第33卷 第24期 计 算 机 工 程 2007年12月
Vol.33 No.24 Computer Engineering December 2007
·安全技术· 文章编号：1000—3428(2007)24—0155—03 文献标识码：A 中图分类号：TP309

Montgomery 模平方算法及其应用

王金荣[1,2]，周 贇[3]，王红霞[4]
(1. 杭州师范学院信息工程学院，杭州 310018；2. 浙江大学计算机科学与技术学院，杭州 310027；
3. 衢州学院信电系，衢州 324000；4. 成都理工大学信息工程学院，成都 610059)

摘 要：分析Montgomery模乘算法的设计思想和模平方中乘法的计算过程，通过引入两种新的平方计算方法以及对Montgomery算法的优化，提出适合于通用32位处理器实现的Montgomery模平方算法。将该方法应用于模幂计算，给出基于滑动窗口技术的Montgomery模幂算法。实验结果表明，该算法能将模幂的计算速度提高9%~12%。

关键词：RSA公钥；DSA公钥；Montgomery模乘算法

Montgomery Modular Squaring Algorithm and Its Application

WANG Jin-rong[1,2], ZHOU Yun[3], WANG Hong-xia[4]
(1. College of Information Engineering, Hangzhou Teacher's College, Hangzhou 310018; 2. College of Computer Science, Zhejiang University, Hangzhou 310027; 3. Department of Information and Electronic, Quzhou College, Quzhou 324000;
4. College of Information Engineering, Chengdu University of Technology, Chengdu 610059)

【Abstract】 This paper analyzes the basic design principle of Montgomery algorithm and the computing produce of multiplication in computing modular squaring. By adopting two new methods of computing squaring and optimizing Montgomery algorithm for squaring, it proposes the Montgomery modular squaring algorithm which is best suited for standard 32-bit processors. It also applies this new method to modular exponentiation and gives a Montgomery modular exponentiation algorithm based on sliding window techniques. According to experimentation, the new algorithm improves its efficiency with 9%~12%.

【Key words】 RSA; DSA; Montgomery modular multiplication algorithm

大整数模幂是RSA[1]、ElGamal[2]和DSA[3]等公钥密码和数字签名算法的基本运算，其运算速度对这些算法的实现起着重要的作用。模幂一般转化为一系列模平方和模乘来实现，其中模平方次数固定，模乘次数可优化。在目前常用的模幂算法中，模平方次约占2/3。因此，如何提高模平方的运算速度就显得非常重要。

1 Montgomery 模乘算法及其实现

1.1 Montgomery 模乘算法

1985年Montgomery提出了一种模乘的有效算法[4]，其设计思想是借助一个新的特殊剩余系，将普通模乘转换为易计算的特殊模乘。

定理 假设 n 和 r 是互素的两个整数，$n'=-n^{-1}\bmod r$，则对于所有整数 t，当 $m=t\times n'\bmod r$ 时，$(t+m\times n)/r$ 是一个整数，且满足：$(t+m\times n)/r\equiv t\times r^{-1}\bmod n$。

假设a,b,n均为k比特二进制大整数，$2^{k-1}\leqslant n<2^k$，$r=2^k$，且n和r互素，$n'=-n^{-1}\bmod r$，则计算 $c=a\times b \bmod n$ 时，可先将a,b转换到新的特殊剩余系(称为n-剩余系)中对应数 $\overline{a}$，$\overline{b}$：$\overline{a}=a\times r \bmod n$，$\overline{b}=b\times r \bmod n$；再计算 $t=\overline{a}\times\overline{b}$，$m=t\times n'\bmod r$；然后根据定理[5]，计算得

$$\overline{c}=\overline{a}\times\overline{b}\times r^{-1}\bmod n$$

此时 $0\leqslant\overline{c}<2n$ [6]，增加一条件减法使 $0\leqslant\overline{c}<n$。Montgomery模乘算法如下：

input $\overline{a}$, $\overline{b}$, n, r
output $\overline{c}=\overline{a}\times\overline{b}$
(1) $n'=-n^{-1}\bmod r$;
(2) $t=\overline{a}\times\overline{b}$;
(3) $m=t\times n'\bmod r$;
(4) $\overline{c}=(t+m\times n)/r$;
(5)if ($\overline{c}\geqslant n$) $\overline{c}=\overline{c}-n$;
(6)return $\overline{c}$;

