**成绩：**

RSA快速算法及C++实现

马颖涵

摘要：RSA算法是当今流行的加密算法，目前广泛运用于各类安全领域。本文介绍了RSA的加解密算法，并对算法中的求逆和求幂的算法进行了分析，详细介绍了利用扩展欧几里得算法求逆和利用传统BR算法快速求幂，并在传统BR算法的基础上进行改进从而加快求幂的计算速率，最后利用高精度算法对RSA快速算法进行了C++实现。

关键词：RSA；扩展欧几里得；BR算法；C++

The RSA Algorithm with C++

Yinghan Ma

Abstract: The RSA algorithm is a popular encryption algorithm, which is widely used in various security fields. In this paper I introduce the RSA encryption and decryption algorithms, and analyze the algorithm of inverse ans power. Then I introduce the extended Euclidean algorithm for inverse and the traditional BR algorithm for power in detail, and improve the BR algorithm to speed up the calculation. Finally, I implement the RSA algorithm with a hands-on C++ coding project。

Keywords: RSA; the extended Euclidean algorithm; BR algorithm; C++

0 引言

现代通信离不开对信息的加密，加密算法在许多安全应用中有至关重要的作用。然而加密技术的安全性通常依赖于加密时所使用的密钥而非加密方法本身[1]。传统对称密码需要在在通信双方都知道密钥的前提下进行消息传递，消息安全性无法得到很好的保证，且大量密钥的产生也使密钥管理成为一大难题。因此1976年美国斯坦福大学学生W. Diffie和他的导师M. hellman提出可以将发送方的密钥信息公开，从而消除安全密钥分发通道的需要，第一个公开密钥密码的基本概念被提出[2]。

RSA算法是罗纳德·李维斯特（Ron Rivest）、阿迪·萨莫尔（Adi Shamir）和伦纳德·阿德曼（Leonard Adleman）于1978年提出的一种公钥密码体制[3]。RSA是目前最具有影响力的公钥加密算法，能够抵抗到目前为止已知的绝大多数密码的攻击。RSA的安全性建立在现代计算机对于大合数因子分解的困难性[4]，因此RSA密钥的长度一般都很长，从而造成其加解密时的计算量大。实现RSA算法面临的主要问题就是大数模幂乘算法的效率[5]。本文介绍了RSA加密算法的计算过程，分析了当前对于大数求幂所使用的传统算法并加以改进，最后完成了RSA算法基于高精度的C++实现。

1 RSA加解密算法

RSA是利用欧拉函数、质数和因子分解实现安全数据传输的一种公钥加密密码系统[6]。

RSA加解密算法的过程如下：

1. 随机选择两个大素数p和q，将p和q保密。
2. 计算n=pq，将n公开。
3. 计算，将保密。
4. 随机选取一个整数e，满足1 < e < 且 gcd( e , ) = 1 ，将e公开。
5. 由 得到d，将d保密。
6. 加密运算：
7. 解密运算：

可见，RSA加解密的运算均依赖于大数的模幂乘运算，因此，只要在运算时选取的大素数足够大，RSA算法的安全性就能够得到一定的保证。

2 计算方法

由于RSA算法的运算都是基于大数的运算，运算量大，算法的效率成了一大难题。RSA算法中主要用到的运算是求乘法逆元和幂运算，接下来将介绍利用扩展欧几里得算法求逆元、传统的快速幂求法以及基于传统快速幂求法的一些改进方法。

2.1 利用扩展欧几里得算法求逆元

扩展欧几里得算法是基于欧几里得算法，即用辗转相除法求最大公约数，如下公式[7]：

存在不全为零的整数x , y使得，当 ，即ax+by=1时，x为a关于模b的乘法逆元。利用扩展欧几里得算法求逆元，以 121 为例：

53；

；

；

；

；

所以，且

(1)

(2)

(3)

(4)

(5)

得出时满足，即。

将（5）～（1）写成：

令 ，则。

由此可写出扩展欧几里得求逆元的C++代码：

int exgcd(int a,int b,int &x,int &y)

{

    if(b==0)

    {

        x=1;y=0;

        return a;

