无移位操作的快速comb乘法算法

李 忠[1,2] 彭代渊[1] 郝悦彤[3]
(西南交通大学信息科学与技术学院 成都 610031)[1] (宜宾学院计算机科学与技术系 宜宾 644000)[2]
(西南大学经济管理学院 重庆 400715)[3]

摘 要 有限域GF(2^n)上乘法运算是影响GF(2^n)上椭圆曲线密码实现效率的关键运算之一。基于窗口技术的comb乘法算法,被认为是目前有限域GF(2^n)上乘法运算最快的算法之一。但是,它仍然使用了移位操作,而移位操作恰好又是域GF(2^n)乘法运算中很耗时的操作。提出并实现了一种新的基于窗口技术的快速comb乘法算法,该算法避免了移位操作,且不增加异或运算次数。理论分析和实验结果表明,新算法有很好的实现效率,适合于有限域GF(2^n)上椭圆曲线密码算法的软件实现。

关键词 GF(2^n)乘法运算,移位操作,comb乘法算法,椭圆曲线密码

Fast Comb-multiplication without Shift Operation

LI Zhong[1,2] PENG Dai-yuan[1] HAO Yue-tong[3]
(School of Information Science & Technology, Southwest Jiaotong University, Chengdu 610031, China)[1]
(Department of Computer Science&Technology, Yibin University, Yibin 644000, China)[2]
(College of Economics and Management Southwest University, Chognqing 400715, China)[3]

Abstract Efficient algorithms for multiplication in GF(2^n) are required to implement elliptic curve cryptosystems over GF(2^n). Comb-multiplication, based on window technology, is considered one of the fastest algorithms for multiplication in GF(2^n). However, it includes SHIFT operations, and SHIFT operations are just time-consuming operation among the operations of multuiplication in GF(2^n). A new algorithm for multiplication in GF(2^n), based on window technology, was presented, it completely avoided the SHIFT operations, and did not increase the number of XOR operations. The analysis and experiment results show that the new algorithm is faster than the comb-multiplication based on window technology, and is particularly useful for software implementation of elliptic curve cryptosystems over GF(2^n).

Keywords Multiplication in GF(2^n), Shift operation, Comb multiplication, Elliptic curve cryptography

1 引言

1985年,Neal Koblitz[1]和Victor Miller[2]分别独立地提出了椭圆曲线密码体制(elliptic curves cryptography, ECC),因其具有更高的安全性、计算量小、存储空间占用少、带宽要求低等突出优点,使得其具有广阔的应用前景,被认为是最值得关注的密码体制,是目前信息安全领域研究的一个热点。由于ECC的运算是基于有限域运算的,从而有限域运算的有效实现是有效实现ECC的重要的先决条件[3-5],其中有效的乘法运算是有效实现ECC的关键之一。

ECC的有效实现主要用到3种域:素域GF(p)、二进制域GF(2^n)和最佳扩域(OEF)。但从方便实现来看,二进制域是最有吸引力的,其运算只涉及移位(SHIFT)和按位的模2加(XOR),这在通用处理器上用软件实现是十分诱人的。有限域GF(2^n)上乘法运算的最基本算法是shift-and-add算法,该算法非常适合硬件实现,其中向量的一次移位可用一个时钟周期完成,但大量的字移位使得它不适合软件实现。Koc与Acar在文献[6]中给出了一种应用在有限域GF(2^n)上的Montgomery算法,该算法难于在通用处理器上用软件实现。C. Lim和P. Lee在文献[7]中给出了comb乘法算法,由于它大大减少了移位运算次数,其效率明显优于shift-and-add算法,在此基础上,J. López与R. Dahab在文献[8]中将窗口技术应用到有限域GF(2^n)的乘法运算上,提出了一种基于窗口技术的comb乘法算法,进一步减少了移位运算次数,其运算效率比shift-and-add算法的运算效率提高了大约3~5倍,特别适合于椭圆曲线密码的软件实现[8],被认为是目前有限域GF(2^n)上乘法运算最快的算法之一[4]。

上述这些算法都使用了移位操作,而移位操作恰好又是域GF(2^n)乘法运算中很耗时的操作,从而导致这些算法的效率还不够高,本文提出并实现了一种新的基于窗口技术的快速comb乘法算法,避免了移位操作,且不增加异或运算次数。理论分析和实验结果表明,所得算法有更高的实现效率,适合于域GF(2^n)上椭圆曲线密码算法的软件实现。