1.2 SOS 实现

为加快Montgomery模乘的计算，1990年Dusse修改模简化计算[5]，将上述算法中的第(3)步、第(4)步有机地结合起来，用 ω 位的 n_0 ($n_0=-n_0^{-1}\bmod W$, $W=2^{\omega}$, ω 为CPU字长)代替 n'，字乘次数由原来的 $3s^2$ 次降为 $2s^2+s$ 次，额外空间降为 $2s+2$ 字。Koc于1996年归纳并总结了该算法的5种实现方法[7]，详细地讨论了SOS(separated operand scanning)方法：

input a,b,n,n_0;
output $t=a\overline{\times}b$;
(1)for i = 0 to s - 1
(2) C = 0;
(3) for j = 0 to s-1
(4) (C,S) = t[i + j] + a[j] * b[i] +C;

基金项目：浙江省自然科学基金资助项目(Y105067)；浙江省教育厅高校科研计划基金资助项目(20050718)
作者简介：王金荣(1973－)，男，讲师、博士研究生，主研方向：信息安全，计算机图形学；周 贇，讲师；王红霞，硕士研究生
收稿日期：2007-01-15 **E-mail**：jrwang17@163.com

(5) t[i + j] = S;
(6) t[i+s] = C;
(7) for i = 0 to s - 1
(8) C = 0;
(9) m = t[i] * n_0 mod W;
(10) for j = 0 to s-1
(11) (C,S) = t[i + j] + m * n[j] + C;
(12) t[i + j] = S;
(13) ADD(t[i + s],C);
(14) for j = 0 to s u[j] = t[j + s];
(15) if (t ⩾ n) t = t – n ;
(16) return t;

第(13)步的 ADD 函数完成进位传递，它先将进位 C 加到给定数组的 t[i+s]上，然后逐渐向上传递，直到不会出现进位为止。该方法首先使用普通乘法算法计算乘法，然后利用 Montgomery 算法进行模简化，其时间复杂度如表 1 所示。

表 1 SOS,OMMS1 和 OMMS2 三者的复杂度比较

算法	乘法	加法	读内存	写内存
SOS	$2s^2+s$	$4s^2+4s+2$	$6s^2+7s+3$	$2s^2+6s+2$
OMMS1	$\frac{3}{2}s^2+\frac{1}{2}s$	$3s^2+9s+3$	$4s^2+14s+4$	$\frac{3}{2}s^2+\frac{13}{2}s+3$
OMMS2	$\frac{3}{2}s^2+\frac{1}{2}s$	$3s^2+12s+\frac{11}{2}$	$\frac{9}{2}s^2+\frac{25}{2}s+4$	$\frac{3}{2}s^2+10s+3$

因为 $a_i \cdot a_j = a_j \cdot a_i$ 的特点，第(1)步-第(7)步可用下面的代码进行优化：

(1)for i = 0 to s - 1
(2) (C,S) = t[i + i] + a[i] * a[i];
(3) for j = i + 1 to s - 1
(4) (C,S) = t[i + j] + 2 * a[j] * a[i] + C;
(5) t[i + j] = S;
(6) t[i + s] = C;

上述代码在 32 位处理器上实现时，第(4)步的结果最大为 65 位，无法用两个 32 位字存储。为此，本文提出两种适合于通用 32 位处理器计算大整数平方的新方法，该方法易于通用处理器实现。

