优化扩域上椭圆曲线标量乘的一个新算法

刘 铎, 戴一奇
(清华大学计算机科学与技术系, 北京 100084)

摘 要: 提出了一种优化扩域上椭圆曲线标量乘的新算法. 算法基于 Frobenius 映射和二进制串的逻辑操作. 文中对这个算法给出了细致精确的分析, 而且在此基础上对新算法作了进一步改进. 最后从理论分析和实际仿真两个方面就新算法和传统算法进行了比较. 指出新算法执行时间比传统的 φadic 算法要少 20% 到 40%.

关键词: 密码学; 椭圆曲线; 优化扩域; Frobenius; 标量乘
中图分类号: TN918, TP309 文献标识码: A 文章编号: 0372-2112 (2005) 08-1451-06

A New Method of Scalar Multiplication of Elliptic Curve over OEF

LIU Duo, DAI Yi-qi
(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084, *China*)

Abstract: Elliptic curve cryptosystem is the focus of public cryptology nowadays, for it has many advantages which the RSA lacks. In this paper, a new elliptic curve scalar multiplication algorithm is proposed. The algorithm uses the Frobenius map on optimal extension field (OEF) and the logic operations of binary strings. Based on this algorithm, a new method of computing scalar multiplication of elliptic curve over an OEF is presented. The accurate analysis about it is given. The comparisons of traditional method and the new method are presented. The running time of new method is about 60% ~ 80% of that of the traditional φadic scalar multiplication algorithms of elliptic curve over OEF.

Key words: cryptology; elliptic curve; optimal extension field; Frobenius; scalar multiplication

1 引言

优化扩域[1] (Optimal Extension Field, 以下简称为 OEF) 在软件实现中的使用保证了椭圆曲线密码体制 (elliptic curve cryptosystem) 要快于基于模指数运算的其他公钥密码体制. 另一方面所有的用以计算指数运算的[2] 也都可用于椭圆曲线上的标量乘运算; 而且如果椭圆曲线是定义在扩域上的, 还可以采用 φadic 展开的方法计算标量乘. 标量乘是椭圆曲线密码体制中最重要的运算, 是构成各种密码协议最核心的部分, 例如 EC-DH[3§7.2.1], EC-NR[3§7.2.5§7.2.6], EC-DSA[3§7.2.7§7.2.8]等. Koblitz 提出了在定义在 GF(2)、GF(4), GF(8) 和 GF(16) 上椭圆曲线的 φadic 标量乘的算法[4]; Muller[5]和 Cheon[6]等研究者将其扩展到定义在 GF(2^r) 的椭圆曲线上; Kobliz 还在文献[7] 中将该方法扩展到 GF(3), GF(7) 上; Solinas 提出了 φadic 窗口算法[8]. 但几乎所有的算法都是针对小特征的域的, 应用到定义在 OEF 上的椭圆曲线上则效率很低. 也有一些研究者针对 OEF 上的椭圆曲线标量乘的算法作了一些工作, 如文献[9, 10] 等, 但这两种方法所需要的预计算空间都比较大. 在本文中, 作者则针对 OEF 上椭圆曲线提出了标量乘的一种新算法, 其基于二进制串的交, 较之传统的 φadic 标量乘算法在时间上减少了 20% 到 40%.

本文的组织形式是: 在第二部分中表述了一些基本定义和本文中要用到的记号; 在第三部分中回顾了 OEF 上椭圆曲线的基于 φadic 的传统标量乘算法, 并指出了其主要缺陷; 本文提出的新的标量乘算法及其分析和改进将在第四部分中进行详细描述; 最后, 我们对新的算法和传统的算法在理论和实际两个方面进行了比较.

2 定义与记号

2.1 记号

为行文的简洁与方便, 在本文中将使用如下的记号:

$\lfloor x \rfloor$: 不超过 x 的最大整数, $x \in R$;

$wt(e)$: e 的二进制表示中 1 的个数, $e \in Z$;

W: 处理器的字长;

$GF(q)$: 有 q 个元素的有限域;

$\lg x$: x 的以 2 为底的对数, $x \in R^+$;

$\{0,1\}^W$: 长度为 W 的 0-1 二进制串全体, 在本文中将一个整数的二进制展开也视为 0-1 串;

0^W: 长度为 W 的全 0 串, 即整数 0;

1^W: 长度为 W 的全 1 串, 即整数 2^W-1.

收稿日期: 2004-12-07; 修回日期: 2005-05-01
资金项目: 国家自然科学基金 (No. 90304014)

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

2.2 椭圆曲线[11]

定义在 GF(q)上的椭圆曲线是指方程 $E: y^2 = x^3 + ax + b$, 其中 $a, b \in GF(q)$ 且 $4a^3 + 27b^2 \neq 0$, $\mathrm{char}(GF(q)) > 3$. 其上的点(及一个无穷远点 O)构成一个有限可换群. 由曲线 E 上一个点 P 和一个正整数 $m \in Z$ 计算点 mP 称为标量乘.

