

计算椭圆曲线上多标量乘的快速算法

刘 铎 戴一奇

(清华大学计算机科学与技术系 北京 100084)

摘 要 椭圆曲线密码体制最主要的运算就是椭圆曲线上的标量乘和多标量乘,在各种密码协议中起到了核心作用.文中设计了多个整数的一种新的联合带符号二进制表示的编码算法,它每次最多处理相邻的两列,因此在实现上是简单而快速的;在此基础上提出了计算椭圆曲线上多标量乘的一个新算法,并对这个算法进行了分析,最后将新算法和已有多标量乘算法进行了比较,指出新算法在一般情况下($m \geq 3$ 时)效率可提高 7%~15%.

关键词 密码学;椭圆曲线;多标量乘;联合带符号二进制表示

中图法分类号 TP309

A New Algorithm of Elliptic Curve Multi-Scalar Multiplication

LIU Duo DAI Yi-Qi

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084)

Abstract The main operations of elliptic curve cryptosystem are scalar multiplication and multi-scalar multiplication for a pair of integers. In this paper, a new encoding algorithm, which transforms multiple integers into a new kind of signed binary representation of them, is presented. The new encoding algorithm needs only to handle two adjacent columns, and thus is fast and easy to implement. Using this new kind of joint signed binary expressions, a new elliptic curve multi-scalar multiplication algorithm is proposed. The analysis about its time complexity is given, and the comparisons of traditional methods and the new method are also presented, based on which the authors draw the conclusion that the new multi-scalar multiplication algorithm requires about 7% to 15% less running time than the known ones.

Keywords cryptology; elliptic curve; multi-scalar multiplication; joint signed binary representation

1 引 言

1985 年, Koblitz^[1] 和 Miller^[2] 分别独立地提出了椭圆曲线密码体制(Elliptic Curve Cryptography, ECC), 此后一直受到研究者的青睐, 众多数学家和密码学家进行了很多理论算法和实际实现方面的研究, 取得了很大的进展, 并被广泛地应用于实践中, 成为公钥密码学研究的热点之一. 与传统的 RSA 密码体制相比, ECC 可以提供相同的功能; 而

且在相同的安全强度下, ECC 所需要的各项参数(如基础代数系统的大小、密钥的长度等), 都要小得多, 这样就在实现上比 RSA 更简单、更快速和更具有普遍性; 另外一方面, 椭圆曲线具有其一般的代数定义(亏格为 1 的代数曲线), 这使得其远比 RSA 灵活多变, 无论是在密码系统与协议的研究及实现还是在分析与攻击的手段及方法上都有很多新的发展空间和崭新课题.

首先叙述椭圆曲线的概念^[3]. 用 $GF(q)$ 表示有 q 个元素的有限域, 其中 $q = p^m$, p 是一个大于 3 的

素数, m 是一个正整数. 定义在 $GF(q)$ 上的 (标准 Weierstrass 形式) 椭圆曲线为 $E: y^2 = x^3 + ax + b$. 其上的点 (加上一个无穷远点 O) 构成一个有限可换群, 群操作表示为加法. 令 $P = (x_1, y_1) \in E$, 则 $-P = (x_1, -y_1) \in E$, 若 $Q = (x_2, y_2) \in E, Q \neq -P$, 则 $P + Q = (x_3, y_3), x_3 = \lambda^2 - x_1 - x_2, y_3 = \lambda(x_1 - x_3) - y_1$, 其中:

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}, & P \neq Q \\ \frac{3x_1^2 + a}{2y_1}, & P = Q \end{cases}.$$

前者 $P \neq Q$ 的情形称为点加运算; 后者 $P = Q$ 的情形称为点倍运算. 从这里也可以看到, 由点 P 计算点 $-P$ 是很容易的, 可以认为不花费任何时间和空间, 因此计算 $P - Q = P + (-Q)$ 和 $P + Q$ 在复杂度上是完全相同的.

