素数判定设计与实现

谢日敏*

(福建商业高等专科学校 计算机系,福建 福州 350012)

〔摘 要〕 本文对素数判定测试算法进行分析,并结合 Miller-Rabin测试算法,基于 Miracl大数运算库,采用 VC++.NET语言实现了 Rabin素数测试算法。

〔关 键 词〕 素数定理; Fermat定理; Rabin-Miller测试法

中图分类号: TP309 文献标识码: A 文章编号: 1008-4940(2007)02-0117-004

随着 E-Commerce的快速发展,信息的安全性已不仅是军事和政府部门感兴趣的事,各企业也越来越感到信息安全性的重要性。而 RSA算法是目前最优秀的公钥解决方案之一,其安全性建立在大整数分解为两个素数之积的困难性假设基础之上。

因此, RSA算法的核心问题是要解决通过何种方式能快速的找到两个大的随机素数。这样既有利于提高 RSA加密的安全性,又有利于提高加密效率。

1 素数检测

对于大素数检测的实现过程,具体的步骤有以下两个部分构成:

- 生成大的“随机数”;
- 使用概率多项式时间算法进行素数测试。

对于产生大随机数,可以通过使用商业 Miracle大数运算库实现,而提高素数判定测试算法的效率成为关键问题。

1.1 概率测试算法

一般来说,直接判定一个数是大素数复杂度较高,但是确认它不是大素数要容易得多。例如,被 2整除的偶数就不是大素数,同样,被 5整除的大数,也不属于大素数。我们可以通过以上的判断,再对余下的大数进行素数判定。由于大素数的分布具有稀疏的特征,我们还要将合数过滤掉。

定理 1(素数定理)[9] 设 $\pi(x)$为小于或等于 P的全部素数个数,则当 $\lim\limits_{x\to\infty}\pi(x)/(x/\ln x)=1$ 当 x充分大时, $\pi(x)\approx x/\ln x$

由素数定理产生的算法主要有 Solovay—Strassen素数测试算法、Lehmann算法、Miller-Rabin算法等[10]。对一个奇整数的 Solovay—Strassen素数测试算法为:

1)选择一个随机整数 n. $1\leqslant d\leqslant n-1$;

2)计算 gcd{a, n);

3)若 gcd{a, n)≠1,则 n非素数;

4)计算 (a/n)及 $a^{n-1/2} \bmod n$;

5)若 $(a/n)\equiv a^{n-1/2} \bmod n$,则 n可能是素数,否则 n是合数。

这个算法的正确性至少为 1/2,出错的概率小于 1/2若随机均匀地产生序列 $a_1, a_2, a_3\cdots\cdots a_n$,都通过此判定法来判定是素数,但 n是合数的概率为 $(1/2)^k$,当 k充分大时, $(1/2)^k$是很小的数。

另一种测试方法是由 Lehmann提出来的算法:

(1) 选择一个小于 n的随机数 a。

(2) 计算 $a^{(n-1)/2} \bmod n$。

(3) 如果 $a^{(n-1)/2}\neq 1$或 $-1 \pmod n$,那么 n肯定不是素数。

(4) 如果 $a^{(n-1)/2} \bmod n$或,那么 n不是素数的可能性值是 50%。同样,重复 t次,那么 n可能是素数所冒的错误风险不超过 $(1/2)^t$。

1.2 Miller-Rabin素数测试算法

定义:令 $n-1=2^f m$,其中 f是非负整数, m是正奇数.若 $b^m\equiv 1 \pmod n$或 $b^{2fm}\equiv -1 \pmod n$,则 $0\leqslant j\leqslant f-1$称一通过以 b为基的 Miller-Rabin测试。

定理 2(Fermat定理)[1,5] 若 n是素数,则对于任意的整数 a,应有 $a^{n-1}=1 \bmod n$,此定理给出素数 n的必要条件,若不满足,则可断定它不是素数。