2.3 Frobenius 映射

令 $E/GF(p)$ 表示一个定义在 GF(p) 上的非超奇异椭圆曲线. $P = (x, y)$ 是其上一个 GF(p^m) 有理点. Frobenius 映射 φ 定义为 $\varphi: (x, y) \to (x^p, y^p)$. 其是 $E(GF(p^m))$ 的一个自同态且满足 $\varphi^2 - t\varphi + p = 0$, $0 \leqslant t^2 \leqslant 4p$. 是域中 Frobenius 的扩展.

2.4 优化扩域(OEF)

OEF[1] 是满足下述条件的一种有限域 GF(p^m):

- p 是一个大于 3 的素数且 $\lg p$ 小于但接近于 W;
- $p = 2^n \pm c$, 其中 $\lg c \leqslant n/2$;
- 存在 GF(p) 上不可约的多项式 $f(x) = x^m - \omega$, 其中 $\omega \in GF(p)$.

则元素 $a \in GF(p^m)$ 可以表示为: $a = a_{m-1}\alpha^{m-1} + \cdots + a_1\alpha^1 + a_0$, 其中 $a_i \in GF(p)$, $\alpha \in GF(p^m)$ 是 $f(x)$ 的一个根.

2.5 OEF 上的 Frobenius 映射

对于 GF(p^m) 中元素 a, 域中的 Frobenius 映射将 a 映到 a^p:

$$\varphi(a) = a^p = a_{m-1}\alpha^{(m-1)p} + \cdots + a_1\alpha^p + a_0,$$

而由 α 是 $f(x) = 0$ 的根, $\alpha^m = \omega$, 于是

$$\alpha^{ip} = \alpha^{(ip \bmod m)}\omega^{\lfloor ip/m \rfloor}$$

一般 m 都相当小(不超过 10), 可假定 $gcd(m, p) = 1$. 于是 $(i_1 p \bmod m) = (i_2 p \bmod m)$ 当且仅当 $i_1 = i_2$. 因此映射 $\pi(i) = (ip \bmod m)$ 是一个双射, 可以写 a^p 为:

$$a^p = a'_{m-1}\alpha^{m-1} + \cdots + a'_1\alpha + a'_0$$

其中 $a'_{\pi(i)} = a_i\omega^{\lfloor ip/m \rfloor}$.

由于 p, m 和 ω 都是不依赖于 a 的且 m 的值比较小, 因此 $\omega_i = \omega^{\lfloor ip/m \rfloor}$ 可以预计算. 于是得到如下算法, 计算 OEF 中一个元素的 Frobenius 映射的像:

算法 1 计算 OEF 上 Frobenius 映射[9]

输入: $[a_0, a_1, \cdots, a_{m-1}] (= a)$

输出: $[a'_0, a'_1, \cdots, a'_{m-1}] (= a^p)$

步骤 1: 对 $i = 1$ 到 $m-1$, 计算 $b_i = a_i\omega_i$;

步骤 2: 对 $i = 1$ 到 $m-1$, 计算 $a'_{\pi(i)} = b_i$;

步骤 3: $a'_0 = a_0$.

算法 1 的过程只需要 GF(p) 中的 $m-1$ 次乘法. 而一般 GF(p^m) 中的乘法则需要 m^2 次 GF(p) 中的乘法(直接算法). 因此计算 Frobenius 映射是要比乘法快很多的. 从椭圆曲线点运算的算法[3] 可以看到计算椭圆曲线上点的 Frobenius 映射与点运算相比时间复杂度是可以忽略的.

3 φ-adic 标量乘算法

考察标量乘 kP. 传统的算法是基于 k 的二进制表示的, 例如 LR 算法[2 §14.76], RL 算法[2 §14.79], NAF 方法[12 Algorithm IV.5] 等. 而 φ-adic 方法基于事实: $Z \subset Z[\varphi]$. 于是可以将 k 表示为 φ 的多项式: $k = \sum_{i=0}^{m-1} k_i\varphi^i$, 其中 $-p/2 \leqslant k_j \leqslant p/2$(注意到 $\varphi^m = 1$). 由此 kP 可以采用类似二进制算法来计算. 下面给出一个 φ-adic 标量乘的算法[9]:

算法 2. φ-adic 标量乘

输入: k, P, E, t, p

输出: $Q(= kP)$

步骤 1.1: 计算 k 的 φ-adic 展开

步骤 1.1: 对所有 j, 令 $i \leftarrow 0$, $x \leftarrow k$, $y \leftarrow 0$, $u_j \leftarrow 0$;

步骤 1.2: 若 $x = 0$ 且 $y = 0$ 则转到步骤 2;

步骤 1.3: $u_i \leftarrow x \bmod p$, $v \leftarrow (x - u_i)/p$, $x \leftarrow tv + y$, $y \leftarrow -v$;

步骤 1.4: 转到步骤 1.2.

