数据加密算法与大素数的生成及运算

刘少涛, 凌 捷

(广东工业大学 计算机学院, 广东 广州 510090)

摘要: 数据加密算法 RSA 的关键在于大素数的生成, 本文采取链表结构解决大素数的存储和运算问题, 并给出了生成大素数的一种方法, 可生成超过 200 位的十进制数的大素数, 以应用于数据加密和数字签名.

关键词: 素数; RSA 算法; 模运算

中图分类号: TP309.7 文献标识码: A 文章编号: 1007-7162(2001)04-0025-05

数据加密技术是保证信息的安全性、完整性和确证性, 防止信息被篡改、伪造和假冒的主要手段之一. 计算机密码学是结合数学、计算机科学、电子与通信等诸多学科于一身的交叉学科, 随着计算技术的发展, 密码学也得到了空前的发展, 产生了纷杂繁多的数据加密算法. 在众多的加密算法中, RSA 公钥密码算法是一种公认的较安全的公钥密码算法, 被广泛应用于如信用卡数据加密及电子商务数字签名等领域, 是目前最流行的算法之一. RSA 算法的公开密钥和私有密钥是一对大素数(超过 100 位的十进制数)的函数, RSA 算法的安全性就是基于数论中大素数分解的困难性, 即从一个公开密钥和密文中破译出明文的难度等价于分解两个大素数之积. RSA 算法的发展离不开对大素数的研究, 算法的关键在于大素数的生成以及大素数在计算机里的存储和运算速度. 本文采用链表的形式对大素数进行存储和运算, 并开发了可以生成超过 200 位的十进制数的大素数的程序, 可应用于 RSA 加密和解密算法.

1 RSA 算法与大素数

1.1 RSA 算法原理

RSA 算法的命名来源于它的三个创始人: Rivest、Shamir 和 Adelman. RSA 加密算法可分为三个过程:

1) 选取加密密钥和解密密钥

首先要选取两个大素数: p 和 q(均为超过 100 位的十进制数), 并计算其乘积 $n = pq$, 再随机选取加密密钥——正整数 e, 使得 e 和 $(p-1)(q-1)$ 互素. 然后用欧几里德扩展算法计算解密密钥 d, 以满足:

$$ed = 1 \bmod ((p-1)(q-1)).$$

注意: d 和 n 也互素. e 和 n 是公开密钥, d 是私人密钥. 两个素数 p 和 q 不再需要, 它们应该被舍弃, 但绝不可泄露.

收稿日期: 2001-04-11

资助项目: 广东省自然科学基金项目(994385), 教育厅"千百十"优秀人才基金项目(200017).

作者简介: 刘少涛(1974-), 男, 硕士研究生, 主要研究方向为密码技术, 图像处理, 算法理论.

2) 加密过程

加密明文信息 m 时, 首先将它分成比 n 小的数据分组(采用二进制数, 选取小于 n 的 2 的最大次幂), 即 $m = m_1 m_2 \cdots\cdots m_k$. 如果你需要加密固定的消息分组, 那么可以在它的左边填充一些 0 并确保该数比 n 小. 加密后生成的密文 c, 将由相同长度的分组组成, 即 $c = c_1 c_2 \cdots\cdots c_k$. 加密公式简化为:

$$c_i = m_i^e (\mathrm{mod}\ n).$$

3) 解密过程

解密过程是加密过程的逆运算, 首先取每一个加密后的分组 c_i 并计算相应的 m_i:

$$m_i = c_i^d (\mathrm{mod}\ n).$$

然后对每个 m_i($i = 1, 2, \cdots\cdots, k$) 按次序恢复全部明文信息 m.

RSA 算法实现的技术关键在于: (1) 寻找大素数 p 和 q, 为了提高安全性, 要求 p 和 q 应选择安全素数和强素数, 通常用概率算法来产生; (2) 求解解密密钥 d, 对 d 的限制是 $\gcd(d, (p-1)(q-1)) = 1$, 一般取大于 p 和 q、且小于 $(p-1)(q-1)$ 的素数; (3) 需要进行大量的大指数模 n 运算.