例如, 67是一个素数,则 $2^{66} \bmod 67=1$。

利用 Fermat定理,对于给定的整数 n,可以设计一个素数判定算法。通过计算 $d=2^{n-1} \bmod n$来判定整数 n的素性。当 d不等于 1时, n肯定不是素数;当 d等于 1时, n则很可能是素数。但也存在合数 n使得

*收稿日期: 2006-06-07

作者简介:谢日敏(1979-),男,福建商业高等专科学校计算机系助教

$2^{n-1} \equiv 1 (mod\ n)$。例如,满足此条件的最小合数是 n = 341。为了提高测试的准确性,我们可以随机地选取整数, Fermat定理毕竟只是素数判定的一个必要条件。满足 Fermat定理条件的整数 n未必全是素数。有些合数也满足 Fermat定理的条件。这些合数被称做 Carmichael数,前 3个 Carmichael数是 561, 1105, 1729。Carmichael数是非常少的。在 1~100000000范围内的整数中,只有 255个 Carmichael数。

定理 3若 n是素数, b是整数,且 n\b,则必须通过以 b为基的 Miller-Rabin测试.

证明 令 $S_k \equiv b^{\frac{n-1}{2^k}} (mod\ n) \equiv b^{2^{1-k}} (mod\ n)$ k= l, l-1……, 2, 1, 0

其中 $S_t \equiv b^m (mod\ n)$, $S_0 \equiv b^{n-1} (mod\ n)$

若 n是素数,则由 Fermat定理可知: $S_0 \equiv b^{n-1} (mod\ n)$ 必然成立,它的必然结果是

$S_1 \equiv b^{(n-1)/2} (mod\ n) \equiv 1 (mod\ n)$或 $S_1 \equiv -1 (mod\ n)$ 必然成立,而且或 $S_1 \equiv b^{(n-1)/2} (mod\ n) \equiv 1 (mod\ n)$ 或 $S_1 \equiv -1 (mod\ n)$成立时, Fermat定理必然成立.因此 $S_0 \equiv {S_1}^2 \equiv 1 (mod\ n)$。同理,若 $S_2 \equiv 1 (mod\ n)$或 $S_2 \equiv -1 (mod\ n)$成立时,则 $S_1 \equiv 1 (mod\ n)$,因而满足 Fermat定理。

依次类推 若已知整数 k>0

$S_{k+1} \equiv 1 (mod\ n)$或 $S_{k+1} \equiv -1 (mod\ n)$

则 $S_k \equiv S_{k-1} \equiv S_{k-2} \equiv S_{k-3} \cdots\cdots \equiv S_0 \equiv 1 (mod\ n)$ 即 Fermat定理成立。

定理说明 Miller-Rabin测试一旦通过, Fermat定理便可满足。

定理 4若一是奇合数,则 n通过以 b为基的 Miller-Rabin测试数目最多为 (n-1)/4, $0 \leqslant b \leqslant n-1$。

程序算法:

1) 先计算出 m、j使得 $n-1 = m \times 2^j$,其中 m是正奇数, j是非负整;

2) 随机取一个 b, $2 \leqslant b$;

3) 计算 $v = b^m mod\ n$;

4) 如果 v==1,通过测试,返回;

5) 令 i=1;

6) 如果 v=n-1,通过测试,返回;

7) 如果 i==j,非素数,结束;

8) $v = v^2 mod\ n$, i=i+1;

