RSA加密算法

俸昊嵘 16307110262

加密与解密

- 给一段文字。由于字符串最终也可编码为数字，问题直接简化为加密一个数字明文
- （无穷长的字符串也可以分段编码，每段各自加密）
- 对明文X进行一定变换$\mathscr{F}$得到密文Y
- 对密文Y进行逆变换$\mathscr{F}^{-1}$得到译文=明文X

- 即：$\mathscr{F}(X)=Y$；$\mathscr{F}^{-1}(Y)=X$

RSA——一种非对称加密算法

- RSA算法中，使用者持有三个数E,D,N。
- E,N是对外公开的；D是密钥。
- 请他人给自己发送信息X时，用$Y=X^E \bmod N$生成密文Y。
- 密文Y送达，用私有的密钥解密：$X=Y^D \bmod N$。
- 即需要上述两个变换互为逆变换。
- 理论上已知E,N可以求出密钥D；但可以使D很难算出。
- 实际上RSA中解开密钥涉及N的因数分解，这对于很长位数的整数N目前没有办法。
- 这就是“非对称”。

欧拉φ函数

- **小于m且与m互质的正整数个数**称为欧拉函数，记作φ(m).
- 若$m=(p_1^{n_1})*(p_2^{n_2})*(p_3^{n_3})*\cdots\cdots,(n \in N^*)$ 则 $\varphi(m)=m*(1-1/p_1)*(1-1/p_2)*(1-1/p_3)$
- **当m为质数时，φ(m)=m-1.**
- 当m为质数，k为正整数时，
- $\varphi(m^k)=(m-1)*(m^{(k-1)})$
- 若(a,b)=1,则φ(ab)=φ(a)*φ(b).

欧拉定理

- 欧拉定理：$m \geqslant 2$,且$(a,m)=1$，则$a^{\varphi(m)} \bmod m=1$

- 如果n非常大，怎么化简$a^n \bmod c$？（a,c互质）

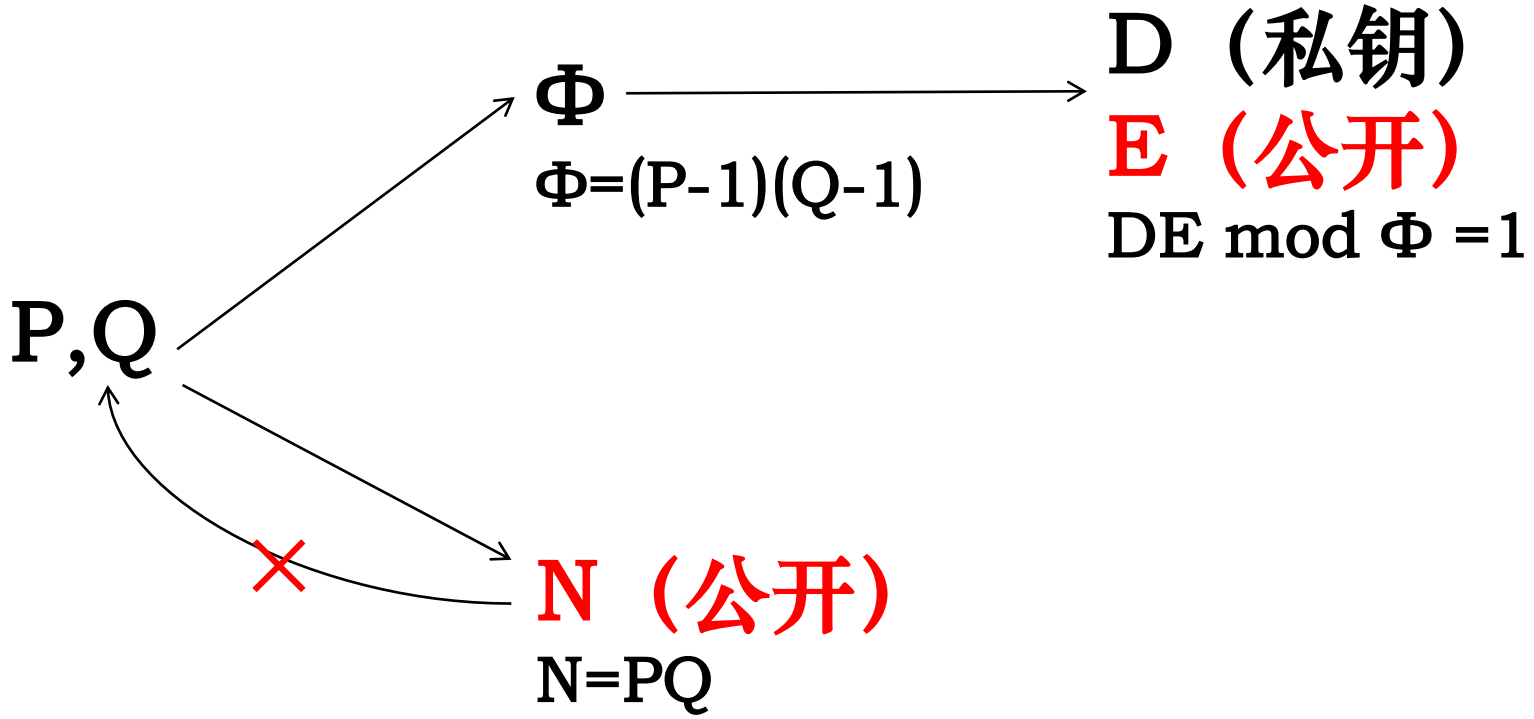
  $(a \bmod c)^{n \bmod \varphi(c)} \bmod c$

