mpz\_t phi; mpz\_init(phi);

mpz\_mul(phi,E,D);

mpz\_sub(phi,phi,unit);

mpz\_t revE2;

mpz\_init(revE2);

mpz\_invert(revE2,E2,phi);

mpz\_t ans; mpz\_init(ans);

Fp(&C2,&revE2,&N,&ans);

gmp\_printf("%Zd",ans);

return 0;

}

以下是python脚本代码：

N = 10236530053413740615100003541461460233696198138588648053574782642621113532013599569736016589697066609435085953791071316781990891297728233389595200844085429013993604784065921131895163780486688585066824095388606146520812445833320459811669886678897204794566277075500569113375457825702128793449479173591856724944140025593767329361604592004773658512873974858714486136609510924226576855937595607960930558477369540341973218021198150286479745009537887992730203917889741655262591911222415808708605825245334178692506489194219186374615592399978702275393752880411760819063348476014509319635114812321325934297686086105183890300627  
c = 9449320385894752926624804542019323800323541305948652717266559126121615695868092659426277847001567439835260035434412481688638914577933906772911672920571162758248308440365638593572918755830927073758982659548634609355982573550845608735680173453315819466711938957375199402903277153079606741525171024630985724515895565896903068397524362286738892980250891878122253739142469861947568339354395542657625289619671633576553802540817504244245175382927948655218161574440927852601031709226084360488728327725201297539770244693163995690834752002857085540542598689410878678503621852495664608956592426461544966418042286037959678345930  
d = 7676278678887066279526180716380305119365622968116301454872966797599770036938783479770871386651535370562858792127602301314885383471378141464258149521808967650943555490410997949483008519343459024327317083410636123979605430704181848846197447836733408315166260570578470920335354044111545091000734961013978700872537887153995670435626207562409813229076309505503717503916812113256455700205632887791488424784926403183085084573752917550844843596766978304081125871887658805074801729696719479399648102190722552128471085242736033224748556210336929916720666392618752274915222660865983509734543269838240904797844396165050648961873  
e = 8545970560619293415863869061629533131353052726527575105498824066096173134202176407771698168891393791901508803718121936403456285176580496345915728675609872087314945931309067503425183949923543095686689494879877359916226960617400586139116144067054907570518268413146641607525379529898998165978606786256497  
c2 = 5087000621440594322709637255820043188106499641806702306362676329919365177486123582713209295458442881352358564937574804230394285513085672898592567202530951126632285210242520789703382715844914284238941476697077285088249125925212985034224326584500708213790735918447213390775056492281404446380085218159365975419577058738958810204464612511166060396496896787422814724825226392924525692371684984474093454416208048937323512834197598748088935364383049864344370823167974664045741855051202797203588822747547107616777925783800167131111928398533753674856947863108509218085745917218907064709530198706643607925629673283747589808949  
  
  
phi=(e\*d-1)  
'''这里我们不需要解出phi而是用其任意一个倍数便可解出'''  
d2=invmod(e2,phi)  
print(d2)  
Flag=pow(c2,d2,N)  
print(Flag)  
Flag=n2s(Flag)  
print(Flag)