9) 循环到 5)。

程序实现:

```
/* * * * * * * * * * * * * * * * * * * * * * * * *
* 函数原型: BOOL RabinMillerKnl(big n)
* 功能说明: Rabin-Miller素数测试
* 输入参数: n为大数
* 输出参数: 无
* 返 回 值: FALSE:不是素数; TRUE是素数。
* 备    注: 此函数利用了 Miracl大数运算库的加、减、乘、除、模逆
* 注    意: 通过测试并不一定就是素数,非素数通过测试的概率是 1/4
* * * * * * * * * * * * * * * * * * * * * * * * * /
BOOL RabinMillerKnl(big n)
{
    if (size(n) <= 1) return FALSE;
    int i, j;
    big b=mirvar(0);
    big m=mirvar(0);
    big v=mirvar(0);
    big test1=mirvar(1);
    big test2=mirvar(2);
    decr(n, 1, m);   /* m=n-1 */
    j=0;
/* 1、先计算出 m、j使得 n-1=m*2^j,其中 m是正奇数, j是非负整数 */
    while(subdivisible(m, 2)) //m%2==0
    {
        ++j;
        subdiv(m, 2, m);  //求 m=m/2;
    }
/* 2、随机取一个 b, 2<=b<n-1 */
    irand(123);
    big NTEST=mirvar(0);
    decr(n, 3, NTEST);
    bigrand(NTEST, b);
    incr(b, 2, b);  //b=b+2;
/* 3、计算 v=b^m mod n */
    powmod(b, m, n, v);  //PowMod(b, m, n, v);
/* 4、如果 v==1,通过测试 */
    if(!compare(v, test1))  //比较 v==1
    {
        return TRUE;
    }
/* 5、令 i=1 */
    i=1;
/* 6、如果 v=n-1,通过测试 */
    decr(n, 1, NTEST);  //NTEST=n-1;
    while(compare(v, NTEST)) //v!=n-1
    {
/* 7、如果 i==j,非素数,结束 */
    if(i==j)
    {
```

```
            return FALSE;
        }
/*  8, v= v^2 mod n, i= i+ 1*  /
        powmod( v,  test2,  n, v);
        + + i;
/*  9,循环到 5*  /
        }
        mirkill( m);        //释放大数变量
        mirkill( b);
        mirkill( v);
        mirkill( test1);
        mirkill( test2);
        return TRUE;
        }
```

Rabin - Miller测试法是个使用广泛的简单算法,它基于 Gary Miller的部分想法,由 Michael Rabin发展。事实上,这是在 NIST的 DSS建议中推荐的算法的一个简化版。

但是,运行一次测试的判断结果当然不能满意,多运行几次随机测试,这样我们判断错的概率就变为,但是往往还要先通过一个小的素数表,进行一些筛选来提高效率。

程序算法:

1. 构造小素表数组;

2. 产生一个大数 n和循环次数 loop;

3. 判定大数 n是否能整除小素数表中的一个素数,如果能整除则说明不是大素数返回到第 2步;

4. 循环调用 Rabin - Miller素数测试核心函数 loop次全部通过返回 1,否则返回 0。

程序实现:

```
/* * * * * * * * * * * * * * * * * * * * *
* 数组原型: const static long g_PrimeTable
* 功能说明:小素数表
* 返 回 值:小素数的值
* 备    注:利用此表进行素数筛选,有利于提高判定大数是否为素数的效率。
* * * * * * * * * * * * * * * * * * * * * /
const static long g_PrimeTable[ ] =
{
        2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41,
        43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97
        };  //小素数表
/* * * * * * * * * * * * * * * * * * * * *
* 函数原型: long RabinMiller( big n,  long loop)
* 功能说明: Rabin - Miller素数测试,循环调用核心 loop次全部通过返回 1,否则返回 0
* 输入参数: n为大数, loop为循环次数
* 输出参数:无
* 返 回 值: 0不是素数; 1是素数。
* 备 注: 此函数利用了 Miracl大数运算库的加、减、乘、除、模逆
* * * * * * * * * * * * * * * * * * * * * /
long RabinMiller( big n,  long loop)
{
        //先用小素数筛选一次,提高效率
        for( long i= 0;  i <  g_PrimeCount;  i+ + )
        {
            if( subdivisible( n, g_PrimeTable[ i] ) )
            {
                return 0;
        }
            }
        //循环调用 Rabin - Miller测试 loop次,使得非素数通过测试的概率降为 ( 1/4) ^loop
        for( long i= 0;  i <  loop;  i+ + )
        {
            if(!  RabinMillerKnl( n) ) //调用判定素数函数
            {
                return 0;
            }
        } return 1;
}
```

