#include"iostream"

#include"cstdio"

#include"cstring"

#include"cstdlib"

#include"ctime"

#define N 10000000000

#define ll long long

/\*

1.明文数字化我才用ASCII码的直接替换

2.随机数选取的时候的扩展倍数，因为我没有写大整数类，时间紧迫，我选择用long long代替大整数

long long的最大表示上限是1.8446744073709552 E19,经过科学计算锁定素数的最大上限是4294967296

这里我用N的1e10来选定大随机素数

3.这里对于随机数用Miller-Rabin算法，利用2,3,5,7,11作为测试参数可以再2152302898747中100%确定素数

但是对于更大的素数（用的整数类）需要则会出现概率性错误，但是容错概率高于99.09%

4.?? 老师我这里有个问题，私钥和公钥的选取是随机的吧 ??希望可以得到解答，因为从本质上看，私钥和公钥没有什么差别

选择任一个都可以

5.?? 老师加密指数是越大越好吗 ??从算法本质上看的话，没有必要选择大的整数？？我这里将加密指数设定成用户选择可以吗？

6.老师随机生成大素数我的PC机跑了很长时间没有结果，为了验证RSA类的模块其他成员都正确，我用快速线性筛晒了两个大素数

并问一下，怎么快速生成两个大随机素数，我查了资料，除了随机没有其他的好方法？

7.老师我的程序解决书本上的物体一进房绰绰有余，但是一旦素数大了，结果就不对了，我还是搞不懂为什么，甚至连定理都不正确？

\*/

using namespace std;

int length; //加密文本的长度

ll pre[100]; //存储待加密信息，字符

ll aft[100]; //存储解密后信息，数字

bool isprime[N/100];

int prime[N/100];

int number=0;

class RSA

{

public:

RSA(ll,ll);

RSA() //构造函数+格式输入

{

//随机生成大素数

double time=clock();

init\_prime\_new();

p=prime[number-1];

q=prime[number-2];

printf("随机生成素数耗时是:%.2lfs\n",(clock()-time)/1000);

n=p\*q;

printf("n:%lld\n",n);

Euler\_n=(p-1)\*(q-1);

printf("Euler\_n:%lld\n",Euler\_n);

//代码嵌套Miller-Rabin素数检测法,保证返回一个小于4294967296的大素数

printf("请选择随机加密指数（公钥）:(0-7)\n");

while(1)

{

scanf("%lld",&public\_key); //随机生成加密公钥,这个随机选择的只能是公钥

if(judge\_gcd(public\_key,Euler\_n)==1) break;

else printf("公钥选取错误,请重新选择\n");

}

move=false; //还没有生成私钥

}

~RSA(){};

inline bool return\_status() //返回是否加密过这个状态

{

return move;

}

inline void init\_prime\_new();

inline ll init\_prime(); //生成随机数函数

bool Miller\_Rabin(ll,int); //大整数素性检测

ll Extends\_gcd(ll,ll,ll&,ll&); //第三个参数是我们生成的秘钥

ll quick\_mod(ll,ll,ll); //快速幂取模函数

ll judge\_gcd(ll,ll);

void creat\_private\_key();

//加密函数

void encoding();

//解密函数

void decoding();

private:

ll buffer; //明文处理，一次处理一个字符

//大随机素数

ll p;

ll q;

//

ll n; //公共模数

ll Euler\_n; //n的欧拉函数

ll private\_key; //私钥

ll public\_key; //公钥

bool move; //判断是否生成过私钥秘钥

};

ll RSA::judge\_gcd(ll a,ll b)

{

return b?judge\_gcd(b,a%b):a;

}

RSA::RSA(ll p,ll q) //重载

{

bool flag=true;

int table[5]={2,3,5,7,11};

while(flag)

{

int t=0;

for(int i=0;i<5;i++)

{

if(Miller\_Rabin(p,table[i])&&Miller\_Rabin(q,table[i])) t++;

else

{

printf("非素数，重新输入\n");

scanf("%lld%lld",&p,&q);

break;

