一种基于优化正规基的域元素乘法改进算法

王金荣[1,2]
WANG Jin-rong[1,2]

1.杭州师范大学 信息科学与工程学院,杭州 310036
2.浙江大学 计算机科学与技术学院,杭州 310027
1.School of Information Science & Engineering,Hangzhou Normal University,Hangzhou 310036,China
2.College of Computer Science,Zhejiang University,Hangzhou 310027,China

WANG Jin-rong.Improved algorithm of finite field multiplication in optimal normal basis.Computer Engineering and Applications,2008,44(23):62-64.

Abstract: Firstly,this article has discussed finite field multiplication,and given its general formula in optimal normal basis.Then, the authors has researched the Rosing algorithm and Ning-Yin algorithm,and put forward an improved algorithm and three pre-computation methods.Finally,the author has tested all above algorithms.According to the analysis and experimentation,the new improved algorithm improves in its efficiency with about 20% by comparison with Ning-Yin algorithm.
Key words: ECC;optimal normal basis;finite field multiplication

摘 要:首先讨论了基域 $GF(2^m)$上域元素的乘法运算,给出了优化正规基下乘法的一般计算公式。然后深入研究了 Rosing 和 Ning-Yin 算法,提出了一种改进算法和三种预计算方法。最后,分析和测试结果表明该改进算法比 Ning-Yin 算法提高了约 20%。
关键词:ECC;优化正规基;域元素乘法
DOI:10.3778/j.issn.1002-8331.2008.23.019 **文章编号**:1002-8331(2008)23-0062-03 **文献标识码**:A **中图分类号**:TP309

基域 $GF(2^m)$上域元素乘法是实现椭圆曲线公钥密码体制(ECC)的核心运算。虽然对 ECC 而言,基域中元素的表示方法并不影响其安全性,但是它直接影响该体制的可行性和运行效率。目前常使用多项式基和正规基表示域元素,与多项式基相比,正规基的优势在于域元素平方可通过向量表示的一次简单循环移位来完成,其缺点是乘法运算相对较为复杂。研究表明[1],正规基特别是优化正规基更适合于硬件实现,而在软件实现上多项式基更为高效。因此如何在保持正规基已有优势的情况下,提高乘法运算的速度是一个有较高研究价值的问题。

本文首先讨论了正规基和优化正规基上的乘法运算,给出了乘法的一般计算公式;深入研究了 Rosing[2]和 Ning-Yin[3]算法,提出了一种改进算法和三种预计算方法。最后,分析和测试结果证明了该改进算法的可行性和有效性。

1 基域 $GF(2^m)$和正规基

设 β 是基域 $GF(2^m)$中的元素,则可用多项式加以表示,即:$\beta=a_{m-1}x^{m-1}+\cdots+a_1x+a_0$。由此基域上一组正规基可表示为:$\{\beta^{2^{m-1}},\cdots,\beta^{2^2},\beta^{2^1},\beta^{2^0}\}$,对于任意的 m,正规基总是存在的[4]。

基域中任一元素 A 可表示为 $A=a_{m-1}\beta^{2^{m-1}}+\cdots+a_1\beta^{2^1}+a_0\beta^{2^0}=(a_{m-1}\cdots a_1a_0)$,$a_i\in\{0,1\}$,$0\leqslant i\leqslant m-1$。特别地,零元素表示为 0=(0…00),乘法单位元 1 表示为 1=(1…11)。

正规基表示的最大优势是可以快速完成平方计算,它仅需对向量表示的域元素作一次简单循环移位即可。由于$(\beta^{2^i})^2=\beta^{2^{i+1}}$及$\beta^{2^m}=\beta$,则:$A^2=(\sum_{i=0}^{m-1}a_i\beta^{2^i})^2=\sum_{i=0}^{m-1}a_i\beta^{2^{i+1}}=\sum_{i=0}^{m-1}a_{i-1}\beta^{2^i}=(a_{m-2}\cdots a_1a_0a_{m-1})$,那么对任一整数 $s(0\leqslant s\leqslant m-1)$,域元素 A 的第 2^s 次幂可通过 A 循环左移 s 位而快速得到,即$A^{2^s}=(a_{m-s-1}a_{m-s-2}\cdots a_1a_0a_{m-1}\cdots a_{m-s+1}a_{m-s})$。