对于给定的椭圆曲线 E 、曲线上一点 P 以及一个正整数 k , 点 kP 的计算称为椭圆曲线上的标量乘; 对于给定的点 P_1, P_2, \dots, P_m 和正整数 k_1, k_2, \dots, k_m , 点 $k_1P_1 + k_2P_2 + \dots + k_mP_m$ 的计算称为多标量乘.

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, 如 EC-DH^[4]、EC-NR^[4]、EC-DSA^[4] 等等; 而多标量乘在 ECC 中也起着重要的作用: 一些基于 ECC 的密码协议需要计算多标量乘, 如 ECDSA 的验证部分需要计算 $kP + lQ$, 如文献[5]提出的多重数字签名; 另外, 标量乘也可转化成多标量乘来计算, 如 Comb 算法^[6]、Lim 和 Lee 算法^[7] 以及一些为抵抗 SCA 而设计的随机化标量乘算法^[8] 等.

虽然可以先分别计算 m 个标量乘 $k_1P_1, k_2P_2, \dots, k_mP_m$, 再计算这 m 个点的和; 但这个方法明显效率不高, 因此研究者提出了同时多标量乘的想法 (simultaneous scalar multiplication), 并设计了各种新的算法: 如直接算法、Shamir 算法^[9]、Shamir-NAF 方法等. Solinas 对于两个整数提出了一种新的带符号二进制表示——联合稀疏形式 (Joint Sparse Form, JSF), 并基于 JSF 给出了一个椭圆曲线上 2-标量乘的新算法^[10].

Solinas 在论文最后一部分“Further Work”中提出了“Three or more Terms”的问题. 本文即是对这个问题的一个解决方案: 提出了多个整数的一种新的联合带符号二进制表示及其编码算法, 它采取了新的算法思路、与 JSF 编码算法不同 (这可以从算法 4 和算法 6 后面的例 4 和例 5 看出, 即使对两个整数, 编码方式也是完全不同的); 这个算法每次

最多处理两列, 因此在实现上是简单而快速的.

在此基础上, 本文提出了计算多标量乘的一种新算法, 并在本文的最后一部分中给出了已有的各种多标量乘算法和新算法的性能比较. 可看出在 $m \geq 3$ 时, 本文提出的多标量乘新算法较已有的各种算法, 时间复杂度减少 7~15%; 实际上, 本文也主要是为解决 $m \geq 3$ 的情形, $m = 1$ 时, 新算法即是传统的 NAF 标量乘方法; $m = 2$ 时, 新算法与 Solinas 算法复杂度相当.

2 多标量乘的传统算法

假定 k_1, k_2, \dots, k_m 可能取值的范围为 $0 \leq k_1, k_2, \dots, k_m < 2^L$. k_1, k_2, \dots, k_m 各自的二进制展开形式为 $(k_{i,L-1}, k_{i,L-2}, \dots, k_{i,1}, k_{i,0})_2$, 满足 $k_i = \sum_{j=0}^{L-1} k_{i,j} 2^j$, 其中 $k_{i,j} \in \{0, 1\}, 1 \leq i \leq m, 0 \leq j \leq L-1$. 再记 $I_j = (k_{1,j}, k_{2,j}, \dots, k_{m,j})^T$, 即 $(k_1, k_2, \dots, k_m)^T$ 的二进制表示的第 j 列. 另外, 为下文中算法描述的简洁, 定义 $(P_1, P_2, \dots, P_m) \cdot (b_1, b_2, \dots, b_m)^T = \sum_{j=1}^m b_j P_j$, 其中 $b_j \in \{0, 1, -1\}, 1 \leq j \leq m$.

2.1 直接算法

简单地讲, 这个算法“差不多”就是分别计算 m 个标量乘; 区别在于其采用了同时多标量乘的技巧, 即同时进行点倍运算. 这是最简单, 却也最慢的算法.

算法 1.