}

}

if(t==5) break;

}

n=p\*q;

printf("n:%lld\n",n);

Euler\_n=(p-1)\*(q-1);

printf("Euler\_n:%lld\n",Euler\_n);

//代码嵌套Miller-Rabin素数检测法,保证返回一个小于4294967296的大素数

printf("请选择随机加密指数（公钥）:(0-7)\n");

while(1)

{

scanf("%lld",&public\_key); //随机生成加密公钥,这个随机选择的只能是公钥

if(judge\_gcd(public\_key,Euler\_n)==1) break;

else printf("公钥选取错误,请重新选择\n");

}

move=false; //还没有生成私钥

}

void RSA::init\_prime\_new() //快速线性筛筛大素数

{

memset(isprime,1,sizeof(isprime));

isprime[0]=isprime[1]=0;

for(ll i=2;i<N/100;i++)

{

if(isprime[i]) prime[number++]=i;

for(int j=0;j<number&&i\*prime[j]<N/100;j++)

{

isprime[i\*prime[j]]=0;

if(i%prime[j]==0) break;

}

}

}

void RSA::creat\_private\_key()

{

ll x;

ll y;

Extends\_gcd(public\_key,Euler\_n,x,y);

if(x<0) private\_key=Euler\_n+x;

else private\_key=x;

move=true;

printf("私钥:%lld\n",private\_key);

}

ll RSA::quick\_mod(ll a,ll b,ll c)

{

ll ans=1;

a=a%c;

while(b!=0)

{

if(b&1) ans=(ans\*a)%c; //溢出了,C没有大整数类型

b>>=1;

a=(a\*a)%c;

}

return ans;

}

bool RSA::Miller\_Rabin(ll n,int t) //t是测试参数2,3,5,7,11的其中一个，保证素性的100%正确

{

ll p=n-1;

ll w=0;

while((p&1)==0)

{

p>>=1;

w++;

}

ll a=quick\_mod(t,p,n);

for(int i=1;i<=w;i++)

{

ll y=quick\_mod(a,2,n);

if(y==1&&a!=1&&a!=n-1) return false;

a=y;

}

if(a!=1) return false;

else return true;

}

ll RSA::init\_prime()

{

ll small\_ll;

while(1)

{

small\_ll=rand();

ll help;

while(small\_ll<=1294967000) help=small\_ll,small\_ll=small\_ll\*small\_ll; //平方生成大素数

int table[5]={2,3,5,7,11}; //Miller-Rabin算法的参数

bool judge=true;

for(int i=0;i<5;i++)

{

if(Miller\_Rabin(help,table[i]));

else

{

judge=false;

break;

}

}

if(judge) return small\_ll;

}

}

ll RSA::Extends\_gcd(ll a,ll b,ll& x,ll& y)

{

if(b==0)

{

x=1;

y=0;

return a; //返回最大公约数

}

ll t=Extends\_gcd(b,a%b,x,y);

ll w=x;

x=y;

y=w-(a/b)\*y;

return t;

}

void RSA::encoding() //用公钥加密

{

//逆序加入，倒序

for(int i=0;i<length;i++)

{

buffer=pre[i];

aft[i]=quick\_mod(buffer,public\_key,n);

}

}

void RSA::decoding()

{

for(int i=0;i<length;i++)

{

ll buffer=aft[i];

ll w=quick\_mod(buffer,private\_key,n);

//这里其实可以加上异常处理，但是因为只考虑算法本质我就不写那么多了

printf("%lld ",w);

}

printf("\n");

}

int main()

{

//输入

printf("输入待加密文字长度\n");scanf("%d",&length);

for(int i=0;i<length;i++) scanf("%lld",&pre[i]);

//加密

RSA test(4,4);

test.creat\_private\_key();

test.encoding();

//密文输出

printf("加密后密文\n");

for(int i=0;i<length;i++) printf("%lld ",aft[i]);

printf("\n");

//解密输出

test.decoding();

return 0;

}