**Level4**

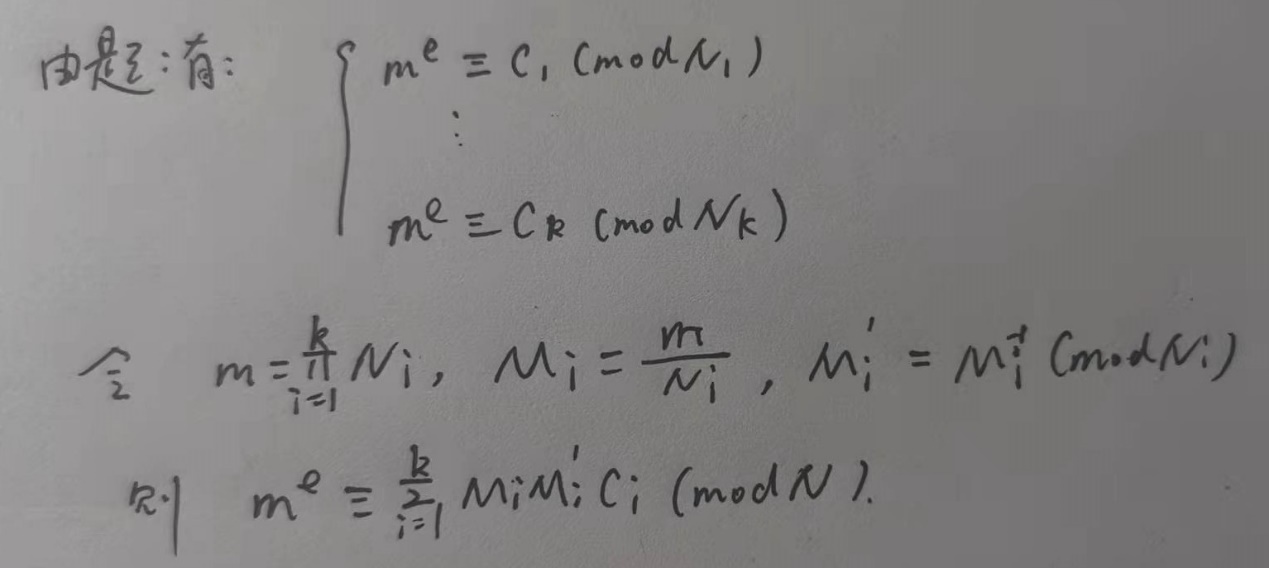
**1.分析：**

在这里助教给出了一句非常关键的提示：即这个加密者经常会发送一些相同的明文信息。且通过观察可知：其加密指数e很小，且都是3。这就是典型的低加密指数广播攻击。考虑到数据组数较多，所以就两两中国剩余定理暴力试就可以了。

(PS:为什么不三组三组试？因为三组n2s函数就爆了-\_\_-haha)。

**2.具体算法与思路：**

暴力破解(也不是那么暴力哈)。



**3.这种安全问题出现的原因：**

为了方便省事，消息并不是采用一次一密的形式加密，因为这样的话需要传输不同的密钥才可解密，除此之外e这个指数也选取得较为随意，而且这么小。这么做的确很方便，但是在方便的同时也给予破解者可乘之机。(由此可见若使用RSA加密，就不能图省事。)

**4.攻击代码：**

n1 = 25080255043824973924491600205173673654914836126141632133628296175294068141097740170756112021717523944656561077347071413616181254260636023015612089767289717341708129170645923885447727986861865879987580652312107571833431677907772207496091723982578969717260733978611113866705959488417604537611898785195053589896770263985143125840142390003829111830708403187779787276770736890079803444222691655225999579438227261427621282061827148359568394865673155085771955880100372378773300556612904248319518128778102272816413698653346326225214449776028604453537806816286462239340926553527908996853516959778553585893013089947092041704341  
c1 = 2842709230876142326160748723953264478122167596313983752014572310948752781051702575610630771011612203915673981867604083101932488970774472199330821377193351243144787546932967588366764362585403614226841689194666102168635466226939232525708406884650733357846215597665259414630801093697168462794580007960470329377130463551704454356039959047394915651848800395128387622651271515030161040085329474888172264402512630304713323167864054943265723623036241446357619766490171466826144113996666406652149617423524629818979356290406504047061896989888439741341  
  