RSA 算法在世界上许多地方已成为事实上的加密标准, 并被广泛应用于数据签名技术. 随着大整数分解算法和计算能力的不断提高, 对 RSA 算法的破译能力也在增强. 所以安全的 RSA 算法需要采用足够大的素数, 素数的位数越大时, 因子分解越困难, 密文信息就越难以破译, 加密强度就越高. 2000 年 12 月, 日本东芝情报系统公司和 NTT 电子公司、东洋情报系统公司(TIS) 3 家企业联合推出了使用公开密钥加密方式的认证系统“Castation@ 白山”, 该系统的密码算法采用了 RSA 方式, 公钥的长度为 2048bit(相当于 600 多位的十进制数), 在目前的技术水平和计算能力下, 对 n 进行因数分解实际上是无法实现的.

1.2 素数的生成

n 位的十进制位的大素数生成步骤如下:

1) 产生一个 n 位的随机数 p.

2) 若最高位为 0, 则将它置为 1, 以确保该素数达到所要求的长度.

3) 若最低位为偶数, 则将它加 1, 以确保该素数为奇数, 从而保证了平均节省一半的运算时间.

4) 检查以确保 p 不能被任何小素数整除, 如 3, 5, 7, 11 等等. 目的是排除 p 是合数的绝大部分可能性, 减少了下面步骤对 p 进行素数测试的总次数, 从而大大节省了运算时间. 我采用了除 2 以外(因已经检验过 p 的奇偶性)的所有小于 10000 的素数(共 1228 个), 来测试 p 对于它们的整除性.

5) 产生一个随机数 a, 对 a 进行 Rabin-Miller 测试. 如果 p 通过测试, 则另外产生一个随机数 a, 再重新进行测试. 选取较小的 a 值, 以保证较快的运算速度. 作 5 次 Rabin-Miller 测试(1 次看起来已足够, 但为保证较高的精确性, 可以多做几次).

6) 如果 p 通过测试, 则 p 是素数. 否则将原被测试数加 2, 得到一个新的数, 再对新数进行测试, 直到找到一个素数为止.

1.3 素数的证明

对于素数的生成和证明, 目前流行着多种多样的算法, 如 Solovag-Strassen 算法、Lehmann 算法、Rabin-Miller 算法[1] 等. Solovag-Strassen 算法是一种基于概率的基本测试算法, 它使用雅可

比函数来测试一个整数是否素数, 当通过 t 次测试后, 被测数是合数的可能性不超过 $1/2^t$. Lehmann 算法是一种更简单的测试, 和 Solovag- Strassen 算法一样, 利用 Lehmann 算法对被测数进行 t 次测试后, 被测数可能是素数所冒的错误风险也不超过 $1/2^t$. Rabin- Miller 算法是一个容易实现且被广泛使用的算法, 它基于 Gary Miller 的部分想法, 而由 Michael Rabin 发展完善. 和前两种算法比较起来, Rabin- Miller 算法的运算速度更快, 且通过测试后被测数是素数的准确率更高(通过 t 次测试后, 被测数是合数的可能性不超过 $1/4^t$).

定义 1　令 $n-1=2^l m$, 其中 l 是非负整数, m 是正奇数. 若 $b^m \equiv 1 \pmod d$ 或 $b^{2^j m} \equiv -1 \pmod d$, $0 \leqslant j \leqslant l-1$, 则称 n 通过以 b 为基的 Rabin- Miller 测试[2].

定理 1　如果 n 是奇合数, 那么 n 通过以 b 为基的 Rabin- Miller 测试的数目最多为 $(n-1)/4$, $1 \leqslant b \leqslant n-1$.