2 Montgomery 模平方算法

假设大整数

$$a=\sum_{i=0}^{s-1}a_iW^i$$

则 $t=a^2$ 的计算过程如下：

$$2a_{0,s-1}+2a_{0,s-2}+\cdots+2a_0a_4+2a_0a_3+2a_0a_2+2a_0a_1+a_0^2$$
$$2a_{1,s-1}+2a_{1,s-2}+\cdots+2a_1a_4+2a_1a_3+2a_1a_2+a_1^2$$
$$2a_{2,s-1}+2a_{2,s-2}+\cdots+2a_2a_4+2a_2a_3+a_2^2$$
$$\cdots$$
$$2a_{s-3,s-1}+2a_{s-2,s-2}+a_{s-3}^2$$
$$2a_{s-2,s-1}+2a_{s-2}^2$$
$$a_{s-1}^2$$

由上得

$$t=a^2=2\sum_{j=1}^{s-1}a_0a_jW^{0+j}+2\sum_{j=2}^{s-1}a_1a_jW^{1+j}+\cdots+$$
$$2\sum_{j=i+1}^{s-1}a_ia_jW^{i+j}+\cdots+2\sum_{j=s-1}^{s-1}a_{s-2}a_jW^{s-2+j}+$$
$$\sum_{i=0}^{s-1}a_i^2W^{i+i}=2\sum_{i=0}^{s-2}\sum_{j=i+1}^{s-1}a_ia_jW^{i+j}+\sum_{i=0}^{s-1}a_i^2W^{i+i} \quad (1)$$

可用下列两种方法计算式(1)。

方法 1

基本思路：首先计算

$$\sum_{i=0}^{s-2}\sum_{j=i+1}^{s-1}a_ia_jW^{i+j}$$

然后将结果左移 1 位并和

$$\sum_{i=0}^{s-1}a_i^2W^{i+i}$$

相加，最后得 t(设 $t=\sum_{i=0}^{2s-1}t_iW^i$)。算法如下：

(1)for i = 0 to s - 2
(2) C = 0；
(3) for j = i + 1 to s - 1
(4) (C,S) = t[i + j] + a[j] * a[i] + C;
(5) t[i + j] = S;
(6) t[i + s] = C;
(7) t = 2*t;
(8) C = 0；
(9) for i = 0 to s - 1
(10) (C,S) = t[i + i] + a[i] * a[i] + C;
(11) ADD(t[i+i+1],C);
(12) t[i + i] = S;
(13) t[2s - 1] = t[2s - 1] + C;

其中，第(7)步为左移 1 位。其时间复杂度如表 1 所示。

方法 2

基本思路：先计算 b=2a，然后通过 a 和 b 的特殊乘法规则计算式(1)。例如，s=3 时

$$a=(a_2a_1a_0)=a_2W^2+a_1W^1+a_0$$

可得t=a^2的计算过程如下:

(1)b=2a

设 $b=b_3W^3+b_2W^2+b_1W^1+b_0$, $\beta_i(0\leqslant i\leqslant 2)$ 表示 a_i 向左移 1 位后产生的进位，则有

$$b_0=2a_0-\beta_0W^1$$
$$b_1=2a_1-\beta_1W^1+\beta_0$$
$$b_2=2a_2-\beta_2W^1+\beta_1$$
$$b_3=\beta_2$$

(2)计算 t'

令 $t'_{2i+1}=\beta_ia_i$, $t'_{2i}=0$, $\beta_i=0$ 或 1($0\leqslant i\leqslant 2$)，则有

$$t'=\beta_2a_2W^5+\beta_1a_1W^3+\beta_0a_0W^1$$

利用特殊乘法规则计算 t：

$$t=b_3a_2W^5+(b_3a_1+a_2^2)W^4+(b_3a_0+b_2a_1)W^3+$$
$$(b_2a_0+a_1^2)W^2+b_1a_0W^1+a_0^2=$$
$$2\sum_{i=0}^{1}\sum_{j=i+1}^{2}a_ia_jW^{i+j}+\sum_{i=0}^{2}a_i^2W^{i+i}+\beta_2a_2W^5+\beta_1a_1W^3+\beta_0a_0W^1$$
$$t=t-t'$$