n2 = 21331625540702629675162385801265012214701860172071883735410494163189423777643366078435654098681792960402861512889192158914915407723476091426080477484173546409392459139618250797987490882821028899758841951440381823031763595471491789360727726009031942691267111998802271512764976044384220506582969824702397416931945484478674468868045983564249141402490359919298507358491852410337295035308612166618187413383512004649366795569958663878401352106206759508008095507802412696755495633574201771494600238318758833436843135303137972383760083863213287398506295551440736900964114830936568703723382128176940126607790308763580164280783  
c2 = 17617455825062986168299376761808151045681004358874750328820358366841227787409196507736843223340700440715500219476609523239334433841247271055487041253247596146317206856413846876160682620024998091173688344287745511795669442107091900249158221077059791088126768832344403329348156635079017358556861929845153903733509524069117231093849903821668314089939638453187302041202679316265879307751944792668553294931541144761677413414938754524139023692589642745780370376505332341686879162283814371552149317702157579530595704933443632829849009668200188965260148638659715947847681719027441009442591498019800742561739801263848931220889  
  
n3 = 14031415472367525556594631328480053597587599603502057117983400463757008022824973720928613704425128423297118876463610139563237418995211164787654239049789968822532327604233063914571648566439358363370120606174651920194848826051112432903216656103560572206272109995441562013461231913332475330823126585951341423847006389939068136943502072371324350729617604148988301861703485670199180766939195084254958938430811100568310216990805623643383725186492287366136437569728785530281392001576243167538909980147466650941912598232776664715392607580780934110968622526020203300167256353814892203571456231374809337262553462069760599072297  
c3 = 7046340509272552348776847899321322472980862699447798494072975182456630632603713862027049185530212106182504106297576497629669672090309171717491519773235453458598106659887252067491255228971363783719658620873407508103219576833636893293354937448940123478283016779622437162576393622299211239169145863639654258809082506165649242235094257643744514204219770991244828194131336341652747249553223948451870839401153805329377806688789992866392898415871203861793231445971944284836150151990703655953475083025710907549506371569921949449428381798958615089839584303996330027334713498131947555260897899743048344740694261926243381312088  
  
n4 = 23793398614688432603839829185773009726983677076630510398504281009994912978978843013011671106105072555012431783600842068796912059556435238367874481729603224030205486493776207216544125269348555010140468565594886237234784038499821895056213482688772557809358231673187022756664758900079052164711492198434494686217627795964101197353247961750261228582413166443826665988811924884988813135405136193078676105708835855198575545610930859340301257880841868329646885299538018637274937930148029870880326794286915936731257828813179001864408487894078083405588050312312294241354330466176601867531625677357990437119641724960568872064309  
c4 = 21567463529856129576672925781875257309682157622821338890174810154510158553760540944342300773726602101388182421926701962227698767279400145861530618239232128060625751821044993203756898061643739327591785623955821133220139788187282969109529724358683125847718495254643300148856232249900782787968338164060455037010909851068176851180118932589802643706006903845990692498122816461060624004515024514599905591985809537581178114894287906755197872701258087858045771683916032168848179853939682233242792026302785878089095549574437938817377427777620251016373329765342649365797585480232719344016426864604593060338410930866264568080994  
  
n5 = 20086680429220786401416317723117365101565703118552561885584967799747949413562416087970309005253050995334978845677097039974566195736002278986244780016865856215748791480027304984387200709844524082958556995978269620062009216577376967243855066961183456119924891225681210267473505138637764433941362262941621663820995607347416415613204148419682413846391291847628593182823402061340938003353787864404278919992321575954107744458497857631689626372736089371418122614944632354748842930605538221274791105252741380373264869068232208249880226303224983230618264177754776425277571880166281269134985020054407661956841508283585714835243  
c5 = 2842709230876142326160748723953264478122167596313983752014572310948752781051702575610630771011612203915673981867604083101932488970774472199330821377193351243144787546932967588366764362585403614226841689194666102168635466226939232525708406884650733357846215597665259414630801093697168462794580007960470329377130463551704454356039959047394915651848800395128387622651271515030161040085329474888172264402512630304713323167864054943265723623036241446357619766490171466826144113996666406652149617423524629818979356290406504047061896989888439741341  
  