RSA步骤

- 找两个大素数P,Q；令N=PQ，即为模运算共用除数
- 欧拉函数$\Phi=\varphi(N)=(P-1)(Q-1)$
- 找一个$E<\Phi$，使$(E,\Phi)=1$，即为公钥
- 解出$D<\Phi$，使$DE \bmod \Phi = 1$，即为私钥。
- 可以证明这样产生的$Y=X^E \bmod N$与$X=Y^D \bmod N$是互逆运算。（$X^{DE} \equiv X^1 \pmod N$，用Euler定理，分P,Q是否互质两种情况）

- “找数”的基本思路：取一个范围恰当的随机数，逐渐往上加1直到满足条件。
- 还需要：对超大的整数进行运算 快速检验是否素数 求幂 求最大公约数 解线性同余方程

非对称性



如何存储与运算超大数字

- 系统自带的数据类型精度不够，容易溢出。
- 而实际应用的N应是上千二进制位的。（较长才不可被破译）

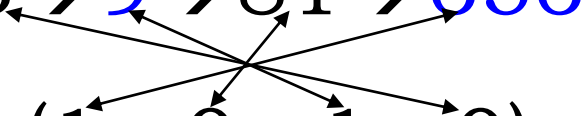
- 解决方法：
- 用数组代替单个数据——“高精度”
- 但各种基础运算都要自己写（复习小学数学，一“位”“位”地算），增加一些（duī）子VI，比较繁琐

快速幂(循环版)

```
m=1;   //存所求
s=a;   //滚动幂
while(n>0)
{    if(n mod 2 ==
 1)
          m=m*s;
     s=s*s;
     n=n/2;    }
return m;
```

如：求3^{10}

s=3→9→81→6561→……



$10=(1\quad 0\quad 1\quad 0)_2$

m=9*6561

最大公约数(gcd)

- 欧几里得(Euclid)算法:

```
int gcd(int a,int b)
{
      if(b==0) return a;
           else return gcd(b,a mod b);
}
```