步骤 2: 对 φ-adic 展开进行优化

步骤 2.1: 对所有的 $0 \leqslant i < m$, 计算 $k_i \leftarrow u_i + u_{i+m} + u_{i+2m}$;

步骤 3: 计算标量乘 $S = kP$.

步骤 3.1: $Q \leftarrow k_{m-1}P$;

步骤 3.2: 对 $j = m-2$(递减) 到 0, 执行 $S \leftarrow \varphi(S) + k_jS$

传统的 φ-adic 方法是在算法 2 的基础上使用预计算的. 但是对于 OEF 而言, k_j 取值的范围是非常大的($-p/2 \leqslant k_j \leqslant p/2$), 因而对所有的 k_jP 进行预计算是不现实的. 另一方面, 由于 m 非常小, 每一次标量乘的计算只用到了所有的 k_jP 中很小的一部分, 预计算部分被使用的效率实际上很低.

因此研究能够快速地计算算法 2 中用到的 m 个 k_jP 的算法是非常有意义和必要的.

最一般的算法是用普通的标量乘算法 R-L 算法和 L-R 算法计算 $k_0P, \cdots, k_{m-1}P$, 平均的计算复杂度是: L-R 算法, 先计算 P, $2P$, $4P$, $\cdots$, $2^{m-1}P$, 用 $m-1$ 次点倍运算, 再计算 $k_0P, \cdots, k_{m-1}P$, 平均共用 $mW/2$ 次点加运算; R-L 算法, 计算 $k_0P, \cdots, k_{m-1}P$, 平均共用 $mW/2$ 次点加运算和 $(m-1)W$ 次点倍运算. 因此只考虑较快的 L-R 算法. (如果不使用 φ-adic 方法而是用 NAF 方法的话, 平均用 $mW/3$ 次点加运算和 $mW-1$ 次点倍运算. 虽然点加运算减少为 2/3, 但是点倍运算增加了 $m-1$ 倍, 必定是较 L-R 算法为慢的.)

4 新算法

4.1 位操作的扩展和交错列

为了算法描述的方便, 我们首先将位的逻辑操作进行扩展:

定义 1(串的位操作) 假设 $S1, S2 \in \{0, 1\}^W$, 表示成 $S_1 = (S_{1,W-1}, S_{1,W-2}, \cdots, S_{1,1}, S_{1,0})$, $S_2 = (S_{2,W-1}, S_{2,W-2}, \cdots, S_{2,1}, S_{2,0})$, 则定义:

$$S_1 \vee S_2 = (S_{1,W-1} \vee S_{2,W-1}, \cdots, S_{1,1} \vee S_{2,1}, S_{1,0} \vee S_{2,0})$$

$$S_1 \wedge S_2 = (S_{1,W-1} \wedge S_{2,W-1}, \cdots, S_{1,1} \wedge S_{2,1}, S_{1,0} \wedge S_{2,0})$$

$$\neg S_1 = (\neg S_{1,W-1}, \neg S_{1,W-2}, \cdots, \neg S_{1,1}, \neg S_{1,0})$$

如果 $S_1 \wedge S_2 = (0, \cdots, 0) = 0^W$ 则称 S_1 和 S_2 是不相交的.

定义 2(串与位的位操作) 假设 $S \in \{0, 1\}^W$, 表示成 $S = (S_{W-1}, S_{W-2}, \cdots, S_1, S_0)$, $b \in \{0, 1\}$ 则定义:

$$S \vee b = (S_{W-1} \vee b, S_{W-2} \vee b, \cdots, S_1 \vee b, S_0 \vee b)$$

$$S \wedge b = (S_{W-1} \wedge b, S_{W-2} \wedge b, \cdots, S_1 \wedge b, S_0 \wedge b)$$

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

$S \oplus b = (S_{W-1} \oplus b, S_{W-2} \oplus b, \cdots, S_1 \oplus b, S_0 \oplus b)$

在下面的文章中, 处理的 0– 1 串序列都是具有一种特殊性质的, 因此笔者将其统一定义为交错列, 具体定义如下:

定义 **3**(交错列)　对于集合$\{s_j\}_{1 \leqslant j \leqslant K}$, 如果满足下述性质:

1. $0 \leqslant s_i < 2^W, \forall 1 \leqslant i \leqslant K$;

2. $s_i \wedge s_j = 0^W, \forall 1 \leqslant i < j \leqslant K$;

则称$\{s_j\}_{1 \leqslant j \leqslant K}$是一个长度为 W 的交错列. 在本文中除特殊说明, 所指的交错列长度均为 W.