算法效率分析:

本程序使用 Miracl大数运算库,随机发生器产生一个 512位大数,同时采用 Miracl大数运算库中的加、减、乘、除、模逆运算。算法效率分析采用调用 Pentium CPU内部时间戳进行计时。

```
/* * * * * * * * * * * * * * * * * * * * *
* 函数原型: inline unsigned __int64 GetCycleCount( )
* 功能说明:取 CPU的时间戳
* 输入参数:无
* 输出参数:无
* 返 回 值: CPU的时间戳的值
* 备 注: CPU的时间戳的值越大说明效率越低。
* * * * * * * * * * * * * * * * * * * * * /
inline unsigned __int64 GetCycleCount( )
{
        __asm _emit 0x0F
        __asm _emit 0x31
}
```

系统信息说明:

OS名称: Microsoft(R) Windows(R) Server 2003, Enterprise Edition
OS版本: 5.2.3790 Build 3790
OS制造商: Microsoft Corporation
OS配置: 独立服务器
OS构件类型: Multiprocessor Free
系统制造商: INTELR
系统型号: AWRDACPI
系统类型: X86-based PC
处理器: 安装了 2个处理器。
[1]: x86 Family 15 Model 3 Stepping 4 GenuineIntel ~ 2992Mhz
[02]: x86 Family 15 Model 3 Stepping 4 GenuineIntel ~ 2992 Mhz
BIOS版本: InteR - 42302e31
物理内存总量: 510 MB
可用的物理内存: 136 MB
程序性能分析:

程序使用此算法获得的 CPU 时间戳值 = 1931423732,本人同时使用 Miracl大数运算库的 isprime()方法查找 p和 q大素数的测试结果为 CPU时间戳值 = 2838214726

从 CPU时间戳值结果可以看出虽然 Miracl大数是商用大数库,但 isprime()方法并没有使用小素表进行素数的筛选,因而效率比较低。

2结论

通过以上内容,我们可以通过素数预处理来较快地判定素数.另外,对算法稍作改动,就可以利用该算法来进行其他类型的大素数的求解,同时还可以用于 RSA加密。由此可见,建立“概率素数"是切实可行的。

〔参 考 文 献 〕

[1]卢开澄.计算机密码学[J]. 北京:清华大学出版社, 2003, 242~246.
[2] 雷建云,余启港,蒋天发,陈哲.安全素数判定算法的实现[J].中南民藨学院学报(自然科学版) 1999-3(1).
[3] 韦萍萍,戎士奎.判定素数的新方法及程序[J].贵州教育学院学报(自然科学) 2005-4(2).
[4] 刘勇飞.与素数判定有关的三个命题[J].中等数学, 2005(9).
[5] 吴长海,孙宝林.素数测试在 RSA公开密钥密码算法中的分析研究[J].武汉交通科技大学学报 2000-8(4).
[6] 甘志国.用素数判定多 I页式不可约[J].中学数学, 2002(4).
[7] 朱玉扬.广义 Fermat数素性判定问题的几个结论[J].合肥学院学报(自然科学版) 2004-3(1).
[8] 贺毅朝,沈春璞,王立壮,徐绍珍. Rabin密码系统的分析与实现[J].河北省科学院学报 2002-11(4).
[9] 赵华杰.用统计思想说明素数定理[J].天津教育学院学报(自然科学版).
[10] 耿海飞,苏锦海.大素数的快速生成研究与实现[J].电脑与信息技术, 2005年 4月.

The Design and Practice of Prime Theorem

Xie Rimin

Abstract: This paper analyses carefully the test algorithm of prime number. Based on the MIRACL library, the author makes use of the Miller-Rabin Test Algorithm to realize the-Rabin Prime Number testing algorithm by applying Microsoft VC++.NET 2003 tool.

Key words: Prime Theorem, Fermat Theorem, Rabin-Mille Test Algorithm

责任编辑[梁小红]