- 即：若其中一数(b)为0,最大公约数等于另一数；否则进行辗转相除。（递归）

线性同余方程

- 形如$ax \equiv b \pmod n$的方程
- 可以用扩展Euclid算法，不过在听说它之前我想了另一种算法（其实是等价的）。
- 设f(a,b,c)为$ax \equiv b \pmod c$的解x.($0 \le x < c$)
- 等价于求ax=ny+b的整数解
- 移项: cy=ax-b.
- 那么y=f(c,-b,a)。形成了递归。
- 每一步都可以趁机将a,b模一下c。使数据逐层减小。
- 还没完，我们要求的是x=(cy+b)/a.
- 递归终点:a=0。这时x是任意的，取为0可以使最终结果<n。

- Labview递归很麻烦，我只好存储了每一步的a,b,c，循环到终点后从0开始反着算回去。

线性同余方程

```
bool can=1;           //假定有解,若can=0则无解
int f(int a,int b,int c)     //aX mod c = b
{
  a%=c;   if(a<0) a+=c;        //a和b对c取模,注意负数处理
  b%=c;   if(b<0) b+=c;
  if(a==0)                  //0*X mod c = b
  {
       if(b==0)
            return 0; //0*X mod c = 0,任意解,返回最小非负整数0
       else           //剩下的情况,0*X mod c = n (n!=0),无解
            {can=0;   return 0;}
  }
  return (f(c,-b,a)*c+b)/a;//这步为防越界可考虑用long long
}
```

费马小定理与素数测试

- **费马小定理：p是质数,且(a,p)=1，则$a^{p-1}\equiv 1 \pmod p$. (欧拉定理特例)**
- **推广:p是质数,则对任意整数a,$a^p\equiv a \pmod p$.**
- **费马小定理的逆命题不成立，**所以不能由费马小定理**确定**一个数为素，但增加测试量（取若干不同的小质数做测试）为质数的机率很高。虽然绝大部分合数已被筛除，但以防万一，针对极少数顽固的~~混入无产阶级革命队伍中的走资派~~通过费马小定理测试的“**伪素数**”，需要Miller-Rabin素性测试。

Miller-Rabin素数判定

Fermat小定理：如果n是素数，则对于所有不是n倍数的a，有$a^{n-1} \equiv 1(\mathrm{mod} n)$

由此定理，假设$a = 2$时不满足此式，可以确定的说n是合数。然而如果$a = 2$时成立，并不能确定n是素数，因为$2^{340} \bmod 341 = 1$，而$341 = 11 \cdot 31$不是素数。$a = 3$也有反例，如$3^{90} \bmod 91 = 1$，而$91 = 7 \cdot 13$不是素数。

最糟糕的情况是：n是合数，但$1 \sim n-1$中所有与n互素的a全部满足上式，这样我们不管选哪个a都将得到错误的结果。我们称这样的数为Carmichael数。不与n互素的数a显然不满足上式，因为同余符号右边一定是$gcd(n, a)$的倍数。因此Carmichael数是最糟糕的情况。

Miller-Rabin测试 可惜Carmichael数有无穷多，我们需要继续变形。设n为大于等于5的奇素数，写成$n - 1 = 2^r s$。由于$n-1$是偶数，因此$r \geq 1$。由Fermat定理，序列

$$a^s \bmod n, a^{2s} \bmod n, a^{4s} \bmod n, \ldots, a^{n-1} \bmod n \tag{5.11}$$