到稿日期:2008-06-30 本文受四川省教育厅资助科研项目(07ZA145)资助。
李 忠(1963-),男,博士生,副教授,主要研究方向为密码学、信息安全,E-mial:lz8056859@163.com;彭代渊(1955-),男,博士,教授,博士生导师,主要研究方向为密码学、信息安全、编码理论。

2 域 GF(2^n)上元素的多项式基表示

构成有限域 GF(2^n)的最常用的两种方法是正规基表示法和多项式基表示法，正规基表示法更适合于硬件实现[9]，本文采用多项式基表示法构成域 GF(2^n)，在这种表示法中，域 GF(2^n)的元素是次数最多为 $n-1$ 次的二进制多项式，即：

$$GF(2^n)=\{a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+\cdots+a_2x^2+a_1x^1+a_0 \mid a_i\in\{0,1\}\}$$

选取 $f(x)\in GF(2)[x]$为 n 次二进制不可约多项式，设 $a=a(x)=a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+\cdots+a_2x^2+a_1x^1+a_0$，$b=b(x)=b_{n-1}x^{n-1}+b_{n-2}x^{n-2}+\cdots+b_2x^2+b_1x^1+b_0\in GF(2^n)$，则域 GF($2^n$)上的加法和乘法运算如下：

加法：$c=a+b$，其中，$c=c(x)=\sum_{i=0}^{n-1}c_ix^i$，$c_i=(a_i+b_i)\bmod 2$

乘法：$c=a\cdot b \bmod f$，其中，$c=c(x)=\sum_{i=0}^{n-1}c_ix^i$ 是多项式 $\sum_{i=0}^{n-1}a_ix^i$ 与多项式 $\sum_{i=0}^{n-1}b_ix^i$ 的乘积被不可约多项式 $f(x)$ 除的余式。

3 基于窗口技术的 comb 乘法算法

J. López 与 R. Dahab 扩展了 C. Lim 和 P. Lee 所提出的 comb 算法，下面介绍 J. López 与 R. Dahab 提出的基于窗口技术的 comb 乘法算法。

在多项式基表示下，一个域元素 $a=a(x)=a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+\cdots+a_2x^2+a_1x^1+a_0\in GF(2^n)$与一个 n 维向量 $a=(a_{n-1},a_{n-2},\cdots,a_2,a_1,a_0)\in GF(2)^n$ 一一对应。从而 GF(2^n)又可以表示为：

$$GF(2^n)=\{(a_{n-1},a_{n-2},\cdots,a_2,a_1,a_0) \mid a_i\in\{0,1\}\}$$

假设实现平台是字长为 z 位的系统(其典型取值为 8，16，32，64)，令 $k=\lceil n/z\rceil$，域元素 $a=(a_{n-1},a_{n-2},\cdots,a_2,a_1,a_0)\in GF(2^n)$可表示为：

$a=(A_{k-1},\cdots,A_1,A_0)$，其中 $A_i=(a_{iz+z-1},\cdots,a_{iz+1},a_{iz})\in GF(2^z)$，$0\leqslant i\leqslant k-1$，$A_{k-1}$的最左边的 $zk-n$ 位置为 0。

于是有：

$$a(x)=\sum_{i=0}^{n-1}a_ix^i=\sum_{i=0}^{k-1}A_i(x)x^{iz}=\sum_{i=0}^{k-1}(\sum_{j=0}^{z-1}a_{iz+j}x^j)x^{iz}$$

进而：

$$a(x)\cdot b(x)=(\sum_{i=0}^{k-1}(\sum_{j=0}^{z-1}a_{iz+j}x^j)x^{iz})b(x)=\sum_{i=0}^{k-1}(\sum_{j=0}^{z-1}a_{iz+j}x^jb(x))x^{iz} \quad (1)$$

取窗口宽度 w，可以根据式(2)预计算 $P_{2^w}[u](x)$ $(0\leqslant u<2w)$：

$$u=(u_{w-1}\cdots u_1u_0)_2,\ P_{2^w}[u](x)=(u_{w-1}x^{w-1}+\cdots+u_1x+u_0)b(x) \quad (2)$$