4.2　算法描述

首先通过一个简单的例子来说明我们提出的新算法的基本想法:

假设要计算 $aP = (10111101)_2P$, $bP = (11101111)_2P$ 和 $cP = (10110110)_2P$, 注意到:

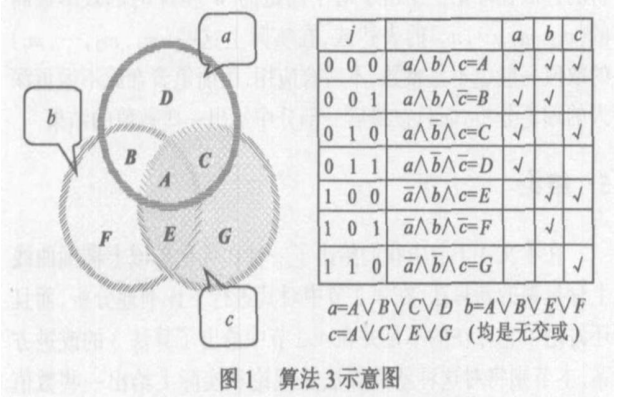
$$aP = (10100100)_2P + (00001001)_2P + (00010000)_2P;$$

$$bP = (10100100)_2P + (00001001)_2P + (00000010)_2P + (01000000)_2P;$$

$$cP = (10100100)_2P + (00010000)_2P + (00000010)_2P.$$

于是可以先计算$(10100100)_2P$, $(00001001)_2P$, $(00010000)_2P$, $(00000010)_2P$, $(01000000)_2P$ 以及$(00000010)_2P$, 而后再计算 aP, bP 和cP.

一般地假设要计算 aP, bP 和 cP, 则按下图所示先计算 AP, BP, CP, DP, EP, FP 和 GP, 再计算 $aP = AP + BP + CP + DP$, $bP = AP + BP + EP + FP$, $cP = AP + CP + EP + GP$ 即可. (图中右上表的j 对应下面的算法 3 中"步骤 8." 的j、是否有"√"对应下面算法 3 中的"步骤 17.")



j		a	b	c
0 0 0	$a \wedge b \wedge c = A$	√	√	√
0 0 1	$a \wedge b \wedge \bar{c} = B$	√	√	
0 1 0	$a \wedge \bar{b} \wedge c = C$	√		√
0 1 1	$a \wedge \bar{b} \wedge \bar{c} = D$	√		
1 0 0	$\bar{a} \wedge b \wedge c = E$		√	√
1 0 1	$\bar{a} \wedge b \wedge \bar{c} = F$		√	
1 1 0	$\bar{a} \wedge \bar{b} \wedge c = G$			√

$a = A \vee B \vee C \vee D$　$b = A \vee B \vee E \vee F$
$c = A \vee C \vee E \vee G$　(均是无交或)

图 1　算法 3 示意图

下面基于这个思想给出本文提出的新算法:

算法 3　计算 $k_0P, k_1P, \cdots, k_{m-1}P$

输入:　$E, P, k_0, \cdots, k_{m-1}, W$

输出:　$Q_0 = k_0P, \cdots, Q_{m-1} = k_{m-1}P$

步骤 1.　$P_0 \leftarrow P$

步骤 2.　对 $i = 1$ 到 $W-1$ 执行

步骤 3.　　$P_i \leftarrow 2P_{i-1}$　　// $P_i = 2^iP$

步骤 4.　结束 i 的循环

步骤 5.　对 $t = 0$ 到 $m-1$ 执行

步骤 6.　$Q_t \leftarrow O$　　// 初始化

步骤 7.　结束 t 的循环

步骤 8.　对 $j = 0$ 到 $2^m - 2$ 执行

步骤 9.　　假设 $j = (j_{m-1}, j_{m-1}, \cdots, j_0)_2$ 是 j 的二进制展开表示

步骤 10.　$a_j \leftarrow 2^W - 1 = (1, \cdots, 1) = 1^W$

步骤 11.　对 $t = 0$ 到 $m-1$ 执行

步骤 12.　　$K_t \leftarrow k_t \oplus j_t$

步骤 13.　　$a_j \leftarrow a_j \wedge K_t$

步骤 14.　结束 t 的循环

步骤 15.　$Q \leftarrow O$

步骤 16.　对 $s = 0$ 到 $W-1$ 执行

步骤 17.　　若 $a_{j,s} = 1$ 则 $Q \leftarrow Q + P_s$

步骤 18.　结束 s 的循环

　　// 到此结束后 $Q = a_jP$

步骤 19.　对 $t = 0$ 到 $m-1$ 执行

步骤 20.　　若 $j_t = 0$ 则 $Q_t \leftarrow Q_t + Q$

步骤 21.　结束 t 的循环

步骤 22. 结束 j 的循环

在此基础上, 可以对算法 2 的步骤 3 进行具体描述:

步骤 3:　计算标量乘 $S = kP$.

