模逆算法的分析、改进及测试

谭丽娟，陈 运

(电子科技大学通信与信息工程学院 成都 610054)

【摘要】公钥密码实现中，模逆算法经常是算法实现的瓶颈。通常求模逆的运算方法牵涉到大量的除法和减法操作，而除法操作需要大量的运算开销。基于现有的求最大公因子的方法，分析利用扩展欧几里德求模逆的方法，以及二进制扩展欧几里德算法，提出了利用二进制扩展欧几里德算法求模逆的方法，给出了几种算法性能比较的测试环境和测试结果。测试结果表明：改进的算法比利用扩展欧几里德求模逆的方法速度更快，对硬件实现更具有普遍性。

关 键 词 公钥密码体制； 欧几里德算法； 扩展欧几里德算法； 模逆

中图分类号 TP309.7 文献标识码 A

Analysis and Improvement of Modular Inverse Algorithm

Tan Lijuan，Chen Yun

(School of Communication and Information Engineering, UEST of China Chengdu 610054)

Abstract Usually modular inverse operation becomes bottlenecked in realizing the public key cryptosystem. The method commonly used leads to a lot of division operation. The modular inverse operation by extended Euclidean algorithm and the binary extended Euclidean algorithm are analyzed, and the improved modular inverse operation by binary extended Euclidean algorithm is presented in this paper. The result shows that the new algorithm runs faster than the old one under the given testing environment. Furthermore it has much more feasibility in hardware realization.

Key words public key cryptosystem; Euclidean; extended Euclidean; modular inverse

假设u为模数，x为系数，r为余数，满足$x\times v=r \bmod u$。当$r=1$时，称x为v模u的逆，即$x=v^{-1} \bmod u$；否则，即当$r\neq 1$时，称x为v模u的系数。

模逆运算在公开密钥加密算法和数字签名算法中占有重要的位置，如Massey-Omura公钥体制和ElGamal公钥体制中都要用到。但是，模逆运算是模运算中效率最低的，因此，模逆运算的运行速度会直接影响到整个密码体制的执行效率。

1 现有算法基础

欧几里德算法获得了广泛的应用，可以用来求两数的最大公因数,也可以用来求模逆。但是这种算法却不是最好的算法，因为算法中每步都要进行除法操作，开销很大。大约是1962，Roland Silver 和John Terzian发现了一种二进制算法，但并没有公开算法。到了1967年J.Stein公开了这种算法，该算法也称为Stein算法[1]。

Stein算法中没有除法指令，只有减法和对二进制数的移位操作。算法基于以下几个性质改进了欧几里

收稿日期：2003 – 02 – 20

作者简介：谭丽娟(1978 –)，女，硕士，主要从事信息安全方面的研究.

德算法。其中$\gcd(u,v)$表示u和v的最大公因数。$\max(u,v)$表示u和v两个数中大的一个。

$$\gcd(u,v)w=\gcd(uw,vw) \tag{1}$$

$$\gcd(u,v)=\gcd(u/2,v) \tag{2}$$

式中 v不能被2整除。

$$\gcd(u,v)=\gcd(u-v,v) \tag{3}$$

式中 如果u，v都是偶数，则$(u-v)$也是偶数，而且

$$(u-v)<\max(u,v) \tag{4}$$

算法1 Stein算法

A_1-1 $c=0$；

A_1-2 if $u=0$ $g=\gcd(u,v)=2^c\times v$；

A_1-3 if (u and v are even) $u=u/2$; $v=v/2$; $c=c+1$；

A_1-4 if v is even swap(u,v);/*交换u、v的值*/；

A_1-5 if u is even $u=u/2$;

A_1-6 if $u-v<0$ wap(u,v);

A_1-7 if $u-v>0$ $u=u-v$;

A_1-8 goto A_1-2。

Stein算法中的除2操作在二进制运算中仅作移位操作，没有用到除法，只用到减法操作。减法比起移位要稍微花更长的时间。假设$u\geq v>0$，那么用这个方法求$\gcd(u,v)$所需的减法次数$d_1=\log_2 u+(\log_2 3-1)\leq(\log_2 u)+1$，所以减法的复杂度为$O(\log_2 u)$。