进而式(1)为

$$\begin{aligned}a(x)\cdot b(x)&=\sum_{i=0}^{k-1}(\sum_{j=0}^{z-1}a_{iz+j}x^jb(x))x^{iz}\\&=\sum_{i=0}^{k-1}(\sum_{j=0}^{z/w-1}(a_{iz+j+w-1}x^{w-1}+\cdots+a_{iz+j+1}x+a_{iz+j})x^{wj}b(x))x^{iz}\\&=\sum_{j=0}^{z/w-1}x^{wj}(\sum_{i=0}^{k-1}x^{iz}P_{2^w}[u_{i,j}](x))\end{aligned}$$

其中 $u_{i,j}=(a_{iz+j+w-1}\cdots a_{iz+j})_2$ (3)

根据式(3)有如算法 1 所示的基于窗口技术的 comb 乘法算法。

算法 1 López/Dahab's algorithm with windows of width w

INPUT: a=($A_{k-1}\cdots A_1A_0$), b=($B_{k-1}\cdots B_1B_0$), f=($F_{k-1}\cdots F_1F_0$)

OUTPUT: c(x) = a(x) · b(x) mod f(x)

1. for j from 0 to 2^w do
   $P_{2^w}[j]=(j_{w-1}x^{w-1}+\cdots+j_1x+j_0)b(x)$; where $j=(j_{w-1}\cdots j_1j_0)_2$
2. $T_i=0$; i=0,1,2,…,2k−1
3. for j from z/w−1 downto 0 do
   3.1 for i from 0 to k−1 do
   $u_{i,j}=A_i/2^{wj} \bmod 2^w$;
   for l from 0 to k−1 do
   $T_{l+i}=T_{l+i}\oplus P_{2^w}[u_{i,j}][l]$;
   3.2 if j≠0 then $T=x^wT$;
4. c =T mod f;
5. return(c).

其中 $T=(T_{2k-1},\cdots,T_1,T_0)$为中间结果。

4 无移位操作的快速 comb 乘法算法

算法 1 大大减少了移位操作，通过预计算，计算时查表一次处理 A[i]的 w 位，其运行效率比 shift-and-add 算法要快大约 3～5 倍，但是，它在预计算中(算法 1 的第 1 步)包含了 $w-1$ 次 k 字移位操作，在乘法运算(算法 1 的第 3 步)中包含了 $z/w-1$ 次 $2k$ 字移位操作[8]，这恰好又是整个域 GF(2^n)乘法运算中很耗时的操作，从而导致其效率还不够高。下面将对其进行改进，设计无移位操作的快速 comb 乘法算法。

4.1 无移位操作的预计算算法

对于 $b=(b_{n-1},b_{n-2},\cdots,b_2,b_1,b_0)\in GF(2^n)$，窗口宽度 w，根据式(2)，对任意 $u=(u_{w-1}\cdots u_1u_0)_2$，有：

$$\begin{aligned}P_{2^w}[u](x)&=(u_{w-1}x^{w-1}+\cdots+u_1x+u_0)b(x)\\&=(\sum_{i=0}^{w-1}u_ix^i)(\sum_{j=0}^{n-1}b_jx^j)\\&=\sum_{i=0}^{w-1}\sum_{j=0}^{n-1}u_ib_jx^{i+j}\in GF(2^{n+w-1})\end{aligned} \quad (4)$$

根据式(4)，并充分利用 $u_i\in\{0,1\}$这一特性，有如算法 2 所示的无移位操作的预计算算法。

算法 2 precomputation algorithm without shift operation

INPUT: b=(b_{n-1}, b_{n-2}, …, b_1, b_0), w

OUTPUT: P_{2^w}

```
for j from 0 to 2^w do
    P_{2^w}[j]=0; m=j;
    for i from 0 to w−1 do
        if (m mod 2)≠0 then
            myprecat_add(P_{2^w}[j], b, i);
        m=m/2;
return P_{2^w}
```

其中，子算法 myprecat_add 如算法 3 所示。

算法 3 myprecat_add Subalgorithm

INPUT: p=(p_{n+w-1}, p_{n+w-2}, …, p_1, p_0), b=(b_{n-1}, b_{n-2}, …, b_2, b_1, b_0), pindex