n6 = 13939630094628797946488082275028808671581765824857548923956107051005919078790500386826799259738430369158172267511393727391612199479458829180355467592857118158346298524013245716870159203188135263923838741847514422499819652137793855198644898162047420204590264851928260379875028386334731732206948737951668722890547668005934029723234632459362757514762205021623703088156751188713003214677810810384976855130055768283164147758358419987970675401195961422029778716497024185398222590535477779144703569324220393253133141861174686399772345879350249035188769808946929163134511474744592104000010688741280974892513700143387056664737  
c6 = 5609175416818467083738781485213729897785803752480886525706026898238231926267016919242771952419174604567857082448729917329010665733255089063468043388085464968844646069180711177540126651845656197049409688220088423817655903332249794364639969167707521670675151127154480183771572288926952662496867627380239383797294262067164986797900883034866880515223104005808577879162492157917834968707194789973761272166444370213934067872426483687512909127059548296642018648693454615849297358569186800536668533049740709336892155676877114154111030611270056109416676809481430382282338968517396956328785974855963543057294187533113504067904  
  
n7 = 15427693701271725146406422130038003555257589197016491088383722043748010429188601215684330344711447177067020890121818098006427953637274200546169576843265533636214623188211302179791213630733551487151533337378565533416211209255725019997596212176739208603996760151013104976123054267684905773126287695464369938502559495490426403517997919584967553136584888521190808034329736711886856091813133396341126358268830219743418009471626625010522633218576813254212993965204071772231968636358248205582824923879400008417590129089050652342003077307675973984056258725209404704897642215638154133246682861626137295440463343105985282042813  
c7 = 2126395448590396304217191000043888891656467987104468653623270884513441754267157723837948836623916521660835134303996654144802087038030652047515940492552886558180357499199667454597610724264056160858233387262333468253569387147572706479000797839747913110824272767642474338676270892431289840016299351917646165835218570466316949440247759576642552455729431972633335193241095105244135078342237883270991069736829672071674804210568015953939107461361207118255899308711285625501617175928748327735588214543295563960360863821338176231460898558076811051943047638474838181110522843655735575331465729748528559060076367695890843999336  
  
n8 = 10619477919273189796320222002344083253705220578530727144627898343181458711158643558816311050423437377772296527491319818077072314784101422807135803450793263026206042489323257182528212711778068652952444008819719985113137642871343003780381650473425057966072228404976625010516060541304969578597847805671416312502274989911673858356688371204605747943749326394222609363520945256130111629777048427302701791983495715325455117886824799721252921431270399446116303462578153014968712815362110212076365775774029656012087792065568354646971398183138336810504551880956398620605868213380506195815589293117911996945008150650018883936307  
c8 = 809937276350763389378236631257511394574956944090889409616398860784122048923432663541169660852271666306141574382084160255921407839611355023705048959492928486086144145418244854985661198922190954490307110057890639722980407890850079516780703899698898255850682070505706464415275821495749977308022176424548584735328632513846250796247546084794402900373334333934279499431169419724669909828918967414537353014746947663411785686914984257142893549333263463156270785861856152415134855379397907048635288255389128272786068040408347708104330983392573853314320523869067298418545865357940818375529031978588211362456729437317924608082  
  
n=[0]\*8; c=[0]\*8;  
n[0]=n1; c[0]=c1;  
n[1]=n2; c[1]=c2;  
n[2]=n3; c[2]=c3;  
n[3]=n4; c[3]=c4;  
n[4]=n5; c[4]=c5;  
n[5]=n6; c[5]=c6;  
n[6]=n7; c[6]=c7;  
n[7]=n8; c[7]=c8;  
  