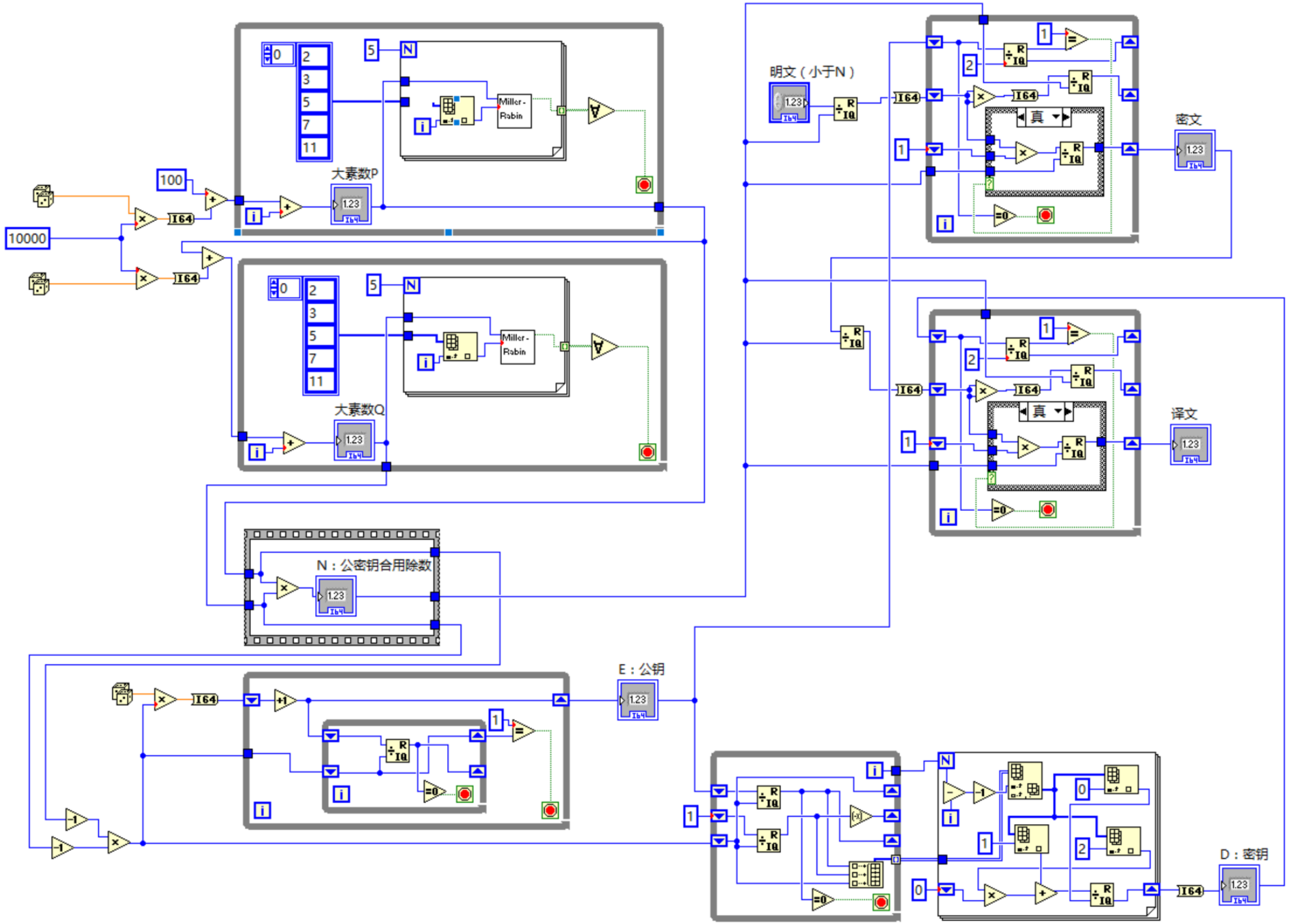
必定以1结束，而且在第一次出现1之前的值必定是$n-1$。这是因为n是素数时，$x^2 \equiv 1(\mathrm{mod} n)$的唯一解为$x = \pm 1$（方程两边取离散对数即可）。

考虑除了最后一个数外的其他数。如果第一个数是1或者任何一个数为-1（意味着下一个数为1），都说明n可能是素数，否则是合数。这样的测试称为Miller-Rabin测试，代码如下（注意做乘法时应使用前面介绍的mul_mod函数而不能直接乘）：