OUTPUT: p=(p_{n+w-1}, p_{n+w-2}, …, p_1, p_0)

```
for l=0 to n−1 do
    p_{l+pindex}=(p_{l+pindex}+ b_l) mod 2;
    return p.
```

采用算法 2 进行预计算完全避免了移位运算，且不增加异或运算次数，提高了预计算的效率。

4.2 无移位操作的快速 comb 乘法算法

对 $a=(A_{k-1},\cdots,A_1,A_0)\in GF(2^n)$，其中 $A_i=(a_{iz+z-1},\cdots,a_{iz+1},a_{iz})\in GF(2^z)$，$0\leqslant i\leqslant k-1$，将 A_i 按如下方式进行划分：

$$A_i=W[i][z/w-1]W[i][z/w-2]\cdots W[i][1]W[i][0],$$

其中窗口 $W[i][j]$的宽度均为 w 比特，$0\leqslant j\leqslant z/w-1$。

此时，算法 1 中第 3.1 步的 $u_{i,j}$就是 $W[i][j]$，于是式(3)可变为：

$$\begin{aligned}a(x)\cdot b(x)&=\sum_{i=0}^{k-1}\left(\sum_{j=0}^{z/w-1}\left(a_{iz+j+w-1}x^{w-1}+\cdots+a_{iz+j+1}x+a_{iz+j}\right)x^{wj}b(x)\right)x^{iz}\\&=\sum_{j=0}^{z/w-1}\sum_{i=0}^{k-1}x^{iz+jw}P_{2^w}[W[i][j]](x)\end{aligned}\quad(5)$$

根据式(5)，算法 1 中第 3.1 步可修改为：

3.1' for i from 0 to k−1 do

mycat_add(P_{2^w}[W[i][j]],T,z * i+w * j);

其中，子算法 mycat_add 如算法 4 所示。

算法 4 mycat_add Subalgorithm

INPUT:p=$(p_{n+w-1},p_{n+w-2},\cdots,p_1,p_0)$,T=$(t_{2n-1},t_{n-2},\cdots,t_1,t_0)$, pindex

OUTPUT:T=$(t_{2n-1},t_{n-2},\cdots,t_1,t_0)$

for i from 0 to n+w−1 do

$t_{i+pindex}=(t_{i+pindex}+p_i)$ mod 2;

return T.

此时，可删除算法 1 中的第 3.2 步，从而避免了移位操作，且不增加异或运算次数。

事实上，可将 $a=(a_{n-1},a_{n-2},\cdots,a_2,a_1,a_0)\in GF(2^n)$按如下的方式划分：

$a=W[s-1]W[s-2]\cdots W[1]W[0]$，其中窗口 $W[j]$的宽度均为 w 比特，$s=\lceil n/w\rceil$，$0\leqslant j\leqslant s-2$，其中，$W[s-1]$最左边的 $sw-n$ 位补 0，于是：

$$\begin{aligned}a(x)\cdot b(x)&=\left(\sum_{i=0}^{n-1}a_ix^i\right)b(x)\\&=\left(\sum_{i=0}^{s-1}x^{iw}W[i](x)\right)b(x)\\&=\sum_{i=0}^{s-1}x^{iw}(W[i](x)b(x))\\&=\sum_{i=0}^{s-1}x^{iw}P_{2^w}[W[i]]\end{aligned}\quad(6)$$

根据式(6)，算法 1 中第 3 步可修改为：

3.' for j from 0 to s−1 do

mycat_add(P_2w[W[j]],T,j * w)；其中子算法 mycat_add 参见算法 3。

综上所述，可得如算法 5 所示的无移位操作的快速 comb 乘法算法。

算法 5 fast comb multioplication algorithm without shift operation

INPUT:a=$(a_{n-1},a_{n-2},\cdots,a_1,a_0)$,b=$(b_{n-1},b_{n-2},\cdots,b_1,b_0)$,f=$(f_n,f_{n-1},\cdots,f_1,f_0)$

OUTPUT:c(x) = a(x) · b(x) mod f(x)

1. 利用算法 2 预计算 $P_{2^w}[j]$;j=0,1,2,…,2^w-1

2. T =0 ;

3. for j from 0 to s−1 do

mycat_add(P_{2^w}[W[j]],T,j * w);//算法 3

4. c =T mod f;

5. return(c).