输入: $GF(q), E, P, (k_{1,L-1}, k_{1,L-2}, \dots, k_{1,1}, k_{1,0})_2, 1 \leq i \leq m$

输出: $k_1P_1 + k_2P_2 + \dots + k_mP_m$

1. $Q \leftarrow O$
2. For $j = L-1$ downto 0
3. $Q \leftarrow 2Q$
4. For $i = 1$ to m
5. If $k_{i,j} = 1$ then $Q \leftarrow Q + P_i$
6. Output Q

一般性地假设 $k_{i,j} \in \{0, 1\}$ 取值为 0 和取值为 1 是等概率的, 于是算法 1 平均要进行的点加次数是 $mL/2$, 点倍次数是 L .

例 1. 计算 $53P_1 + 102P_2$ (以下都用 $[a, b]$ 表示 $aP_1 + bP_2$), 共用 6 次点倍运算和 8 次点加运算.

53	(0	1	1	0	1	0	1)
102	(1	1	0	0	1	1	0)
×2	[0,0]	[0,2]	[2,6]	[6,12]	[12,24]	[26,50]	[52,102]
P_1		[1,2]	[3,6]		[13,24]		[53,102]
P_2	[0,1]	[1,3]			[13,25]	[26,51]	

2.2 Shamir 算法^[9]

Shamir 算法的最主要想法是对于 $(P_1, P_2, \dots, P_m) \cdot (k_{1,j}, k_{2,j}, \dots, k_{m,j})^T$ 进行预计算和存储。

算法 2.

输入: $GF(q), E, P, (k_{i,L-1}, k_{i,L-2}, \dots, k_{i,1}, k_{i,0})_2, 1 \leq i \leq m$

输出: $k_1 P_1 + k_2 P_2 + \dots + k_m P_m$

1. 预计算过程:
2. For $i=1$ to 2^m-1
3. Let $i=(i_1, i_2, \dots, i_m)_2$
4. Compute and store $Q_i=(P_1, P_2, \dots, P_m) \cdot (i_1, i_2, \dots, i_m)^T$
5. 主计算过程:
6. $Q \leftarrow O$
7. For $j=L-1$ downto 0
8. $Q \leftarrow 2Q$
9. If $I_j \neq 0$ then $Q \leftarrow Q + Q_{I_j}$
10. Output Q

同样地, 算法 2 需要 L 次点倍运算, 而其点加运算的平均次数则取决于 $I_j \neq (0, \dots, 0)^T$ (下文中, 在不引起混淆的前提下, 也简记为 $I_j \neq \mathbf{0}$) 的概率, 显然有 $P(I_j = \mathbf{0}) = 1/2^m$, 于是点加运算的平均次数为 $(1 - 1/2^m)L$.

例 2. 计算 $53P_1 + 102P_2$, 共用 6 次点倍运算和 6 次点加运算。

53	(0	1	1	0	1	0	1)
102	(1	1	0	0	1	1	0)
×2	[0,0]	[0,2]	[2,6]	[6,12]	[12,24]	[26,50]	[52,102]
P_1			[3,6]				[53,102]
P_2	[0,1]					[26,51]	
$P_1 + P_2$		[1,3]			[13,25]		

2.3 Shamir-NAF 算法

椭圆曲线上点的减法与加法具有同等时间复杂度, 因此可以使用整数的带符号二进制 (signed binary) 表示方法. 即将一个正整数 k 表示成为 $(K_{t-1}, K_{t-2}, \dots, K_1, K_0)_S$, 使得 $k = \sum_{l=0}^{t-1} K_l 2^l$, 其中 $K_l \in \{0, 1, -1\}$ (为简洁起见, 用 $\bar{1}$ 表示 -1), $0 \leq l \leq t-1$.

与正整数的二进制表示方法不同, 带符号的二进制表示具有以下特性:

- (1) 一个正整数的带符号二进制表示方法不是唯一的, 例如 $5 = (1, 0, 1)_S = (1, 1, \bar{1})_S$;
- (2) 如果令 $k = (k_{L-1}, k_{L-2}, \dots, k_1, k_0)_2$ 是 k 的二进制表示, 且 $k_{L-1} \neq 0$, 而 $(K_{t-1}, K_{t-2}, \dots, K_1, K_0)_S$ 是 k 的一个带符号的二进制表示方法, 则明显地有 $t \geq L$;
- (3) 一个正整数的所有带符号二进制表示方法

的长度 (即 t) 没有上界——因为对于任意的 $T > 1$, 都可以将 1 表示为 $\mathbf{1} = (K_{T-1}, K_{T-2}, \dots, K_0)$, 其中 $K_{T-1} = 1, K_j = \bar{1}, 0 \leq j \leq T-2$.