步骤 3.1:　使用算法 3 计算 $k_0P, \cdots, k_{m-1}P$;

步骤 3.2:　$Q \leftarrow k_{m-1}P$;

步骤 3.2:　对 $j = m-2$(减) 到 1 执行

　　$S \leftarrow \varphi(S) + k_jS$

4.3　算法分析

4.3.1　正确性证明　首先要证明算法 3 是正确的, 尔后才能进行进一步的分析:

定理 **1**　算法 3 中的输出确实是 $Q_0 = k_0P, Q_1 = k_1P, \cdots, Q_{m-1} = k_{m-1}P$.

证明: 对任一个固定的 $1 \leqslant t \leqslant m$, 在"步骤 6." 初始化 $Q_t = O$; 在"步骤 18." 后, $Q = a_jP$; 在"步骤 20." 中当且仅当 $j_t = 0$ 时 $Q_t \leftarrow Q_t + Q$, 只要证明 $\sum_{0 \leqslant j \leqslant 2^{m-2}, j_t = 0} a_jP = k_tP$ 即可. $j_t = 0$ 时, 在"步骤 12." 中 K_t 总是等于 k_t 的, 而$(j_{m-1}, j_{m-2}, \cdots, j_{t+1}, j_{t-1}, \cdots, j_0)_2$ 取遍$\{0, 1\}^{m-1}$中的所有值, 这时对于 $0 \leqslant j \leqslant 2m - 2$ 且 $j_t = 0$, a_j 取遍

$$\left\{ k_t \wedge \left(\bigwedge_{0 \leqslant i \leqslant m-1, i \neq t} K_i \right) \mid K_i \in \{k_i, \bar{k}_i\} \right\}$$

中所有值.

注意到 $\bigvee_{K_i \in \{k_i, \bar{k}_i\}} \left(\bigwedge_{0 \leqslant i \leqslant m-1, i \neq t} K_i \right) = 1^W$, 且 $\left\{ \bigwedge_{0 \leqslant i \leqslant m-1, i \neq t} K_i \mid K_i \in \{k_i, \bar{k}_i\} \right\}$中任何两个不相同的元素彼此不交. 因此 $\bigvee_{K_i \in \{k_i, \bar{k}_i\}} \left[k_t \wedge \left(\bigwedge_{0 \leqslant i \leqslant m-1, i \neq t} K_i \right) \right] = k_t$, 而且 $\left\{ k_t \wedge \left(\bigwedge_{0 \leqslant i \leqslant m-1, i \neq t} K_i \right) \mid K_i \in \{k_i, \bar{k}_i\} \right\}$中任何两个不同的元素不相交. 故 $Q_t = \sum_{0 \leqslant j \leqslant 2^{m-2}, j_t = 0} a_jP = k_tP$.

4.3.2　点运算的次数　点运算的次数分为点加运算的次数和点倍运算的次数两项, 在下面分别进行讨论.

(1) 点倍次数:

由"步骤 3." 决定, 共 $W-1$ 次.

(2) 点加次数:

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

在算法中点加运算主要分为两个部分:“步骤 16.”和“步骤 19.”:

在“步骤 19.”中, 对于每一个 k_i, 都被分成 2^{m-1} 个部分, 至多需要进行 $2^{m-1}-1$ 次点加运算. 而 $a_j=0^W$ 时在算法的“步骤 20.”中 $Q=O$, 实际上不执行点加运算.

在“步骤 16.”中, 总的点加次数是由 a_j(“步骤 13.”之后)决定的, 而计算每个 $Q=a_jP$(“步骤 14.”~“步骤 17.”)一般地需要 $wt(a_j)-1$ 次点加——只有在 $a_j=0^W$ 时需要 $wt(a_j)=0$ 次点加.

因此首先考虑 $a_j=0^W$ 的个数, 有下面的两个引理(证明都是简单的组合计数, 此处略):

引理 **2** 对于固定的 0-1 串 $S\in\{0,1\}^W$, $wt(S)=k$, $0\leqslant k\leqslant W$, 满足 $\bigcup_{1\leqslant j\leqslant K}S_j=S$ 的有 K 个元素的交错列 $\{s_j\}_{1\leqslant j\leqslant K}$ 的个数为 K^k.