    }

    int r=exgcd(b,a%b,x,y);

    int temp=y;

    y=x-(a/b)\*y;

    x=temp;

    return r;   //得到a b的最大公因数

}

如果最后返回的值为1，则，且最后得出的x是a关于模b的乘法逆元。

2.2 传统快速求幂的方法

传统求幂的快速幂算法是二元算法，即BR（Binary representations）算法[8]。BR算法通过将指数表示成二进制来实现快速幂，即：

如。

由此可写出BR算法的快速幂C++代码如下：

int power(int a,int b){

    int c=1;

    while(b>0){

        if(b%2==1){

            c\*=a;

        }

        a\*=a;

        b/=2;

    }

    return c;

}

最后的返回值为。

2.3 基于传统BR算法的改进

上述的BR算法的时间复杂度为。由于二进制数长度较大，计算时需要多次迭代，可以通过减少迭代次数来优化该算法[9]。传统的二元算法将指数表示成二进制，依据传统算法提出先将指数进行十进制的拆分运算，即：

也可表示为当b>=10时：

即：

在对指数进行十进制处理后对进行传统二元算法的快速幂计算，则此时算法的时间复杂度变为  
  
当b足够大时改进后算法的运算时间远小于传统二元算法。

上述改进算法的C++实现如下：

int power\_dfs(int a,int b){

    if(b<10){

//若b<10，则直接返回二元算法的运算结果

        return power(a,b);

    }

//若b>=10，则对b进行十进制拆分

   return power(power\_dfs(a,b/10),10) \*power(a,b%10);

}

3 RSA算法的C++实现

3.1 大数的表示

由于C++的整数表示范围有限，而用于RSA的大数往往无法用C++的预定义的整数或长整数类型表示，因此需要通过自定义类型来对大数进行存储。在高精度算法中通常在自定义类型中使用整型数组p来储存大数每一位数的值，用一个整数len来存储大数的长度。

3.2 大数的基本运算

在进行RSA算法的运算中需要运用到的基本运算有加、减、乘、除、模，由于大数的存储类型是自定义的，所以在对大数进行基本运算前需要对运算符号进行重载，其中包括大数与大数的加法、大数与整数的加法、大数与大数的减法、大数与整数的减法、大数与大数的乘法、大数与整数的乘法、大数与大数的除法、大数对大数的模，在对大数与大数的减法重载时还涉及到了大数大小的比较。由于大数的加减和大小的比较与常规的计算方法没有太大差别，所以接下来只对大数的乘除模进行介绍。

3.2.1 大数的乘法运算

大数与整数的乘法与常规乘法相似，从末位开始逐一乘以整数乘数，若有进位则向前进位。而大数与大数的乘法是基于大数与整数的乘法，与2.3所描述的快速幂方法有相似之处，即当b>=10时将b分解：

则此时

由于b%10<10，所以部分进行的是大数与整数相乘的运算。

大数与大数的乘法的重载函数代码如下：

HP operator \*(const HP &a,const HP &b){

    if(b.len==0) return b;//若b==0则a\*b=0

    if(b.len==1){

        HP aa=a;

        if(a.b!=b.b) aa.b=-1;

        return aa\*b.p[1];

    }//若b为个位数，则返回大数乘整数

    HP c;

    c.len=b.len-1;

    for(int i=c.len;i>0;--i){

        c.p[i]=b.p[i+1];

    }//c=b/10

    return a\*c\*10+a\*b.p[1];

}

3.2.2 大数的除模运算

大数与大数的除法和模运算均基于大数与大数的减法，即当a>=b则，a<b时停止，循环次数则为a/b的结果，循环结束后a的值即为a%b的结果。由于该算法的时间复杂度取决于a/b的值，所以若a和b相差过大，其时间复杂度也会很大，计算除法和模时就要在循环体内浪费很多的时间。于是为了让算法在循环体内的时间充分小，令t为a的长度与b的长度的差，即 ，若t>0则令并对b进行加倍处理，令，再进行循环相减操作，则该算法最多只会进行99次循环。而最终得到的a的值还是a%b的结果，而a/b的结果则为循环次数\*d。

以大数与大数的除法为例展示上述算法的C++代码如下：

HP operator / (const HP &a,const HP &b){

    HP c;

    c.len = a.len;

    HP d;//统计总循环次数

    for(int i=a.len;i>0;--i){

        c.p[i]=a.p[i];