Rabin- Miller 测试的算法思想如下:

首先选择一个待测的随机数 p, 计算 b, b 是 2 整除 $p-1$ 的次数(即, 2^b 是能整除 $p-1$ 的 2 的最大幂数). 然后计算 m, 使得 $n=1+2^b m$.

1) 选择一个小于 p 的随机数 a(即 $1 \leqslant a \leqslant p-1$).

2) 设 $j=0$ 且 $z=a^m \bmod p$.

3) 如果 $z=1$ 或 $z=p-1$, 那么 p 通过测试, 可能是素数.

4) 如果 $j>0$ 且 $z=1$, 那么 p 不是素数.

5) 设 $j=j+1$. 如果 $j<b$ 且 $z \neq p-1$, 设 $z=z^2 \bmod p$, 然后回到第 4 步. 如果 $z=p-1$, 那么 p 通过测试, 可能是素数.

6) 如果 $j=b$ 且 $z \neq p-1$, 那么 p 不是素数.

2　数据结构及实现

2.1　数据结构

加密算法中所用到的大素数一般都要求是 100 位以上的十进制整数, 对于某些要求安全性很高的加密算法, 大素数甚至要求是 512 位以上的十进制整数. 各种高级语言中一般的整数数据类型是远远不能满足该要求的. 本文采用双向链表[3] 的方法, 对大素数进行分解, 并把每部分存到各个结点中, 就能对超过 200 位的十进制整数进行存储和运算.

本文所用链表数据结构如下:

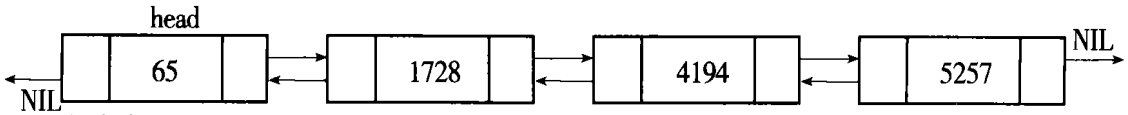
```
Type
  TGInt= ^cont;
  cont = Record
       value : integer;
       next, prev : TGInt;
  End;
```

结点结构为:

prev	value	next

每一个结点的 value 为一个整数类型数据, 以 Delphi 为例, 一个整数类型变量可存放 1~ 9 位的十进制随机数, 由于涉及到两个整数类型数据的乘法, 我们采用每个结点的 value 存放一个四

位的十进制整数, 故若一个 100 位的十进制整数, 其链表结构包括 25 个的结点, 同样, 一个 512 位的整数, 其链表结构包括 128 个的结点 . 例如, 对于一个 14 位的整数 65172841945257, 用链表存储可以表示为:



2.2 程序实现

2.2.1 关于大素数的大指数模运算

在 RSA 算法的加密过程和解密过程以及素数的 Rabin- Miller 测试过程中, 经常出现对于两个大素数 a 和 b, 来计算 $a^b \bmod n$ 的情况 . 这时我们当然不能直接进行 b 个 a 的乘法, 然后再和一个大数的模化简:

$$\underbrace{(a \times a \times \cdots\cdots \times a)}_{n\text{个}a} \bmod n,$$

如果这样运算, 运算速度将会令人不堪忍受. 为了提高运算效率, 可采用模运算的分配律 . 比如 $a^{25} \bmod n$ (为了便于说明, 在这里 b 取小整数), 由于 25 的二进制为 11001, 即 $25 = 2^4 + 2^3 + 2^0$. 故:

$$a^{25} \bmod n = (a \times a^{24}) \bmod n = (a \times a^8 \times a^{16}) \bmod n =$$
$$(a \times ((a^2)^2)^2 \times ((a^2)^2)^2)^2) \bmod n = ((((a^2 \times a)^2)^2)^2 \times a) \bmod n =$$
$$((((((a^2 \bmod n) \times a) \bmod n)^2 \bmod n)^2 \bmod n) \bmod n) \times a) \bmod n ,$$

这种算法称为加法链(addition chaining)或二进制序列的乘法方法. 当 b 为 100 位以上的大素数时, 同样也可利用这种方法来求 $a^8 \bmod n$.