引理 **3** 有 K 个元素的交错列 $\{s_j\}_{1\leqslant j\leqslant K}$ 的个数为 $\sum_{k=0}^{W}C_W^kK^k=(k+1)^W$.

由此, 总体上任取的交错列 $\{a_j\}_{0\leqslant j<2^m-1}$, 使得 $a_j=0^W$ 的 j 的平均个数为

$$\frac{1}{(2^W)^m}\left((2^m-1)(2^m-1)^W\right)=(2^m-1)\cdot\left(1-\frac{1}{2^m}\right)^W.$$

下面考虑 $a_j=0^W$ 对算法 3“步骤 20.”的影响:

引理 **4** 每个 $a_j=0^W$ 在算法 3 的“步骤 20.”中平均减少的点加次数为

$$\frac{1}{2^m-1}\left(\sum_{i=1}^{m}C_m^i\cdot i\right)=\frac{2}{2^m-1}\left(m\sum_{i=1}^{m}C_{m-1}^{i-1}\right)=\frac{m2^{m-1}}{2^m-1}$$

证明: 注意到对于每个 j, $a_j=0^W$ 在算法 3 的“步骤 20.”, 影响的个数是 $m-wt(j)$ 即可.

引理 **5** 假设 $\{X_1,X_2,\cdots,X_n,\cdots\}$ 是一串独立同分布的随机变量, 和 X 的分布相同, 而 Z 是一个正整数取值的随机变量且与 X 序列相独立, 则有 $E\left(\sum_{i=1}^{Z}X_i\right)=EZEX$[13].

综合上面所述, 可以得到:

定理 **6** 总的点加次数平均为 $f(m,W)=(2^m-1)\left[\frac{W}{2^m}-T\right]+m2^{m-1}T-m$,

其中 $T=1-\left(1-\frac{1}{2^m}\right)^W$.

证明: 在“步骤 16.”中点加运算的平均数目是:(由引理 3)

$$\frac{1}{(2^W)^m}\left[\sum_{k=0}^{W}C_W^k(2^m-1)^k(k-(2^m-1))\right]+(2^m-1)\left(1-\frac{1}{2^m}\right)^W=\frac{2^m-1}{2^{Wm}}\left[\frac{W}{2^m}2^{mW}-2^{mW}\right]+(2^m-1)\left(1-\frac{1}{2^m}\right)^W=(2^m-1)\cdot\left[\frac{W}{2^m}-T\right].$$

在“步骤 19.”中点加运算的平均数目是:(由引理 3, 4, 5)

$$m(2^{m-1}-1)-(2^m-1)(1-T)\cdot\frac{m2^{m-1}}{2^m-1}=m2^{m-1}T-m.$$

综合这两部分即可证明定理 6.

通过分析注意到算法 3 和 L-R 算法相比, 点倍数目是一样多的, 点加数目的比例是 $g(m,W)=\frac{f(m,W)}{mW/2}$, 在下面的篇幅中简称之为算法 3 的效率, 其值越小称效率越高.

4.4 算法 3 的改进

注意到 $W\gg 2m$ 时 $T\approx 1$; $W\ll 2^m$ 时 $T\approx W/2^m$, 于是总的(平均)点加次数为

$$\frac{2^m-1}{2^m}[W-2^m]+m2^{m-1}-m\quad(T=1);$$

$$mW/2m\qquad(T=W/2m).$$

故而当 $W\ll 2^m$ 时算法与 L-R 算法的复杂度几乎相等, 新算法在这时优势并不大. 针对这种情况, 笔者对算法进行了改进: 将 m 拆分成 $m=\sum_{i=1}^{t}m_t$, 分成 t 个部分执行算法 3. 例如: $m=6$, $W=32$ 时, $f(6,32)=76.5658$, 而 $2f(3,32)=59.8606$, $2f(3,32)/f(6,32)=0.7818$, 如果先算 k_1P, k_2P, k_3P 再算 k_4P, k_5P, k_6P 则效率提高很多.

记这时的效率为.

$$h((m_1,m_2,\cdots,m_t),W)=\frac{\sum_{i=1}^{t}f(m_i,W)}{\sum_{i=1}^{t}m_i\cdot W/2}$$

则一般地, 将 m 拆分成 $m=\sum_{i=1}^{t}m_t$, 分成 t 个部分执行算法 3 时, 效率

$$h((m_1,m_2,\cdots,m_t),W)=\sum_{i=1}^{t}g(m_i,W)\frac{m_i}{m}.$$

由于 m 一般都很小, 列举完全可以解决, 而且即使用解析的方法在理论上给出了对于固定的 W 和 m 时的效率最高的 $(m_1,m_2,\cdots,m_t)$ 的表达式, 在实际上这些 $(m_1,m_2,\cdots,m_t)$ 的取值一般也不是整数, 不适合应用. 因而笔者在此不做更深入的理论分析, 而只在最后一部分中给出一些数值的结果.