def crt(b,m):  
 for i in range(len(m)):  
 for j in range(i+1,len(m)):  
 if gmpy2.gcd(m[i],m[j])!=1:  
 print("hhh")  
 return -1  
 MA=1  
 for i in range(len(m)):  
 MA\*=m[i]  
 MAm=[]  
 for i in range(len(m)):  
 MAm.append(MA//m[i])  
 MA\_m=[]  
 for i in range(len(m)):  
 \_,a,\_=gmpy2.gcdext(MAm[i],m[i])  
 MA\_m.append(int(a%m[i]))  
 y=0  
 for i in range(len(m)):  
 y=(y+(MA\_m[i]\*MAm[i]\*b[i]))%MA  
 return y  
  
for i in range(0,7):  
 for j in range(i+1,8):  
 b\_list=[c[i],c[j]]  
 n\_list=[n[i],n[j]]  
 flag=crt(b\_list,n\_list)  
 if(gmpy2.iroot(flag,3)[1]==True):  
 flag=gmpy2.iroot(flag,3)  
 print(flag)

运行后得到如下结果：

(mpz(1416589918079205735555805142498491322588280790787789663534124700945767656662082119835332739921700651256042160062917002964968949898644117439558219789855857554589505252350872866945381), True)

(mpz(25108867560488007243478964472710794411776610108489444054454709496831419724315641256696127689751373359324600291342233585843619368982489108987523229172625296196531008282468756338712777105834187132961737234799906956755077130098), True)

(mpz(25108867560488007243478964472710794411776610108489444054454709496831419724315641256696127689751373359324600291342233585843619368982489108987523229172625296196531008282468756338712777105834187132961737234799906956755077130098), True)

(mpz(48706913991022527987362543539655492816242753570561401788427312025707163993430109698685662989123616887360078131282731278123718973234622582386265520221114636408096619045336715454473190028422171507309909232261215158076597610198236104468028515838813486), True)

(mpz(48706913991022527987362543539655492816242753570561401788427312025707163993430109698685662989123616887360078131282731278123718973234622582386265520221114636408096619045336715454473190028422171507309909232261215158076597610198236104468028515838813486), True)

(mpz(25108867560488007243478964472710794411776610108489444054454709496831419724315641256696127689751373359324600291342233585843619368982489108987523229172625296196531008282468756338712777105834187132961737234799906956755077130098), True)

(mpz(48706913991022527987362543539655492816242753570561401788427312025707163993430109698685662989123616887360078131282731278123718973234622582386265520221114636408096619045336715454473190028422171507309909232261215158076597610198236104468028515838813486), True)

发现里面共有3串不同的数字

将其分别利用n2s函数转化为字符串即可得到flag:

m1=1416589918079205735555805142498491322588280790787789663534124700945767656662082119835332739921700651256042160062917002964968949898644117439558219789855857554589505252350872866945381  
m2=25108867560488007243478964472710794411776610108489444054454709496831419724315641256696127689751373359324600291342233585843619368982489108987523229172625296196531008282468756338712777105834187132961737234799906956755077130098  
m3=48706913991022527987362543539655492816242753570561401788427312025707163993430109698685662989123616887360078131282731278123718973234622582386265520221114636408096619045336715454473190028422171507309909232261215158076597610198236104468028515838813486  
print(n2s(m1))  
print(n2s(m2))  
print(n2s(m3))

**Level5**

**1.分析：**

由其密钥生成过程可知，用于优化RSA解密的dp泄露，因此这属于dp泄露攻击。

**2.具体算法与思路：**

∵dp＝d %(p−1)

∴d=k1×(p−1)+dp

即e×d=k1×e×(p−1)+dp×e

已知e×d≡1 mod ϕ(n)

∴e×d=k2×ϕ(n)+1

∴k1×e×(p−1)+dp×e=k2×ϕ(n)+1

又∵ϕ(n)=(p−1)×(q−1),带入得

k1×e×(p−1)+dp×e=k2×(p−1)×(q−1)+1

整理得dp×e=[k2×(q−1)−k1×e](p−1)+1

由dp=d %(p−1)易知dp>p−1

∴e>[k1×e+k2×(q−1)]