类似地,域元素 A 平方根也容易计算,仅循环右移一位即可得到:$A^{1/2}=(a_0a_{m-1}a_{m-2}\cdots a_2a_1)$,但是,正规基的乘法计算相当复杂。

2 正规基上的乘法

假设基域 $GF(2^m)$上的两域元素为 $A=\sum_{i=0}^{m-1}a_i\beta^{2^i}$、$B=\sum_{j=0}^{m-1}b_j\beta^{2^j}$,则 $C=AB=\sum_{i=0}^{m-1}\sum_{j=0}^{m-1}a_ib_j\beta^{2^i}\beta^{2^j}$。同时,因 C 是正规基表示下的基域元素,即 $C=\sum_{k=0}^{m-1}c_k\beta^{2^k}$,则有:$\beta^{2^i}\beta^{2^j}=\sum_{k=0}^{m-1}\lambda_{ijk}\beta^{2^k}$。其中 $0\leqslant i,j\leqslant m-1$,$\lambda_{ijk}=0$

基金项目:浙江省自然科学基金(the Natural Science Foundation of Zhejiang Province of China under Grant No.Y105067);浙江省教育厅高校科研计划项目(No.20050718,No.20070411)。
作者简介:王金荣(1973-),男,讲师,博士研究生,主要研究方向为信息安全、计算机图形学。
收稿日期:2007-10-16 **修回日期**:2007-11-27

或1,系数 λ_{ijk} 所组成的矩阵称为lambda矩阵或乘法表。由前两式可得 $\boldsymbol{c}_k$ 的计算式为:

$$\boldsymbol{c}_k=\sum_{i=0}^{m-1}\sum_{j=0}^{m-1}a_i b_j\lambda_{ijk} \tag{1}$$

可以证明式(1)可以转化为仅具有 λ_{ij0} 的式子[4],即 $\lambda_{ijk}=\lambda_{i-k,j-k,0}$。重写式(1)可得:

$$\boldsymbol{c}_k=\sum_{i=0}^{m-1}\sum_{j=0}^{m-1}a_i b_j\lambda_{i-k \bmod m,j-k \bmod m,0}=\sum_{i=0}^{m-1}\sum_{j=0}^{m-1}a_{i-k \bmod m}b_{j-k \bmod m}\lambda_{ij}$$

即:

$$\boldsymbol{c}_k=(a_k a_{k+1}\cdots a_{m-1}a_0 a_1\cdots a_{k-1})\lambda(b_k b_{k+1}\cdots b_{m-1}b_0 b_1\cdots b_{k-1})^{\mathrm{T}}=$$

$$(a_k a_{k+1}\cdots a_{m-1}a_0 a_1\cdots a_{k-1})\begin{bmatrix}\lambda_{00} & \lambda_{01} & \cdots & \lambda_{0,m-1}\\ \lambda_{10} & \lambda_{11} & \cdots & \lambda_{1,m-1}\\ \vdots & \vdots & & \vdots\\ \lambda_{m-1,0} & \lambda_{m-1,1} & \cdots & \lambda_{m-1,m-1}\end{bmatrix}\begin{bmatrix}b_k\\ b_{k+1}\\ \vdots\\ b_{m-1}\\ b_0\\ b_1\\ \vdots\\ b_{k-1}\end{bmatrix} \tag{2}$$

由式(2)可知,对 A 和 B 作适当的移位可计算出 c_k。因这里的乘即AND运算,加法即XOR运算,所示上式计算无需进位。进一步分析式(2),可得如下等式:

$$\boldsymbol{c}_k=\sum_{i=0}^{m-1}[a_{k+i}(\sum_{j=0}^{m-1}\lambda_{ij}b_{j+k})] \tag{3}$$

由式(3)可知,若给定 $\boldsymbol{c}_k$ 的表示式,只需对 A 和 B 的下标增加一位(需模 m)可得 c_{k+1} 的表示式。

3 优化正规基上的乘法

3.1 乘法的一般计算公式

定义1 优化正规基使lambda矩阵中非零 λ_{ij} 的个数最少的正规基。

定义2 lambda矩阵的复杂度 m_λ 优化正规基中非零的 λ_{ij} 个数。

基域 $GF(2^m)$ 上正规基的最小复杂度是 $2m-1$[5],因此域 $GF(2^m)$ 的优化正规基就是满足 $m_\lambda=2m-1$ 的正规基。