5 结论

在本文 4.1 节中我们给出了一种在优化扩域上椭圆曲线上标量乘的新算法, 在 4.2 节中对其进行了详细地分析, 而且还讨论了它存在的不足并在 4.3 节中给出了算法 3 的改进方案. 本节则将对这种新的算法从理论和实际上给出一些数值的比较.

下面的表 1 中给出了根据 $g(m,W)$ 的表达式和实际的模拟(算法 3 运行 10000 次取平均)得到的效率的数值, 黑体的表示 W 固定时的最高效率, 最右边一列是 m 固定 $W\to\infty$ 时 $g(m,W)$ 的极限值.

从表 1 中看到, 4.3 节中对 $g(m,W)$ 的分析还是相当地精确的(相对误差不超过 2.7%).

而针对 3.3 节中的内容, 我们在下面的表 2 中给出了与 m 和 W 对应的最佳的 $(m_1,m_2,\cdots,m_t)$ 及 $h((m_1,m_2,\cdots,m_t),W)$ 的值.

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

最后, 基于表 2 中的数值结果, 我们给出在计算椭圆曲线的标量乘时的总体时间复杂度比较: 从文献[3]中可以看到点加运算和点倍运算的时间复杂度 Ta 和 Td 有如下关系(近似): $T_a = T_d$(仿射坐标表示)或者 $T_a = 2T_d$(其他坐标表示). 于是用算法 2&3 的改进计算一次标量乘所用的时间是 $(W-1)T_d + h((m_1, m_2, \cdots, m_t), W)T_a$; 传统的算法 2&L-R 算法计算一次标量乘所用的时间是 $(W-1)T_d + (Wm/2)T_a$. 因此可以得到下面的表 3, 其中 T_{LR} 表示在传统的算法 2&L-R 算法计算一次标量乘所用的时间; T_{new} 表示用算法 2&3 的改进计算一次标量乘所用的时间.

表 **1**　算法 **3** 的效率

m \ W		16	32	48	64	96	128	→ ∞
2	理论	0.6869	0.7187	0.7292	0.7344	0.7396	0.7422	3/4
	实验	0.6992	0.7314	0.7464	0.7496	0.7553	0.7577	= 0.75
3	理论	0.6421	0.6235	0.611	0.6042	0.5972	0.5938	7/12
	实验	0.6591	0.6383	0.6277	0.6199	0.6127	0.6092	= 0.5833
4	理论	0.6858	0.6382	0.5962	0.5682	0.5363	0.5195	15/32
	实验	0.6988	0.6542	0.6111	0.5819	0.5482	0.5328	= 0.4688
5	理论	0.7504	0.7157	0.6652	0.6224	0.5611	0.5224	31/80
	实验	0.7565	0.7294	0.6807	0.6355	0.5738	0.5352	= 0.3875
6	理论	0.8017	0.7976	0.7516	0.7235	0.6564	0.6037	21/64
	实验	0.7959	0.8048	0.7727	0.735	0.6694	0.6148	= 0.3281
7	理论	0.8345	0.8571	0.8412	0.8178	0.7681	0.7218	127/448
	实验	0.8118	0.8563	0.8476	0.8273	0.7793	0.7329	= 0.2835
8	理论	0.8534	0.8938	0.8934	0.8834	0.8554	0.8253	255/1024
	实验	0.8059	0.8796	0.8899	0.8851	0.8627	0.8337	= 0.249

表 **2**　改进的算法 **3** 的效率

m \ W	16	32	48	64	96	128
2	(2) 0.6869	(2) 0.7187	(2) 0.7292	(2) 0.7344	(2) 0.7396	(2) 0.7422
3	(3) 0.6421	(3) 0.6235	(3) 0.6110	(3) 0.6042	(3) 0.5972	(3) 0.5938
4	(4) 0.6858	(4) 0.6382	(4) 0.5962	(4) 0.5682	(4) 0.5363	(4) 0.5195
5	(2, 3) 0.6600	(2, 3) 0.6616	(5) 0.6652	(5) 0.6224	(5) 0.5611	(5) 0.5224
6	(3, 3) 0.6421	(3, 3) 0.6235	(3, 3) 0.6110	(3, 3) 0.6042	(3, 3) 0.5972	(3, 3) 0.5938
7	(3, 4) 0.6671	(3, 4) 0.6319	(3, 4) 0.6025	(3, 4) 0.5836	(3, 4) 0.5624	(3, 4) 0.5513
8	(4, 4) 0.6856	(4, 4) 0.6382	(4, 4) 0.5962	(4, 4) 0.5682	(4, 4) 0.5363	(4, 4) 0.5195