算法2 利用扩展欧几里德求模逆[2]

A_2-1 $(X_1,X_2,X_3)=(1,0,v)$;

A_2-2 $(Y_1,Y_2,Y_3)=(0,1,u)$;

A_2-3 if $Y_3=0$ return $X_3=\gcd(u,v)$; /*无逆元*/

A_2-4 if $Y_3=1$ return $X_3=\gcd(u,v)$; $Y_2=u^{-1}\bmod v$；

A_2-5 $Q=\lfloor X_3/Y_3\rfloor$; /*对除法的结果进行向下取整，如$\lfloor 1.11/1.99\rfloor=1$*/；

A_2-6 $(T_1,T_2,T_3)=(X_1-QY_1, X_2-QY_2, X_3-QY_3)$;

A_2-7 $(X_1,X_2,X_3)=(Y_1,Y_2,Y_3)$;

A_2-8 $(Y_1,Y_2,Y_3)=(T_1,T_2,T_3)$;

A_2-9 goto A_2-3。

计算过程中的X，Y，T始终满足关系式

$$v\times X_1+u\times X_2=X_3，\ v\times Y_1+u\times Y_2=Y_3，\ v\times T_1+u\times T_2=T_3 \tag{5}$$

式中 X，Y的关系可以通过归纳法证明，对于T

$$(v\times X_1+u\times X_2)-Q(v\times Y_1+u\times Y_2)=v\times(X_1-Q\times Y_1)+u\times(X_2-Q\times Y_2)=v\times T_1+u\times T_2=X_3-Q\times Y_3=T_3。$$

当$Y_3=1$时，$v\times Y_1+u\times Y_2=Y_3=1$，则$Y_1=v^{-1}\bmod u$和$Y_2=u^{-1}\bmod v$。

扩展欧几里德算法的主要开销是来自除法运算的，假设$u>v>0$，那么用这种方法求模逆所用到除法次数与用欧几里德方法求$\gcd(u,v)$的除法次数是相同的，$d_2<2\lceil\log_2 u\rceil$，所以这种算法用到的除法运算的复杂度就是$O(\lceil\log_2 u\rceil)$。

2 算法改进及分析

利用扩展欧几里德求模逆的算法中，X，Y，T始终满足式(5)，为了利用Stein算法来进行模逆运算，也要试图保证这个性质。Stein算法中主要利用了式(2)和(3)，将欧几里德算法中的除法指令转换成了减法和除2，而除2操作实质上在二进制运算中仅作移位操作，这种变换有利于软，硬件实现，而且指令时间无疑要比除法操作短。

基于上面的分析，可以将利用扩展欧几里德求模逆的算法2进行改进，如果X_3，Y_3为偶数，则可利用关

系式(2)对X_3，Y_3进行移位操作，即$X_3=X_3/2$，$Y_3=Y_3/2$。为了满足式(5)，则要对X_1、X_2、Y_1、Y_2作相应变换，如果X_1、X_2、Y_1、Y_2均是偶数，则同时对X_1、X_2、Y_1、Y_2都进行移位操作，即$X_1=X_1/2$，$X_2=X_2/2$，$Y_1=Y_1/2$，$Y_2=Y_2/2$，这样仍然满足式(5)。但是，如果X_1，X_2，Y_1，Y_2不全为偶数，就不能对它们作简单的移位操作了，因为：$v\times(X_1+u)/2+u\times(X_2-v)/2=(v\times X_1+u\times X_2)/2+(v\times u-u\times v)/2=X_3/2$，类似的：$v\times(Y_1+u)/2+u\times(Y_2-v)/2=(v\times Y_1+u\times Y_2)/2+(v\times u-u\times v)/2=Y_3/2$。因此，对$X_3$、$Y_3$奇偶判断之后就要对$X_1$，$X_2$，$Y_1$和$Y_2$作以上操作。