设X=[k1×e+k2×(q−1)]

已知e=65537,只需要枚举X就行

**3.这种安全问题出现的原因：**

保密者对密钥相关信息的保存不当，导致dp信息的流出，进而被攻击者利用，破解出明文。

**4.攻击代码：**

Python脚本：

N=16659300997760469800971290224376055188405686297652398689514699966750126723116002064117480601604943037747361398159106147985508919422144115442803665638128728658456068095805627272151949493099768235110213987402900175324809128343750811803507788253951385183857282649091226102058218238812988497387082154626099446001844686967217426813666119483287719726214784265942445992471692875243715924992161955606027389637593635965771422501424181423022019960185707474490149499710205367794332177006436386209421123924220259239804441163463073610145479813440889158640799505177681406542625538997614027458334771269843334514339334170038078314027  
dp=48430298223907335963485728689319658202804499677434036794856535852972347314776471846394410411486967061516480908639602585647632225376577896237942009226425359825975124798435284015040499948961107152709420314530237448130051250298550512039969853461775554867529219104683128119827897655365157623952886143229669046485  
c=16118635264286591819724705963751211889812705912536292295803417965614159107559198697056539473068152418458980175969934871690002087855396560301175897289980077719798016073231791994414467594046074883194191458471172234619937264130278814363537096977931241792729127721367410110748219497401983218506346258780700742480814667908558355947644315675650889051684821001143224548379997079060899135678315807331985258138419396979691977911495229523035128500932572555864549420974049774727186938388827289277587031239590109296477770421457790807593386855594075365007397804373502264699762750665721446592727180624223203930778071293015483928565  
e = 65537  
  
for x in range(1, e):  
 if(e\*dp%x==1):  
 p=(e\*dp-1)//x+1  
 if(N%p!=0):  
 continue  
 q=N//p  
 phi=(p-1)\*(q-1)  
 d=gmpy2.invert(e, phi)  
  
  
  
flag = pow(c,d,N)  
print(flag)  
flag=1080364455565827654440627110115628789487833645064395287020077039529496452141946326639460665145938014957566092762495632125345575018877  
m=n2s(flag)  
print(m)

C语言：

int main(int argc, char \*\*argv)

{

char n[1024],dp[1024],c[1024],e[1024];

gets(n);

gets(dp);

gets(c);

gets(e);

mpz\_t N,DP,C,E,unit,none,x,range,edp,p;

mpz\_t step1,step2,step3,q,phi,p\_1,q\_1,d;

mpz\_init\_set\_str(N,n,10);

mpz\_init\_set\_str(DP,dp,10);

mpz\_init\_set\_str(C,c,10);

mpz\_init\_set\_str(E,e,10);

mpz\_init\_set\_ui(unit,1);

mpz\_init\_set\_ui(none,0);

mpz\_init\_set\_ui(x,1);

mpz\_init(range);

mpz\_add(range,E,unit);

mpz\_init(edp); mpz\_mul(edp,E,DP);

mpz\_init(p);

mpz\_init(step1); mpz\_init(step2); mpz\_init(step3);

mpz\_init(q); mpz\_init(phi) ; mpz\_init(p\_1);

mpz\_init(q\_1); mpz\_init(d);

for(;mpz\_cmp(x,range)<0;mpz\_add(x,x,unit))

{

mpz\_mod(step1,edp,x);

if(mpz\_cmp(step1,unit)==0)

{

mpz\_sub(step2,edp,unit);

mpz\_div(p,step2,x);

mpz\_add(p,p,unit);

mpz\_mod(step3,N,p);

if(mpz\_cmp(step3,none)!=0)

continue;

mpz\_div(q,N,p);

mpz\_sub(p\_1,p,unit);

mpz\_sub(q\_1,q,unit);

mpz\_mul(phi,p\_1,q\_1);

mpz\_invert(d,E,phi);

}

}

mpz\_t flag; mpz\_init(flag);