因此, 提出了正整数的非邻接形式 (Non Adjacent Form, NAF) 表示方法^[11]: $k = (K_{t-1}, K_{t-2}, \dots, K_1, K_0)_{\text{NAF}}$, 使得 $k = \sum_{l=0}^{t-1} K_l 2^l, K_j \cdot K_{j+1} = 0$, 其中 $K_l \in \{0, 1, -1\}, 0 \leq l \leq t-1, 0 \leq j < t-1$. NAF 表示方法主要具有以下 3 条性质: $t \leq L + 1$; 概率 $P(K_j = 0) = 2/3$; 在 k 的所有带符号的二进制表示方法中, NAF 表示方法中的“1”最少. 这使得 NAF 表示方法是计算标量乘的最常用的带符号二进制表示.

在引入了正整数的 NAF 表示方法后, 可以给出 Shamir_NAF 算法:

假设 $k_i = (K_{i,L}, K_{i,L-1}, \dots, K_{i,1}, K_{i,0})_{\text{NAF}}, 1 \leq i \leq m$. 在不引起混淆的前提下, 仍然记 $I_j = (K_{1,j}, K_{2,j}, \dots, K_{m,j})^T$, 即 $(k_1, k_2, \dots, k_m)^T$ 的 NAF 表示的第 j 列, $0 \leq j \leq L+1$.

算法 3.

输入: $GF(q), E, P, (K_{i,L}, K_{i,L-1}, \dots, K_{i,1}, K_{i,0})_{\text{NAF}}, 1 \leq i \leq m$

输出: $k_1 P_1 + k_2 P_2 + \dots + k_m P_m$

1. 预计算:
2. For all $T = (T_1, T_2, \dots, T_m), T_l \in \{0, 1, \bar{1}\}, 1 \leq l \leq m$
3. Compute and store $Q_T = (P_1, P_2, \dots, P_m) \cdot (T_1, T_2, \dots, T_m)^T$
4. 主循环
5. $Q \leftarrow O$
6. For $j=L$ downto 0
7. $Q \leftarrow 2Q$
8. If $I_j \neq (0, 0, \dots, 0)$ then $Q \leftarrow Q + Q_{I_j}$
9. Output Q

算法 3 需要 $L+1$ 次点倍运算, 而 $I_j = \mathbf{0}$ 的概率为 $P(I_j = \mathbf{0}) = (2/3)^m$, 于是点加运算的平均次数为 $(1 - (2/3)^m)L$. 与算法 2 相比, 点倍次数至多增加一次, 而点加次数则大幅度减少.

例 3. 计算 $53P_1 + 102P_2$, 共用 6 次点倍运算和 6 次点加运算。

53	(0	1	0	$\bar{1}$	0	1	0	1)
102	(1	0	$\bar{1}$	0	1	0	$\bar{1}$	0)
×2	[0,0]	[0,2]	[2,4]	[4,6]	[6,12]	[12,26]	[26,52]	[52,102]
P_1			[1,2]		[3,6]		[13,26]	
P_2	[0,1]					[6,13]		[26,51]
$P_1 + P_2$								
$P_1 - P_2$								

2.4 Solinas 算法

对于 $m = 2$ 的情形, Solinas 定义了 (k_1, k_2) “联合稀疏形式”^[10], 并且给出了生成算法. 这也是一种

带符号的二进制表示方法, $k_i = (K_{i,t-1}, K_{i,t-2}, \dots, K_{i,1}, K_{i,0})_{\text{Sol}}, i=0, 1$. Solinas 证明: 在所有的 (k_1, k_2) 的带符号的二进制表示方法中, “联合稀疏形式” 的全“0”列的数目是最多的, 列 $(K_{i,0}, K_{j,0})^T = (0, 0)^T$ 的概率为 1/2. 使用 Solinas 表示法计算 $k_1 P_1 + k_2 P_2$ 的算法如下 (Solinas 的生成算法过于长, 且仅适合 $m=2$ 的情形, 因此在此文中不予尽录).