    }//将a的值赋给c

    while(c>b){

        int t=c.len-b.len;

        if(t>0){//如果c和b有位数差则采用上述算法，反之则直接相减

            HP g,gg;

            g.len=t;

            g.p[t]=1;//令g=pow(10,t-1)

            gg=g\*b;

            int k=0;//统计加倍后的循环次数

            while(c>gg){

                c=c-gg;

                ++k;

            }

            d=d+g\*k;//总循环次数应加上 加倍后的循环次数\*倍数

        }

        else{

            c=c-b;

            d=d+1;//总循环次数加一

        }

    }

    return d;

}

3.2 C++实现RSA算法的完整代码

C++实现RSA算法的完整代码见网址： <https://github.com/myhbefine/cryptography/blob/0545fcbfe064ce14c1bfa7753be4fc796eb35e96/RSA.cpp>

4 讨论

出于人们对RSA软件加解密速度的需求，各种基于RSA密码的快速算法层出不穷[10]。RSA算法的简单和易于实现使得RSA密码的实现被广泛研究[11][12][13]。

RSA算法看似容易破译，但要能够实现大合数的因子分解是十分困难的。由于RSA密码对当今计算机的大合数因子分解的计算的依赖，使用RSA密码时需密切关注大合数因子分解的研究进展[14]。目前RSA密码已被广泛应用于各类安全或认证领域[15]。随着世界科学研究的进步，人们对大合数分解的能力在不断提高。若未来量子计算机发展成熟，当今使用的大合数的因子分解将轻而易举，因此RSA密码使用的大素数的位数需要进一步增加以确保RSA密码的安全。

参考文献

1. 陈洁. 从RSA密码体制看现代密码学[J]. 中国教育技术装备,2005(9):47-48. DOI:10.3969/j.issn.1671-489X.2005.09.019.
2. Whitfield Diffie and Martin E. Hellman. 1976. Multiuser cryptographic techniques. In <i>Proceedings of the June 7-10, 1976, national computer conference and exposition</i> (<i>AFIPS '76</i>). Association for Computing Machinery, New York, NY, USA, 109–112. DOI:https://doi.org/10.1145/1499799.1499815
3. Rivest, R, Shamir, A, & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126. doi:10.1145/359340.359342
4. Yang, H. (2014). How secure is RSA algorithm? Retrieve from Dr. Herong Yang’s website: <http://www.herongyang.com/Cryptography/RSA-Algorithm-How-Secure-Is-RSA-Algo[[]]rithm.html>
5. 李强,张继永. 一种改进的RSA快速算法[J]. 小型微型计算机系统,2001,22(1):70-72. DOI:10.3969/j.issn.1000-1220.2001.01.018.
6. Mathonline. (2014). RSA Encryption. Retrieved from the Wikidot.com website:http://mathonline.wikidot.com/rsa-encryption
7. 汪杨海,贺细平. 扩展欧几里德算法改进探讨[J]. 电脑与信息技术,2018,26(6):12-14. DOI:10.3969/j.issn.1005-1228.2018.06.004.
8. 陈运,龚耀寰. RSA快速算法研究[J]. 通信保密,2000(3):43-46. DOI:10.3969/j.issn.1009-8054.2000.03.012.
9. 谢慧. 网络通信中一种数据保密性和完整性的方案研究[D]. 湖南大学,2009. DOI:10.7666/d.y1724268.
10. 贺令亚. RSA加密算法的研究与实现[D]. 中南大学,2009. DOI:10.7666/d.y1536734.
11. Kaliski, B., and Robshaw, M. The Secure Use of RSA. CryptoBytes, RSA Laboratories, (Autumn 1995), 7-13.]]
12. Koç, Ç. K. High-Speed RSA Implementation. Technical Report TR-201, version 2.0, RSA Laboratories, November 1994.]]
13. Rivest, R. L. Response to NIST's proposal. Communications of ACM, 35, 1992, 41-47.]]
14. 张焕国,唐明.密码学引论（第三版）[M].武汉：武汉大学出版社，2020.01，第214页.
15. 张淑芬,陈学斌,刘春风. RSA公钥密码体制的安全性分析及其算法实现[J]. 计算机应用与软件,2005,22(7):108-110. DOI:10.3969/j.issn.1000-386X.2005.07.045.

查重结果截图：