Fp(&C,&d,&N,&flag);

gmp\_printf("%Zd",flag);

return 0;

}

**RSA-OAEP的思考题**

**为什么要在加密中引入随机化**？

首先引入这样的一个思考：

**RSA的短明文可以导致密文易遭受攻击，因为一些短明文对应的密文中存在”先验信息“，这个”先验信息“对攻击者Malice来说就是好事，但对于用户就非常不好。**

于是，考虑给明文填充伪数据（填充位）也许可以对攻击者Malice的工作变得困难些，比如说：

1.在原明文后面添加t个（t是随机的）零，要保证添加后的明文的长度要小于模数N，否则会出现me=m’e=c ( mod N)这种情况，也就是说两种明文加密为同一个密文；

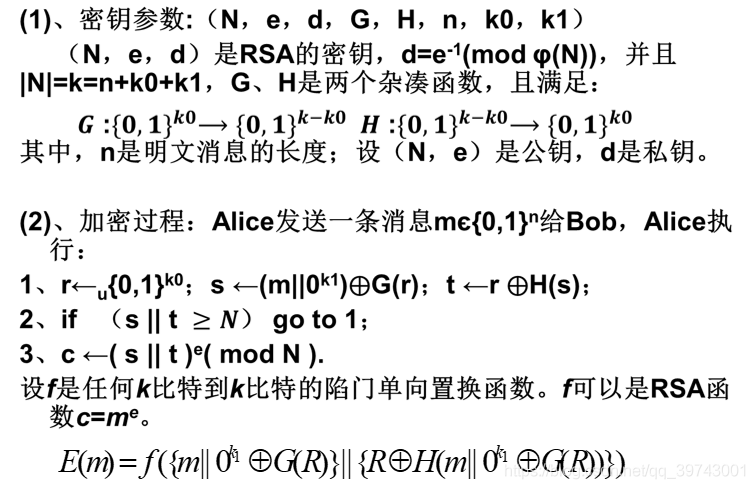
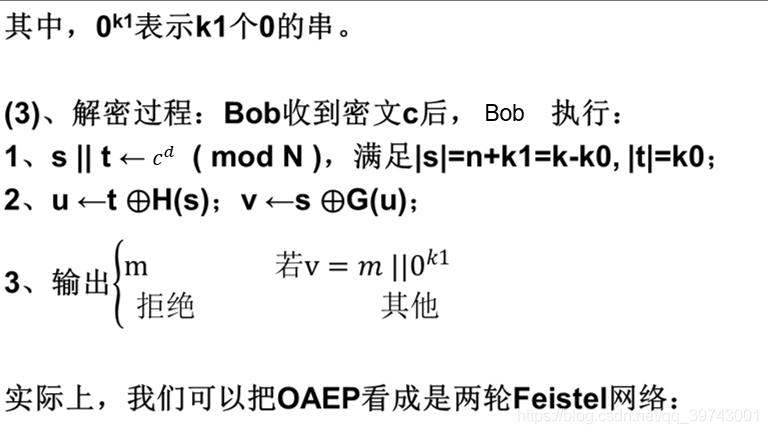
2.在原明文后串接上原明文的Hash值，当然这也要保证串接后的明文的长度与（1）的保证一样。