算法 4.

输入: $GF(q), E, P, (K_{i,t-1}, K_{i,t-2}, \dots, K_{i,1}, K_{i,0})_{\text{Sol}}, i=0, 1$

输出: $k_1 P_1 + k_2 P_2$

- 1. 预计算:
- 2. Compute and store $Q_{(1,1)} = P_1 + P_2, Q_{(1,\bar{1})} = P_1 - P_2, Q_{(\bar{1},\bar{1})} = -Q_{(1,1)}, Q_{(\bar{1},1)} = -Q_{(1,\bar{1})}$
- 3. 主循环.
- 4. $Q \leftarrow O$
- 5. For $j=t-1$ downto 0
- 6. $Q \leftarrow 2Q$
- 7. If $I_j \neq (0, 0)$ then $Q \leftarrow Q + Q_{I_j}$
- 8. Output Q

算法 4 需要 L 或 $L+1$ 次点倍运算, 平均 $L/2$ 次点加运算.

例 4. 计算 $53P_1 + 102P_2$, 共用 7 次点倍运算和 5 次点加运算.

53	(0	1	0	0	$\bar{1}$	0	$\bar{1}$	$\bar{1}$)
102	(1	0	0	$\bar{1}$	$\bar{1}$	0	$\bar{1}$	0)
$\times 2$	[0,0]	[0,2]	[2,4]	[4,8]	[8,14]	[14,26]	[28,52]	[54,102]
P_1	[1,2]		[53,102]					
P_2	[0,1]		[4,7]					
P_1+P_2					[7,13]			
P_1-P_2					[27,51]			

3 椭圆曲线多标量乘新算法

注意到前面所述算法中, 点倍运算运行的次数相差很小, 而算法复杂度的主要差异都在于点加运算的次数, 因此如果能大幅度减少点加运算的次数便能有效地提高算法的执行效率, 而所需点加运算的次数则是由列 $I_j \neq \mathbf{0}$ 的概率所决定的.

基于这种思想, 本文提出了一种新的多个整数 k_1, k_2, \dots, k_m 的特定的联合带符号二进制表示——(为叙述方便, 将其命名为 T 形式, 对于某一个固定的 j, k_j 的 T 形式是依赖于 k_1, k_2, \dots, k_m 全体的) 使得其 $I_j = \mathbf{0}$ 的概率尽量大. 算法 5 给出了 T 形式带符号二进制串的生成方法.

算法 5.

输入: k_1, k_2, \dots, k_m

输出: k_j 的 T 形式表示 $(b_{j,t-1}, b_{j,t-2}, \dots, b_{j,1}, b_{j,0})_T, 1 \leq j \leq m$

- 1. $l \leftarrow 0, k_j^{(0)} = k_j, 1 \leq j \leq m$
- 2. While not all $k_j^{(l)} = 0$ do
- 3. $flag \leftarrow 0$
- 4. For $j=1$ to m do
- 5. $a_{0,j} \leftarrow k_j^{(l)} \pmod{2}$
- 6. $a_{1,j} \leftarrow (k_j^{(l)} - a_{0,j})/2 \pmod{2}$
- 7. If $a_{0,j} = 0$ then $flag \leftarrow flag \vee a_{1,j}$
- 8. For $j=1$ to m do
- 9. If $a_{0,j} = 0$ then $b_{j,l} \leftarrow 0, addon_j \leftarrow 0$
- 10. If $a_{0,j} = 1$ then
- 11. $b_{j,l} \leftarrow 1 - 2 \times (a_{1,j} \oplus flag)$
- 12. $addon_j \leftarrow a_{1,j} \oplus flag$
- 13. $k_j^{(l+1)} \leftarrow \lfloor (k_j^{(l)} + addon_j)/2 \rfloor$
- 14. $l \leftarrow l+1$
- 15. Output $(b_{j,t-1}, b_{j,t-2}, \dots, b_{j,1}, b_{j,0}), 1 \leq j \leq m$

图 1 即表示算法 5 中步 2~13 的操作.