定理1[2] 基域 $GF(2^m)$ 中优化正规基存在定理:(1)基域 $GF(2^m)$ 上存在优化正规基,当且仅当 $m+1$ 是一个素数,且2是 Z_{m+1} 的本原元(Type I ONB)。(2)如果 $2m+1$ 是素数且2是 Z_{2m+1} 的本原元,则基域 $GF(2^m)$ 上存在优化正规基(Type IIa ONB)。(3)如果 $2m+1$ 是素数且 $2m+1\equiv3(\bmod 4)$,同时2是生成 Z_{2m+1} 的二次剩余类群,则基域 $GF(2^m)$ 上存在优化正规基(Type IIb ONB)。

由定理1可知并不是所有 m 均存在优化正规基。具体优化正规基的存在情况可参见文献[2],lambda矩阵的生成可参见文献[2,6]。对优化正规基Type I和Type II来说,它们唯一不同是lambda矩阵中各非零元的计算方法。因此,统一使用数组 $j1[m]$ 和 $j2[m]$ 表示lambda矩阵中的非零元1,其中 $(i,j1[i])$,$(i,j2[i])$ 分别表示矩阵中第 i 行第 $j1[i]$ 列和 $j2[i]$ 列的两个非零元的行号和列号。

由此可得优化正规基表示下的乘法计算式:

$$\boldsymbol{c}_k=\sum_{i=0}^{m-1}\sum_{j=0}^{m-1}a_{i+k}b_{j+k}\lambda_{ij}=(a_k b_{j1[0]+k}+\sum_{i=1}^{m-1}a_{k+i}(b_{j1[i]+k}+b_{j2[i]+k})) \tag{4}$$

由上式可得域元素乘法的一般计算公式:

$$\mathrm{C}=AB_{\to j1[0]}+\sum_{i=1}^{m-1}A_{\to i}(B_{\to j1[i]}+B_{\to j2[i]}) \tag{5}$$

其中 $B_{\to j1[0]}$ 表示 B 向右移 $j1[0]$ 位,$A_{\to i}$、$B_{\to j1[i]}$、$B_{\to j2[i]}$ 亦同理。

3.2 Rosing算法

具体实现式(5)时,用 $s=\lceil\frac{m}{w}\rceil$ 个 w 位(w 为CPU字长)的整数组成的数组来存储域元素,即 $A=(A[s-1],\cdots,A[1],A[0])$,$B=(B[s-1],\cdots,B[1],B[0])$ 其中 $A[i]=a_{i^*w+w-1}a_{i^*w+w-2}\cdots a_{i^*w+1}a_{i^*w}$,$B[i]=b_{i^*w+w-1}b_{i^*w+w-2}\cdots b_{i^*w+1}b_{i^*w}$ $(0\leqslant i\leqslant S-1)$。为方便描述,设 w 整除 m,如不整除,增加少许处理即可。Rosing在文献[2]中提出了一个关于优化正规基乘法的有效算法,其主要思想是:首先预计算 B 的 m 次右循环移位,然后计算式(5)中部分和 $A_{\to i}(B_{\to j1[i]}+B_{\to j2[i]})$,最后累加这些结果得 C。具体算法如下所示。

input:A,B
output:C
(1)预计算得数组 $BM[m]$,其中 $BM[k]=B_{\to k}$,$0\leqslant k\leqslant m-1$
(2)for i=0 to s-1 do
(3) $C[i]=A[i]$ & $BM[j1[0]][i]$;
(4)for i=1 to m-1 do
(5) *rot_right*(A);
(6) for j=0 to s-1 do
(7) $C[j]=A[j]$ &($BM[j1[i]][j]$⊕$BM[j2[i]][j]$)。
(8)return C;

其中 *rot_right* 是右循环移一位函数。

下面分析Rosing算法:算法第(7)步共需 $ms=m^2/w$ 次运算,第(1)步预计算阶段需 $m-1$ 次循环移位,每次移位约需 s 次运算,故预计算需 $m(s-1)\approx m^2/w$ 次运算。所需的存储空间为 $ms\times\frac{w}{8}=\frac{m^2}{8}$。Rosing算法中预计算次数较多,为此,Ning-Yin首先运用不同的预计算策略,并采用不同的计算思想,进一步降低了乘法的时间和空间复杂度。

3.3 Ning-Yin改进算法

Ning-Yin在文献[3]中提出了一种与Rosing不同计算思想的乘法算法,文章中共有4种算法,其中算法3较具代表性,它不仅效率最高且适应于Type I和Type II优化正规基的情况。