如果X_3和Y_3都不为偶数时，就要类似算法2中，作减法操作。但是X_3和Y_3改变后，可能不会像算法2中描述的一样，总是满足$X_3\geq Y_3$。虽然可以参照算法1，对X_3，Y_3的大小做一个判断之后，进行交换，使得X_3和Y_3 满足$X_3\geq Y_3$，这样虽然使程序思路清晰，却增加了算法的冗余度，因此，算法改进时可以省略交换的步骤。因为：$v\times(X_1-Y_1)+u\times(X_2-Y_2)=(v\times X_1+u\times X_2)-(v\times Y_1-u\times Y_2)=X_3-Y_3$，类似的：$v\times(Y_1-X_1)+u\times(Y_2-X_2)=(v\times X_1+u\times X_2)-(v\times Y_1-u\times Y_2)=Y_3-X_3$。因此，判断$X_3$，$Y_3$的大小之后，就可以直接根据情况做减法操作。

算法2中，循环终止的条件是$Y_3=0$，其目的是为了能求出$\gcd(u,v)$。但如果是已知两数互素，而只需求其中一个数对另一个数的模逆时，就可以更改判断条件，省去不必要的操作了。因为：$v\times X_1+u\times X_2=X_3=1$，类似的：$v\times Y_1+u\times Y_2=Y_3=1$则$X_1$、$Y_1$和$X_2$、$Y_2$分别是$v$模$u$和$u$模$v$的逆。对于以上的改进，可以得到算法3。

算法3　利用Stein算法改进的求模逆的算法

A_3-1　$(X_1,X_2,X_3)=(1,0,v)$；

A_3-2　$(Y_1,Y_2,Y_3)=(0,1,u)$；

A_3-3　while ($X_3\neq1$ 或 $Y_3\neq1$)；

A_3-3-1	if (X_3 is even)	if(X_1 X_2 are even)	$(X_1,X_2,X_3)=(X_1/2,X_2/2,X_3/2)$;
		else	$X_1=X_1+u$; $X_2=X_2-v$;
A_3-3-2	else if (Y_3 is even)	if(Y_1 Y_2 are even)	$(Y_1,Y_2,Y_3)=(Y_1/2,Y_2/2,Y_3/2)$;
		else	$Y_1=Y_1+u$; $Y_2=Y_2-v$;
A_3-3-3	else if ($X_3>Y_3$)	$(X_1,X_2,X_3)=(X_1-Y_1, X_2-Y_2, X_3-Y_3)$;	
	else	$(Y_1,Y_2,Y_3)=(Y_1-X_1, Y_2-X_2, Y_3-X_3)$;	
A_3-4	if ($X_3=1$)	return $X_1=v^{-1}\bmod u$;return$X_2=u^{-1}\bmod v$;	
	else	return $Y_1=v^{-1}\bmod u$;return$Y_2=u^{-1}\bmod v$ 。	

算法3比起算法1，增加了4个临时变量，因此算法3中的减法开销是要比算法1中的大。算法3中的A_3-3-3需要执行3次减法操作。所以，算法3中的减法次数可以记为 $d_3=3d_1+A$，其中 d_1 为算法1中的减法次数，A为A_3-3-1和A_3-3-2时的减法操作次数。

算法3既可用来求模逆，还可以用来求模系数。用来求模系数时，循环中止的条件可以改成$X_3\neq r$ 或 $Y_3\neq r$(r为余数)。实际应用的时候往往只需要求其中的一个数对另一个数的模逆，这样对X_1(或Y_1)或X_2(或Y_2)的操作就是多余的了。以求模u的逆为例，分析上面的情况。由于u、v互素，且$v\times X_1+u\times X_2=X_3$，所以，当$X_3$为偶数时，可以分成以下几种情况讨论：

1) v奇数，u奇数，X_1偶数，X_2偶数：$v\times(X_1/2)+u\times(X_2/2)=(v\times X_1+u\times X_2)/2=X_3/2$；

2) v奇数，u奇数，X_1奇数，X_2奇数：$v\times(X_1+u)/2+u\times(X_2-v)/2=(v\times X_1+u\times X_2)/2+(v\times u-u\times v)/2=X_3/2$；

3) v偶数，u奇数，X_1偶数，X_2偶数：$v\times(X_1/2)+u\times(X_2/2)=(v\times X_1+u\times X_2)/2=X_3/2$；