具体实现的程序代码如下:

```
Procedure GIntModExp (GInt, exp, modn : TGInt; Var modrest : TGInt);
Var
    temp, GIntprod: TGInt;
    S : string;
    i : integer;
Begin
    GIntToBinStr(exp, S);                    // 将 exp 转化为一个二进制字符串.
    If S[length(S)] = '0' Then
      DecStrToGInt('1', modrest)             // 将'1'转化为 TGInt 型并赋予 modrest.
    Else
      GIntcopy(GInt, modrest);               // 将 GInt 拷贝给 modrest.
    GIntcopy(GInt, temp);                    // 将 GInt 拷贝给 temp.
    For i := (length(S) - 1) Downto 1 Do
    Begin
      GIntMul (GInt, GInt, temp);            // temp= GInt^2.
      GIntMod(temp, modn, temp);             // temp= temp mod modn.
      If S[i] = '1' Then
      Begin
        GIntMul(modrest, temp, GIntprod);    // GIntprod= modrest × temp
```

```
            GIntMod(GIntprod, modn, modrest);      // modrest = GIntprod mod modn
          End;
        End;
        GIntDestroy(temp);      // 删除链表,已释放内存.
        GIntDestroy(GIntprod);  // 删除链表,已释放内存.
End;
```

2.2.2　大素数

利用本文程序可以生成超过 200 位的十进制大素数, 并可以通过数学软件 MATHEMATICA 中的 PrimeQ( )函数进行验证, 下面给出本文程序生成的一个 200 位的十进制大素数 p 和一个 256 位的十进制大素数 q:

p = {1410 3296 4850 5661 4642 5289 5211 0656 1583 9861 7535 0333 3924 9638 8337 1700 6351 3834 0084 6556 2716 3464 4126 0539 3135 6214 1924 3628 8362 7686 8158 0063 1281 8243 1818 8160 7868 7164 3859 2996 2446 8337 2534 7407 4605 4198 9975 9304 7606 7203},

q = {9205 9102 1564 2891 7775 1774 3844 7878 5520 8929 3578 5462 3672 9126 3100 2560 0528 9893 7769 2022 8138 8037 1874 4051 8810 9025 9907 4438 5639 6525 8666 6945 3796 5714 8154 9680 1782 6434 1746 7808 4379 4804 8596 7455 1499 4357 3743 8121 9668 0332 3896 3475 5354 3176 9359 2857 1855 9880 5068 8338 3979 4229 6208 4713}.

3　结束语

本文利用链表结构对大素数进行存储和运算, 克服了大素数不易操作的困难, 可以生成超过 200 位的十进制大素数(但随着位数的增加, 其运算速度也将变慢). 同时它也采用了面向对象的方法, 把大素数的基本运算打包成相应的基本函数和过程块, 以方便用户的操作和使用. 本文方法对于信息加密、数字签名以及信息的安全性、机密性和完整性, 防止信息被篡改、伪造和假冒等具有重要意义.

参考文献:

[1] Bruce Schneier. Applied Cryptography(Second Edition)[M]. John Wiley & Sons, Inc. 1996.
[2] 卢开澄. 计算机密码学(第二版)[M]. 北京: 清华大学出版社, 1998.
[3] 严蔚敏, 吴伟民. 数据结构(C 语言版)[M]. 北京: 清华大学出版社, 1997.

Data Encryption Algorithm and The Generation and Operation of Great Primes

LIU Shao-tao, LING Jie
(Faculty of Computer, GDUT, Guangzhou 510090, China)

Abstract: This paper resolves the storage and operation of great primes by linked list, and discusses a way to generate great primes which are over 200 decimal bits and can be used in data encryptions and digital signatures.

Key words: prime; RSA; modular operation