定义向量:

$$\begin{aligned}&\boldsymbol{PA}=(PA[2m-1],PA[2m-2],\cdots,PA[1],PA[0])\\ &\boldsymbol{PB}=(PB[2m-1],PB[2m-2],\cdots,PB[1],PB[0])\end{aligned} \tag{6}$$

其中 $PA[i+m]=PA[i]=(a_i a_{i+1}\cdots a_{i+w-1})$,$PB[i+m]=PB[i]=(b_i b_{i+1}\cdots b_{i+w-1})$,$i=0,1,2,\cdots,m-1$。若按相同的方法定义 $\boldsymbol{PC}$ 向量,则有:$\boldsymbol{PC}=(PC[(s-1)w],\cdots,PC[2w],PC[w],PC[0]$,利用式(6)定义的向量,修改式(5)有:$PC[0]=PA[0]PB[j1[0]]+\sum_{i=1}^{m-1}(PA[i+0](PB[j1[i]]+PB[j2[i]]))$,$PC[w]=PA[w]PB[j1[0]+w]+\sum_{i=1}^{m-1}(PA[i+w](PB[j1[i]+w]+PB[j2[i]+w]))$,$PC[2w]=PA[2w]PB[j1[0]+2w]+\sum_{i=1}^{m-1}(PA[i+2w](PB[j1[i]+$

$2w]+PB[j2[i]+2w]))$,于是有:

$$PC[tw]=PA[tw]PB[j1[0]+tw]+\sum_{i=1}^{m-1}(PA[i+tw](PB[j1[i]+tw]+PB[j2[i]+tw]))0\leqslant t\leqslant s-1 \quad (7)$$

由式(7)可得 Ning-Yin 算法,如下所示。

```
input:A,B
output:C
(1)PA=precomputing(A);
(2)PB=precomputing(B);
(3)for k=0 to m-1 step w do
(4)     temp=PA[0] & PB[j1[0]];
(5)     for i=1 to m-1 do
(6)          temp^=PA[i] & (PB[j1[i]]^PB[j2[i]]);
(7)     PC[k]=temp;
(8)     PA+=w;
(9)     PB+=w;
(10)return C=(PC[(s-1)w],PC[(s-2)w],…,PC[w],PC[0])。
```

其中 *precomputing* 是预计算函数,它将域元素 A、B 分别转换成向量 ***PA*** 和 ***PB***。

下面分析 Ning-Yin 算法:算法第(6)步需 $ms=m^2/w$,它与 Rosing 算法一致。附助存储空间为 $4rmw/8=mw/2$(若 $w=32$,即为 $16m$);算法的第(8)、(9)步需$(2m-1)s$ 次的读写内存;*precomputing* 复杂度与具体的方法有关。后面将对 Ning-Yin 算法进行改进,同时提出三种不同的预计算方法。

4 Ning-Yin 方法的改进

4.1 改进方法

通过对式(7)仔细分析得知:(1)结合 Ning-Yin 算法中的预计算方法和 Rosing 算法的计算策略,可减少 Ning-Yin 算法中对 $j1$ 和 $j2$ 数组的访问次数,进一步提高域元素乘法的效率;(2)若令 $k=0,w,2w,\cdots,(s-1)w$,则式(7)可转化为:$PC[k]=PA[k]PB[j1[0]+k]+\sum_{i=1}^{m-1}PA[i+k](PB[j1[i]+k]+PB[j2[i]+k])$,可消除 Ning-Yin 算法的第(8)、(9)步,从而节省$(2m-1)s$ 次的读写内存;(3)通过适当调整第(5)步的循环,可将向量 ***PA*** 从 $2m$ 个存储单元降为 m。由此可得改进算法,如下所示。

```
input:A,B
output:C
(1)PA=precomputing(A);
(2)PB=precomputing(B);
(3)for k=0 to m-1 step w do
(4)     PC[k]=PA[k] & PB[j1[0]+k];
(5)for i=1 to m-1 do
(6)  zero_index=j1[i];
(7)  one_index=j2[i];
(8)  for k=0 to m-1 step w do
(9)     PC[k]^=PA[i+k] &(PB[zero_index+k]^PB[one_index+k]);
(10)return C=(PC[(s-1)],PC[(s-2)w],…,PC[w],PC[0])。
```