4) v偶数，u奇数，X_1奇数，X_2偶数：$v\times(X_1+u)/2+u\times(X_2-v)/2=(v\times X_1+u\times X_2)/2+(v\times u-u\times v)/2=X_3/2$；

5) v奇数，u偶数，X_1偶数，X_2偶数：$v\times(X_1/2)+u\times(X_2/2)=(v\times X_1+u\times X_2)/2=X_3/2$；

6) v奇数，u偶数，X_1偶数，X_2奇数：$v\times(X_1+u)/2+u\times(X_2-v)/2=(v\times X_1+u\times X_2)/2+(v\times u-u\times v)/2=X_3/2$。

以上推导的结果均满足公式：$v\times X_1+u\times X_2=X_3$。而且在满足$u$，$v$互素的条件下，已经穷举了$v$、$u$、$X_1$和$X_2$的奇偶情况。相同的方法可以对$Y$进行分析。其中，1)～4)表示模数$u$为奇数的情形，这时，只要判断到$X_1$为奇数，就执行$X_1=X_1+u$，否则执行$X_1=X_1/2$，不用参考$X_2$的奇偶性。当模数$u$为偶数的情形下，如果$X_1$为偶数，$X_2$可能为奇数，$X_1$是做$X_1=X_1+u$ 还是$X_1=X_1/2$，要由X_2的奇偶性来定：X_2为奇数，$X_1=X_1+u$。否则$X_1=X_1/2$。基于上面的分析，如果求的模数是奇数的时候，就可以省掉X_2和Y_2的操作，得到算法4。

算法4　模数为奇数时，对算法3改进的求模逆的算法

A_4-1 $(X_1,X_3)=(1,v)$;
A_4-2 $(Y_1,Y_3)=(0,u)$;
A_4-3 while (X_3 1 and Y_3 1)
A_4-3-1 if (X_3 is even) if(X_1 is even) $(X_1,X_3)=(X_1/2,X_3/2)$;
else $X_1=X_1+u$;
A_4-3-2 else if (Y_3 is even) if(Y_1 is even) $(Y_1,Y_3)=(Y_1/2,Y_3/2)$;
else $Y_1=Y_1+u$;
A_4-3-3 else if ($X_3>Y_3$) $(X_1, X_3)=(X_1-Y_1, X_3-Y_3)$;
else $(Y_1, Y_3)=(Y_1-X_1, Y_3-X_3)$;
A_4-4 if (X_3=1) return $X_1=v^{-1} \bmod u$;
else return $Y_1=v^{-1} \bmod u$;

算法4比起算法3，减少了2个临时变量，因此算法4中的减法开销要比算法3的小。算法4中的A_4-3-3需要执行的减法操作次数减少到2次，因此，算法4中的减法次数可以记为 $d_4=2d_1+A/2$ ，其中， d_1 为算法1中的减法次数，A为A_3-3-1和A_3-3-2中的减法操作。

如果u为偶数，那么对X_2(或Y_2)的奇偶判断就是必须的。如前页列举的情况5)、6)，X_1(或Y_1)是偶数的时候，X_2(或Y_2)可能为奇数。因此，只能通过判断X_2(或Y_2)的奇偶性来决定是否对X_1(或Y_1)加u。如果u是偶数，那么v必然是奇数(因为u和v是互逆的)。可以恢复算法2中的交换。其他的就直接套用算法4，得到算法5。

算法5 利用算法3和算法4改进的求模逆的算法

A_5-1 flag=0; /*标志位为0，表示模数是奇数；为1，则表示模数是偶数*/
A_5-2 if (u is even) flag=1; swap(u,v);
A_5-3 A_4 /*调用算法4，返回$X_1=u^{-1} \bmod v$ */
A_5-4 if (fla g=1) $X_2=(1-u\times X_1)/v=v^{-1} \bmod u$;/*由式(2)～(5)可以推导该结果*/

其中，X_1、X_2中的u、v均是A_5-2条件判断语句之前输入的u、v。

3 算法性能测试

在Intel PentiumIII 550 MHz**处理器，**192 M**内存的测试环境下用**C++**对以上几个算法进行性能测试，每个算法运行**50 000 000**次。**