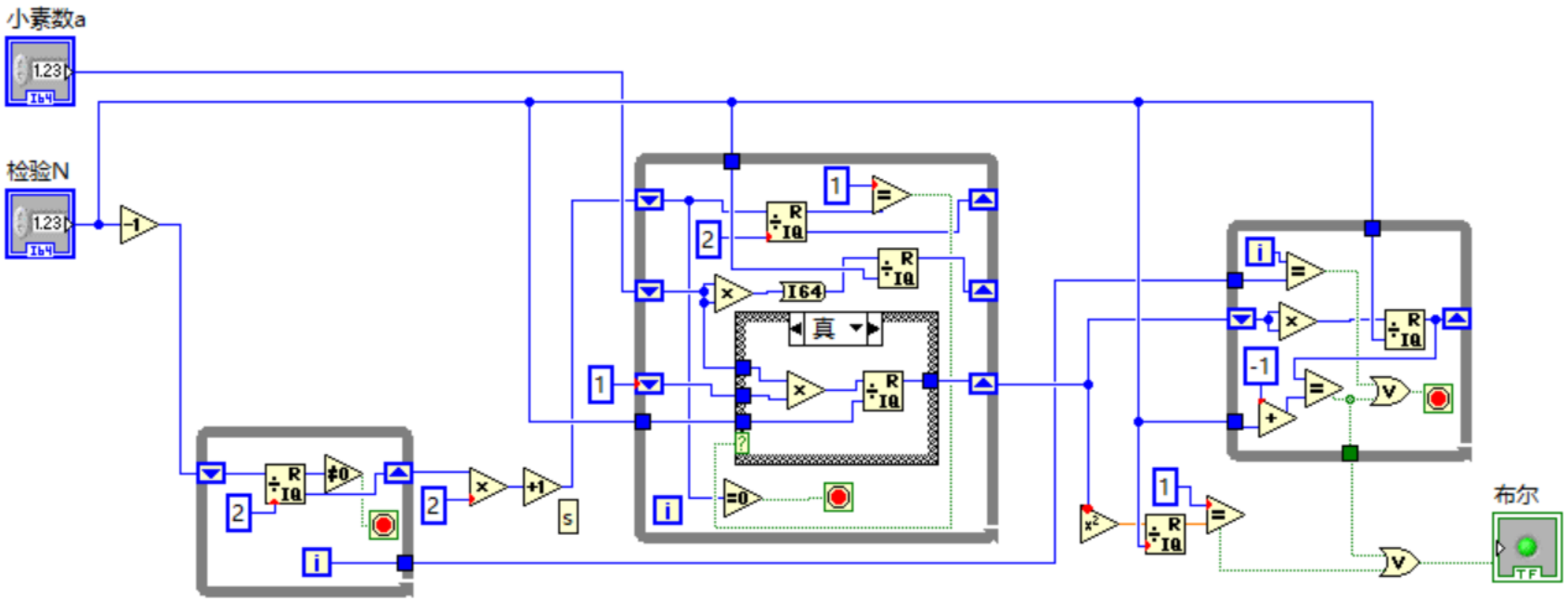
第一个可以同时骗过2~11这五个质数M-R测试的伪素数已达到13位，故此程序用这几个小质数做测试就够了，全部通过测试就可以判为质数。

实现

主程序图


0
2
3
5
7
11
5
N
Miller-Rabin
A
大素数P
100
10000
I64
大素数Q
N：公密钥合用除数
E：公钥
明文（小于N）
密文
译文
真
D：密钥
1.23
I64

Miller-Rabin子程序图



References & Further Reading

- 俸昊嵘，数学专题-南宁三中信息学竞赛培训讲义，2014
- 刘汝佳等，算法艺术与信息学竞赛-学习指导，2005
- T. H. Cormen *et al*，算法导论，2006
- http://www.ruanyifeng.com/blog/2013/06/rsa_algorithm_part_one.html
- http://www.ruanyifeng.com/blog/2013/07/rsa_algorithm_part_two.html
- https://blog.csdn.net/dbs1215/article/details/48953589