由此得方法 2 如下(其时间复杂度如表 1 所示)：

(1) 计算 b=2a，并计算得 t1[0] … t1[2s-1]($t_1[2i+1]=\beta[i]*a[i]$, $t_1[2i]=0$, β[i] 为计算 b 时的各个进位， $0\leqslant i\leqslant$ s-1)
(2)for i = 0 to s - 1
(3) (C,S) = t[i + i]+ a[i] * a[i];
(4) t[i + i] = S;
(5) for j = i + 1 to s
(6) (C,S)=t[i + j]+ a[i] * b[j]+ C;
(7) t[i + j]=S;
(8) t[i + s]=C;
(9)t=t-t_1;

用方法 1、方法 2 代替 1.2 节中的第(1)步~第(7)步，可得 Montgomery 模平方算法 OMMS1 和 OMMS2。三者的时间复杂度如表 1 所示。

3 Montgomery 模幂算法

Montgomery 算法通过剩余系转换方法计算模乘，极大地

提高了模乘计算速度。但因为从普通剩余系转换到 n-剩余系、计算 n' 和从 n-剩余系又转回到普通剩余系需要时间。所以当对某个模数仅计算一次模乘时，使用 Montgomery 模乘算法并不太好，而对于同一模数的多次模乘计算则比较合适(如模幂)。

文献[8]提出了一种改进的基于滑动窗口技术的模幂算法，由于 Montgomery 模乘的结果不是 $a\times b \bmod n$，而是 $a\times b\times r^{-1} \bmod n$，为使 Montgomery 模乘算法适合于模幂计算，本文对模幂算法作了适当的修改，由此得到 MME(Montgomery Modular Exponentiation)算法如下：

input m, e, n, r, r_2, n_0；预计算：$r=2^k \bmod n$, $r_2=r^2 \bmod n$

output $x = m^e \bmod n$

(1) $\bar{x}$ = r ;

(2)temp[1]=SOS(m,r_2,n,n_0);

(3)m2 = OMMS (temp[1],n,n_0);

(4)for i = 3 to 2^{size}-1 temp[i] = SOS (temp[i-2], m_2,n,n_0);

(5) e′ =Recoding(e);

(6)for i = d – 1 downto 0

(7) $\bar{x}$ = OMMS ($\bar{x}$,n,n_0);

(8)if (e'_i ==1) $\bar{x}$ = SOS ($\bar{x}$,temp[stack[top--]],n,n_0);

(9) x = SOS ($\bar{x}$,1 ,n,n_0);

(10)return x;

其中，SIZE 表示窗口长度；Recoding 是用滑动窗口技术对指数进行重编码函数，重编码时将当前窗口的值依次入栈 *stack*,*top* 为栈顶指针，e' 为重编码后指数，d 为 e' 的二进制长度；上述算法中的第(2)步将普通剩余系的整数 m 转化为 n-剩余系中的 $\bar{m}$，这是因为

$$\bar{m} = m \bar{\times} r^2 = m\times r\times r\times r^{-1} \bmod n = m\times r \bmod n$$

并将 $\bar{m}$ 存于 temp[1]中。随后由 $\bar{m}$ 计算

$$\bar{m}^i (i = 3,5,\cdots,\bar{m}^{SIZE} - 1)$$

一并存于 temp 数组中。算法的第(8)步将 n-剩余系中的 $\bar{x}$ 转换回普通剩余系中的 x，这是因为

$$\bar{x} \bar{\times} 1 = \bar{x}\times 1\times r^{-1} \bmod n = x\times r\times r^{-1} \bmod n = x$$

4 结束语

本节给出了 Montgomery 模平方算法的测试结果，测试环境为 Intel 866 MHz/128 MB Windows 2000。

首先对方法 1 和方法 2 进行测试。分析表明，$s = 32$ 时(1 024 位)计算一次平方，普通乘法算法需 1 024 次字乘，方法 1、方法 2 仅需 528 次字乘，约提高了 45%。三者的实际测试结果如表 2 所示。