5 效率分析

由于算法 1 与算法 5 均将乘法(算法中第 1−3 步)运算与取模运算(算法中第 4 步)分别进行，它们可以使用同一种取模运算算法，从而两算法在第 4 步中有相同的耗费，为此，在以下的分析中将不考虑取模运算的影响，只分析两算法在预计算(算法中第 1 步)及乘法运算(算法中第 3 步)方面的耗费。

算法 1 的预计算需 $w-1$ 次 k 字移位操作，同时需要 $w2^{w-1}$次 n 位(bit)异或运算；在乘法运算中需要 $z/w-1$ 次 $2k$ 字移位操作，同时需要 kz/w 次约 n 位(其实为 $n+w-1$ 位)异或运算。而算法 5 的预计算，只需 $w2^{w-1}$次 n 位异或运算即可，在乘法运算中只需 s 次约 n 位(其实为 $n+w-1$ 位)异或运算即可。对于 $n=163$，$z=32$，$w=4$，$k=6$ 时，两算法的操作数如表 1 所列。

表 1 算法 1,5 操作数比较

	n-bit XOR	k-word SHIFT
算法 1	32+48=80	17
算法 5	32+41=73	0

在“n 位异或运算与 k 字移位操作所耗费的时间相同”的假设前提下(在软件实现中，这种假设是合理的)，由表 1 可知，算法 5 的效率比算法 1 的效率提高了 24.74%。在 $b(x)$ 预知的情况下，可以预先计算出 $P_{2^w}[i]$，并存储计算结果，此时，算法 5 的效率比算法 1 的效率提高了 33.87%。

我们在联想天逸 Y200(笔记本)、Microsoft Windows XP (Professional 2002)平台下，用 C 语言在 Visual C++ 6.0 环境下，实现了上述算法。对于 $n=163$，$z=32$，$w=4$，$k=6$，利用 Miracl5.2 中的 bigdig 函数随机产生域 $GF(2^{163})$中的 21 个域元素，算法 1 的运行时间约为 2.1ms，算法 5 的运行时间约为 1.5ms，效率提高了 28.57%。

结束语 有限域 $GF(2^n)$上乘法运算是影响 $GF(2^n)$上椭圆曲线密码实现效率的关键运算之一。基于窗口技术的 comb 乘法算法，被认为是目前有限域 $GF(2^n)$上乘法运算最快的算法之一。但是，它仍然使用了移位操作，而移位操作恰好又是域 $GF(2^n)$乘法运算中很耗时的操作。本文提出并实现了一种新的基于窗口技术的快速 comb 乘法算法，该算法避免了移位操作，且不增加异或运算次数。理论分析和实验结果表明，新算法有很好的实现效率，适合于有限域 $GF(2^n)$上椭圆曲线密码算法的软件实现。

参 考 文 献

[1] Koblitz N. Elliptic curve cryptosystems. Mathematics of Computation, 1987, 48: 203-209

[2] Miller V S. Use of elliptic curves in cryptography// Advances in Cryptology-Proceedings of Crypto'85, Lecture Notes in Computer Science, 218. Springer-Verlag, 1986: 417-426

[3] Washington L C, Curves E. Number Theory and Cryptography. Chapman & Hall/CRC, New York, 2003

[4] Hankerson D, Menezes A, Vanstone S. Guide to elliptic curve cryptography. Springer-Verlag Professional Computing Series, 2004

[5] Rosing M. Implementing Elliptic Curve Cryptography. Manning publications, Greenwich, 1999

[6] Koc C K, Acar T. Montgomery multiplicationinGF(2^k). Designs, Codes and Cryptography, 1998, 14: 57-69

[7] Lim C, Lee P. More flexible exponentiation with precomputation // Advances in Cryptology-CRYPTO'94 (LNCS 839) [115]§ 1994: 95-107

[8] López J, Dahab R. High-speed software multiplication in F_{2^m} // Progress in Cryptology—INDOCRYPT 2000 (LNCS 1977) [393]. 2000: 203-212

[9] Agnew G B, Mullin R C, Vanstone S A. An implementation of elliptic curve cryptosystems over $F_2{}^{155}$. IEEE journal on selected areas in communications, 1993, 203(11): 804-813