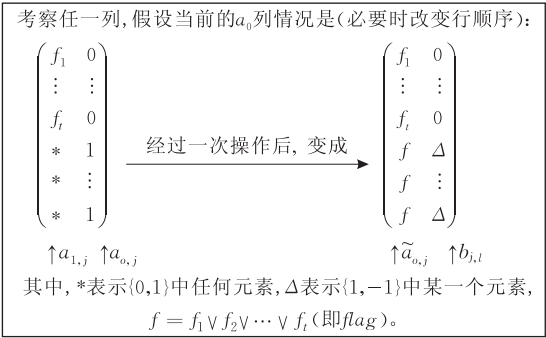


图 1 算法 5 图示

在得到了 k_1, k_2, \dots, k_m 的 T 形式表示后, 可以由此计算 $k_1 P_1 + k_2 P_2 + \dots + k_m P_m$ (命 $I_j = (b_{1,j}, b_{2,j}, \dots, b_{m,j})^T$).

算法 6.

输入: $GF(q), E, P, (b_{i,t-1}, b_{i,t-2}, \dots, b_{i,1}, b_{i,0})_T, 1 \leq i \leq m$

输出: $k_1 P_1 + k_2 P_2 + \dots + k_m P_m$

- 1. 预计算:
- 2. For all $T = (T_1, T_2, \dots, T_m), T_l \in \{0, 1, \bar{1}\}, 1 \leq l \leq m$
- 3. Compute and store $Q_T = (P_1, P_2, \dots, P_m) \cdot (T_1, T_2, \dots, T_m)^T$
- 4. 主循环.
- 5. $Q \leftarrow O$
- 6. For $j=t-1$ downto 0
- 7. $Q \leftarrow 2Q$
- 8. If $I_j \neq (0, 0, \dots, 0)$ then $Q \leftarrow Q + Q_{I_j}$
- 9. Output Q

例 5. 计算 $53P_1 + 102P_2$, 共用 6 次点倍运算和 5 次点加运算.

53	(1	0	0	$\bar{1}$	0	$\bar{1}$	$\bar{1}$)
102	(1	1	0	1	0	$\bar{1}$	0)
$\times 2$	[0,0]	[2,2]	[4,6]	[8,12]	[14,26]	[28,52]	[54,102]
P_1							[53,102]
P_2		[2,3]					
P_1+P_2	[1,1]					[27,51]	
P_1-P_2				[7,13]			

注. 比 Solinas 算法少一次点倍运算.

4 新算法的分析

4.1 稀疏性的说明

首先要指出的是,对于多标量乘,算法都是按照列来进行的,因此在行的方向上是否“稀疏”在复杂度上实际意义并不大.故而虽然可以类似文献[10]对 T 形式定义扩展的“稀疏”性,但是在本文中我们对其不进行过多描述.

注意到在图 1 中,除非 $f_1=f_2=\cdots=f_t=0$ (这时 $f=0$,下一列全 0),否则总有 $f=1, a_{1,j}$ 列中 1 的个数严格增加,而且,若 $t=0$,则下一列一定是全 0.因此在转化成 T 形式后,每相邻的 $m+1$ 列中一定有一列是全“0”的,即对于所有的 $0\leq j\leq l-m-1$,必存在 $s\in\{0,1,\cdots,m\}$,使得 $b_{1,j+s}=b_{2,j+s}=\cdots=b_{m,j+s}=0$.

4.2 算法正确性证明

由于算法 6 是基于算法 5 的,因此我们必须首先证明算法 5 的正确性,才能分析其性能,下面的定理 1 保证了这一点.

定理 1. 算法 5 一定在有限步骤后中止,而且其输出 $(b_{j,l-1}, b_{j,l-2}, \cdots, b_{j,1}, b_{j,0})_T$ 确是 k_j 的一个带符号二进制表示, $1\leq j\leq m$.