表1 测试结果

输入	结果	算法2	算法3	算法4	算法5
(7,31)	9	27	20	18	20
(7,32)	23	31	22		21
(19 999 999,19 999 997)	9 999 999	42	13	12	13

表1中显示的算法5的测试时间有些恶劣，是因为在最开始的程序设计中，对于算法5中的条件判断语句(A_5-2)的处理比较烦琐：先设中间变量 $n=u\&0xfffffffe$；然后判断($n-u$)的值是否为1；这种方法增加了一次减法操作时间，这在迭代次数较少的情况下(如表上第一和第三组数据)，条件判断语句的开销就不能忽略。鉴于这种情况，后来的设计中均直接判断(u&1)的值是否为1。改过之后，第一和第三组数据的算法5的时间开销就同算法4是相同的了。

进一步的测试表明，随着模数的规模扩大，改进算法和原算法的性能差距还将增大。虽然在多精度的情况下，Lehmer**方法可以大大的提高扩展欧几里德的运算效率，但是**Stein**算法也有类似的方法加快运算速度**[3]**。因此，可以认为：两种算法的运算速度主要是由单精度的运算效率决定的。基于二进制的这个改进算法，对于硬件实现来更具有普遍性。在硬件上的实现效率将作为今后的研究方向。**

(**下转第**394**页**)

表1 Lena图像的方差、均值和信噪比参数比较(10%的椒盐噪声和(0,0.01)的高斯噪声)

图像类型	加噪图像	中值滤波	原算法 GCO	原算法 GOC	新算法 GCO	新算法 GOC
噪声方差	2 283.0	217.3	113.9	96.7	34.9	33.0
噪声均值	3.37	−0.65	5.71	−4.61	0.62	−1.41
信噪比	14.55	24.76	27.57	28.28	32.70	32.95

4 结 论

以上的分析和实验证明，本文提出的滤波器设计算法，相对于原方案，无论从滤除噪声方面还是从保持图像的细节信息方面，都有了长足的进步

参 考 文 献

[1] Kenneth R, Castleman. Digital image processing[M]. New Jersey: Prentice-Hall, 1996

[2] Serra J. Image analysis and mathematical morphology[M]. New York: Academic, 1982

[3] Akopian D, Vainio O, Agaian S, *et al*. Generalized stack filters[J]. IEEE Trans, Signal Processing, 1995, 43(6): 1 541-1 546

[4] 龚 炜, 石青云, 程民德. 数字空间中的数学形态学–理论及应用[M]. 北京: 科学出版社, 1997

[5] 赵春晖, 乔景渌, 孙圣和. 一类多结构元自适应广义形态滤波器[J]. 中国图像图形学报, 1997 ,11(2): 806-809

[6] 赵春晖, 孙圣和. 一种形态开、闭自适应加权组合滤波器[J]. 电子学报, 1997, 25(6):109-111

[7] Maragos P, Schafer R W. Morphological filters-part I and part II[J]. IEEE Trans. on ASSP, 1987, 35(8): 1 153-1 184

[8] Bataillou E, Thierry E, Rix H, *et al*. Weighted averaging using adaptive estimation of the weights[J]. Signal Processing, 1995, 44(1): 51-66

[9] 姚天任、孙 洪. 现代数字信号处理[M]. 武汉: 华中理工大学出版社, 1999

编 辑 熊思亮

(上接第386页)

参 考 文 献

[1]Stallings W著. 密码编码学与网络安全: 原理与实践[M]. 杨明, 胥光辉, 奇望东, 等译. 北京: 电子工业出版社, 2001

[2] Knuth D E著. 计算机程序设计艺术 第2卷 半数值算法[M]. 苏运霖译. 北京: 国防工业出版社, 2002

[3] 陈 运, 龚耀寰. RSA快速算法研究[J]. 通信保密. 2000, 25(3): 43-46

[4] 陈 运, 龚耀寰. RSA多重快速算法研究[C]. 中国计算机学会信息保密专业委员会会议: 大连, 2000: 30-36

编 辑 刘文珍