表 **3***a*　传统算法与新算法时间复杂度的比较($T_a = T_d$)

m \ W	16		32		48		64		96		128	
	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}
2	31	25.99	63	54	95	82	127	110	191	166	255	222
3	39	30.41	79	60.93	119	90.99	159	121	239	181	319	241.01
4	47	36.95	95	71.84	143	104.24	191	135.73	287	197.97	383	259.99
5	55	41.40	111	83.93	167	126.82	223	162.58	335	229.66	447	294.17
6	63	45.82	127	90.86	191	134.98	255	179.01	383	266.99	511	355.02
7	71	52.36	143	101.77	215	148.22	287	193.73	431	283.97	575	373.98
8	79	58.88	159	112.69	239	161.47	319	208.46	479	300.94	639	392.98

表 **3***b*　传统算法与新算法时间复杂度的比较($T_a = 2T_d$)

m \ W	16		32		48		64		96		128	
	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}
2	47	36.98	95	77.00	143	117.00	191	157.00	287	237.00	383	317.00
3	63	45.82	127	90.86	191	134.98	255	179.01	383	266.99	511	355.02
4	79	58.89	159	112.69	239	161.47	319	208.46	479	300.94	639	392.98
5	95	67.80	191	136.86	287	206.65	383	262.17	575	364.33	767	461.34
6	111	76.64	223	150.71	335	222.97	447	295.01	671	438.99	895	583.04
7	127	89.72	255	172.55	383	249.44	511	324.45	767	472.93	1023	620.96
8	143	102.76	287	194.38	431	275.94	575	353.92	863	506.88	1151	658.97

从中也可以看出, 本文中提出的新算法较之传统的最快　　的 Φadic 标量乘算法, 执行的时间都减少了 20%~40%.

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

参考文献:

[1] Bailey D V, Paar C. Optimal extension fields for fast arithmetic in public-key algorithms[A]. CRYPTO'98, LNCS1462[C]. New York: Springer-Verlag, 1998. 472-485.

[2] Menexes A J, Oorschot P C, Vanstone S A. Handbook of Applied Cryptography[M]. Boca Raton: CRC Press, 1997.

[3] IEEE P1363/D9 Standard specifications for public-key cryptography[S]. 2001, New York, USA, Institute of Electrical and Electronics Engineers, Inc.

[4] N Koblitz. CM-curves with good cryptographic properties[A]. CRYPTO'91, LNCS576[C]. New York: Springer-Verlag, 1992. 279-287.

[5] V Muller. Fast multiplication on elliptic curves over small fields of characteristic two[J]. Journal of Cryptology. 1998, 11: 219-234.

[6] J H Cheon, S Park, S Park, D Kim. Two efficient algorithms for arithmetic of elliptic curves using frobenius map[A]. PKC'98, LNCS1431[C]. New York: Springer, 1998. 195-202.

[7] N Koblitz. An elliptic curve implementation of the finite field digital signature algorithm[A]. CRYPTO98, LNCS1462[C]. New York: Springer-Verlag, 1998. 327-337.

[8] Solinas J A. An Improved algorithm for arithmetic on a family of elliptic curves[A]. CRYPTO'97, LNCS1294[C]. New York: Springer-Verlag, 1997. 282-290.

[9] Kobayashi T. Base-Φ method for elliptic curves over OEF[J]. IEICE Trans Fundamentals, 2001, E83-A(4): 679-686.

[10] LIU Duo, DAI Yiqi. Addition sequence method of scalar multiplication of elliptic curve over OEF[J]. Wuhan University Journal of Natural Sciences, 2005, 10(1): 174-178.

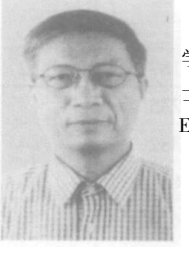
[11] Silverman H. The Arithmetic of Elliptic Curves[M]. GTM106. New York: Springer-Verlag, 1991.

[12] I F Blake, G Seroussi, N P Smart. Elliptic Curves in Cryptography[M]. Cambridge University Press, 1999.

[13] (日)伏见正则. 概率论和随机过程[M]. 李明哲, 译. 北京: 世界图书出版公司, 1997.

作者简介:

刘 铎 男, 1978 年生, 清华大学计算机科学与技术系网络所数据安全研究中心博士研究生, 现进行椭圆曲线密码算法研究工作. E-mail: bat01@mails.tsinghua.edu.cn.



戴一奇 男, 1946 年生, 清华大学计算机科学与技术系网络所数据安全研究中心教授, 博士生导师, 科研方向: 信息安全, 算法设计分析. E-mail: dyq@theory.cs.tsinghua.edu.cn.

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net