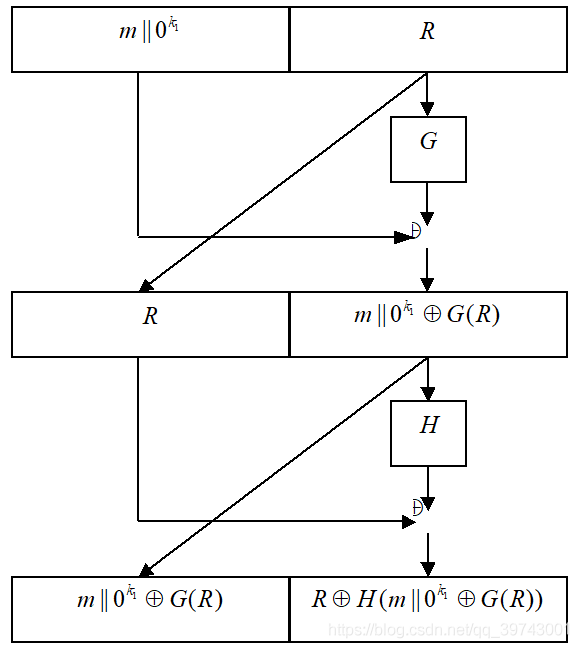
这里的解决方式就用到了最优非对称加密填充（OAEP）。

OAEP变换是把密码学杂凑函数和一个著名对称密码算法结构结合起来构造的。OAEP构造可以看成是一个两轮Feistel密码，第一轮使用杂凑函数G，第二轮使用杂凑函数H，代替Feistel密码的“s盒函数”，但这里的两个“s盒函数”不是加密钥，而且两个半分组的大小可以不同。  
​

​ 正因为OAEP有良好的代数特性，具有逐比特的代数特性：因此，如果Malice想通过篡改密文，都会导致明文的改变是不可控的，因为使用了杂凑函数，在保证杂凑函数的安全性下（比若说单向性、抗碰撞性等），很难找到一个有效的hash值，其对应的明文是有效的。这种改变是由于破坏了增加到明文消息中的冗余。

算法如下：



 Feistal网络

如果密文发出后没有被修改过，那么我们就知道，Bob可以用解密算法恢复出随机数r。

其实，换种说法就是：**如果攻击者试图以适应性的方式修改密文，那么数据完整性检验将以“压倒性”概率失败，并且解密结果为一条没有任何意义的明文。**

其次，除了上面RSA-OAEP保护了选择密文攻击下的安全性，值得注意的是，该算法提供的数据完整性保护很奇特，当解密者Bob看到解密后明文的k1个0的传之后，能够确信一点的就是，密文没有 被修改过，但是Bob却永远不知道该消息的发送者。

那么怎样才能知道消息的发送这是谁？

答：（可以用消息发送者的私钥对整个密文进行加密）

​ 这种公钥加密体制在抗击CCA2的公钥密码系统中，解密过程还能保证数据的完整性验证，可以说这种密码系统内是“无懈可击的”。即便是下面的极端攻击形式：

攻击者和目标用户进行询问-应答的模式。攻击者可以自由的向公钥拥有者发送任意多的“选择适应性密文”的消息，要求目标用户以预言机-服务的方式进行解密，意思就说解密是正确的，不会故意解密错的明文发给攻击者。当然，目标用户也不是真的傻，自己也有一定的识别能力，不会帮助解密之前刚发出的密文。当OAEP所基于的陷门单向置换f是单向安全时，f-OAEP在随机语言机模型下可证明达到适应性选择密文攻击下的不可区分性（IND-CCA2）.

(以上内容摘自《现代密码学理论与实践》，作者为毛文波)

心得体会

怎么说呢，其实这次编程实验对我来说应该算是一个不小的挑战。不会python的我一开始就选择了一条更为困难的路，就是用c实现RSA。尽管过程并不是那么一帆风顺，但是我在这个过程收获到的也一定要比其他人多，这也是这个实验的魅力所在。但是由于时间精力有限，pem的编解码做的并没有我想象中的那么好，在代码方面还有很多可以优化的空间，所以还是觉得有点遗憾，或许在之后有空的时间里我还会去思考这些我觉得非常有意义的东西。

谈到数论，其实我个人觉得我很喜欢数学，虽然高中时被数论知识劝退，但是现在看来数论确实是一门十分灵活，一门非常有深度以至于可以深不可测的一个领域。这也是我切身实地的感受到数字的魅力的一门学科。希望在不远的将来，数论的知识能陪伴我解决一个又一个问题，这是一件非常幸福的事情。