改进算法与 Ning-Yin 算法相比:(1)节省了 $2m-1+(s-1)m$ 次的存储访问,其中 $2m-1$ 是节省了第(8)、(9)步得到的结果,$(s-1)m$ 是减少对 $j1[m]$和 $j2[m]$的存储访问;(2)若将第(8)、(9)步的循环分成 2 个,则可节省 m 个存储单元,但性能上会稍有损失。

4.2 预计算方法

precomputing 函数的功能:将域元素 A 转换为式(6)中的 ***PA***。其中 $A=(a_{m-1}a_{m-2}\cdots a_2a_1a_0)$、***PA***$=(PA[2m-1],PA[2m-2],\cdots,PA[1],PA[0])$,$PA[i]=PA[i+m]=(a_i,a_{i+1},\cdots,a_{i+w-1})$,$0\leqslant i\leqslant m-1$。可以有 3 种方法来完成。

方法 1 将域元素 A 反复向右循环移位,第 0 次移位时,从 A 的最低 w 位(A_0)取得数据存放到 $PA[0]$和 $PA[m]$中;第 i 次移位时,同样从最低 w 位取得数据存放到 $PA[i]$和 $PA[i+m]$中,以此类推直到第 $m-1$ 次循环移位后得到 $PA[m-1]$和 $PA[2m-1]$。该方法需 $m-1$ 次循环右移。

方法 2 方法 1 中,循环移位较多,影响了预处理的速度。事实上,第 0 次移位时,由 A 可得 $PA[0]=PA[m]=A[0]$,$PA[w]=PA[w+m]=A[1]$,$\cdots PA[(s-1)w]=PA[(s-1)w+m]=A[s-1]$;第 i 次移位时,可得 $PA[j*w+i+m]=PA[j*w+i]=A[j](0\leqslant j\leqslant s-1)$。该方法需 w 次循环移位和少许的读写内存。

方法 3 为避免前面方法中的循环移位,可事先记录下每个 $PA[i](0\leqslant i\leqslant 2m-1)$在 A 中位于哪两个字及各自所占的位数,然后预计算时再取出相应的数据存放在 $PA[i]$ 中。这些信息可与 lambda 矩阵一起生成,对特定 m,一次生成终生使用。为此可定义结构体:

```
typedef struct{int preb,postb,pre_i,post_i}MULPRE;
MULPRE prec[m];
```

其中,*preb*、*postb* 分别表示 $PA[i]$横跨的前后字,*pre_i* 表示 *preb* 中的高位数、*post_i* 表示 *postb* 中的低位数。则有:$PA[i+m]=PA[i]=(A[prec[i].preb]>>prec[i].pre_i)|(A[prec[i].postb]<<prec[i].post_i)$,其中 $0\leqslant i\leqslant m-1$。

5 测试结果和结论

为检验各算法的运行效果,在 Pentium 1800、Windows XP、VC6.0(方正文祥)环境下进行了一系列实验,其时间是对随机的域元素进行 100 000 次乘法所需时间的平均值。

在 FIPS 186-2[7]中,NIST 推荐了 10 个有限域,其中 5 个是素域,另 5 个是基域。由于并不是所有都存在优化正规基,为使椭圆曲线密码体制与对称密码体制有可比较的安全性,选取的域长度近似于 NIST 推荐的域长度,如表 1 所示。

表 1 NIST 推荐的基域长度

对称密码密钥长度	加密算法	$GF(2^m)$长度 m	Type I ONB	Type II ONB
80	Skipjack	163	162	158
112	Triple-DES	233	226	233
128	AES-128	283	292	281
192	AES-192	409	418	410
256	AES-256	571	562	575

表 2 和表 3 分别给出了 Type I ONB 和 Type II ONB 时乘法运行时间。在均采用第 3 种预计算方法的情况下,与 Ning-Yin 算法相比,本文的改进算法减少了大约 20%的运行时间。

表 2 Type I ONB 下域元素乘法的计算时间

m	Rosing 算法/μs	Ning-Yin 算法/μs	提高/(%)	改进算法/μs	提高/(%)
162	18.640	9.301	50.1	5.578	70.1
226	31.400	14.040	55.3	9.380	70.1
292	47.593	21.512	54.8	14.188	70.2
418	90.000	41.130	54.3	25.470	71.7
562	167.810	71.155	57.6	41.808	75.1

(下转 104 页)