由表 1 可知，OMMS 比 SOS 算法提高约为 20%~24%。从实际结果看(如表 3 所示)，OMMS1 可提高约 18%~19%，与理论分析也有一定的差距。

表 2 算法计算平方时的测试结果 单位：μs

算法	512 bits	1 024 bits	1 536 bits	2 048 bits
普通乘法算法	6.379	24.425	53.870	93.330
方法 1	4.632	16.508	34.496	59.250
方法 1 的提高/(%)	27.38	32.410	35.960	36.520
方法 2	4.889	17.130	36.856	63.740
方法 2 的提高/(%)	23.35	29.870	31.580	31.700

表 3 算法计算模平方时的测试结果 单位：μs

算法	512 bits	1 024 bits	1 536 bits	2 048 bits
SOS	13.123	48.833	107.483	188.745
OMMS1	10.660	40.600	86.980	151.860
OMMS1 的提高/(%)	18.770	18.850	19.070	19.550
OMMS2	11.780	41.140	89.090	158.630
OMMS2 的提高/(%)	12.200	15.750	17.110	15.950

最后，对 OMMS1 算法在模幂计算中的应用进行了测试(结果如表 4 所示)。模幂计算时，一般模平方次数约占 2/3，由此模平方对模幂速度改善最大为 16.67%。

表 4 SOS 和 OMMS1 计算模幂时的测试结果 单位：ms

算法	512 bits	1 024 bits	1 536 bits	2 048 bits
MME+SOS	7.436	56.889	187.200	431.500
MME+OMMS1	6.752	51.090	167.200	380.530
提高/(%)	9.200	10.200	10.6800	11.800

参考文献

[1] Rivest R L, Shamir A, Adleman L. A Method of Obtaining Digital Signature and Public Key Cryptosystems[J]. Comm. of ACM, 1978, 21(2): 120-126.

[2] ElGamal T. A Public-key Cryptosystem and a Signature Scheme Based on Discrete Logarithms[J]. IEEE Trans. on Information Theory, 1985, 31(4): 469-472.

[3] National Institute of Standards and Technology. NIST FIPS PUB 185, Digital Signature Standard[S]. 1994-05.

[4] Montgomery P L. Modular Multiplication Without Trial Division[J]. Mathematics of Computation, 1985, 44(170): 519-521.

[5] Dusse S R, Kaliski B S. A Cryptographic Library for the Motorola DSP56000[C]//Proc. of Advances in Cryptology-EUROCRYPT'90. New York: Spring-Verlage, 1990: 230-244.

[6] 王金荣, 丁 宏, 吴爱平. 基于大数模幂运算的公钥密码体制快速实现[J]. 杭州电子学院学报, 2003, 23(6): 59-62.

[7] Koc C K, Acar T, Kaliski B. Analyzing and Comparing Montgomery Multiplication Algorithms[J]. IEEE Micro, 1996, 16(6): 26-33.

[8] 丁 宏, 陈 勤. 大数模幂乘动态匹配快速算法及应用[J]. 小型微型计算机系统, 2002, 23(11): 1398-1400.

(上接第 154 页)

[4] Shamir A. ID-based Cryptosystems and Signature Schemes[C]//Proceedings of Crypto'84. Berlin: Springer-Verlag, 1985: 47-53.

[5] Girault M. Self-certified Public Keys[C]//Proc. of Eurocrypt'91. Berlin: Springer-Verlag, 1991: 491-497.

[6] Cao Z. A Threshold Key Escrow Scheme Based on Public Key Cryptosystem[J]. Science in China Series E, 2001, 44(4): 441-448.

[7] Shao Z. Improvement of Digital Signature with Message Recovery and Its Variants Based on Elliptic Curve Discrete Logarithm Problem[J]. Computer Standards & Interfaces, 2004, 27(1): 61-69.

[8] Pointcheval D, Stern J. Security Arguments for Digital Signatures and Blind Signatures[J]. Journal of Cryptology, 2000, 13(3): 361-396.