证明. 分 3 个部分来证明这个定理.

(1) 算法终止

不妨假设 $k_1=\max\{k_j:1\leq j\leq m\}, k_1>1$ ($k_1=1$ 的情形是平凡的,定理 1 显然成立).

在算法 5 的步 13 中,有 $k_j^{(l+1)}=\lfloor (k_j^{(l)}+addon_j)/2\rfloor\leq (k_j^{(l)}+1)/2$,经过简单的计算可以得到 $k_j^{(m)}\leq (k_j-1)/2^m+1$,取 $L=\lfloor\log_2 k_1\rfloor+1$ (即 k_1 二进制表示的长度),则 $k_j^{(L)}<2$,即算法 5 在 $l=L$ 时一定有 $k_j^{(l)}\leq 1, 1\leq j\leq m$.这时算法 5 中步 6 得到的结果 $a_{1,j}=0, 1\leq j\leq m$,算法至多再进行一次循环一定终止.

(2) 输出是 $\{0,1,\bar{1}\}$

从步 11 中即可看出 $b_{j,l}\in\{0,1,\bar{1}\}$.

(3) 正确性

即要证明:

$$2\lfloor (k_j^{(l)}+addon_j)/2\rfloor+b_{j,l}=k_j^{(l)} \quad (*)$$
 k_j 是偶数时, $a_{0,j}=b_{j,l}=0, 2\lfloor (k_j^{(l)}+addon_j)/2\rfloor+b_{j,l}=2\cdot (k_j^{(l)}/2)+0=k_j^{(l)}$,式(*)成立; k_j 是奇数时, $a_{0,j}=1, b_{j,l}=1-2\cdot (a_{1,j}\oplus flag)$,式(*)左边可以写成 $2\lfloor (k_j^{(l)}+(a_{1,j}\oplus flag)\cdot a_{0,j})/2\rfloor+1-2\cdot (a_{1,j}\oplus flag)$.当 $a_{1,j}\oplus flag=0$ 时,式(*)左边为 $2\cdot (k_j^{(l)}-1)/2+1=k_j^{(l)}$;而 $a_{1,j}\oplus flag=1$ 时,式(*)左边为 $2\cdot (k_j^{(l)}+1)/2+1-2=k_j^{(l)}$.故这时式(*)亦成立. 证毕.

4.3 算法 6 点倍次数的估计

在分析算法 6 的复杂度时,要将点加运算的次数和点倍运算的次数分开计算.而要计算点倍运算的次数,就必须估计在算法 5 输出的 T 形式表示方法的宽度,即列数.

定理 2. k_1, k_2, \cdots, k_m 的 T 形式表示方法比二进制表示方法至多多一列,也即算法 6 在点倍次数上至多比算法 1 多 1.

证明. 不妨假设 $k_1=\max\{k_j:1\leq j\leq m\}, k_1>1, L=\lfloor\log_2 k_1\rfloor+1$ ($k_1=1$ 的情形是平凡的,定理 2 显然成立).再假定 k_1, k_2, \cdots, k_m 的 T 形式表示方法共有 l 列,从定理 1 的证明过程中(第一部分)即可看到 $l\leq L+1$;另外,从 2.3 节对于带符号二进制表示方法的说明中有 $l\geq L$. 证毕.

定理 2 说明算法 6 中点倍运算的次数至多为 $L+1$,事实上,算法 6 中的“3”可以改为:“For $j=L$ downto 0”.

4.4 算法 6 点加次数的估计

算法 6 要进行的点加运算的次数是由算法 5 输出结果中 $I_j\neq 0$ 的概率决定的.

可以将图 1 所描述的过程视为一个马尔可夫过程,用 S_j 表示一列中有 j 个“1”的状态,则图 1 左边状态为 S_{m-t} . 转移概率是

$$P(S_k | S_{m-t}) = \begin{cases} 1/2^t, & k=0 \\ 0, & 1\leq k\leq m-t \\ \binom{t}{k+t-m}/2^t, & k>m-t \end{cases}.$$

由是,可以写出转移概率矩阵 $\mathbf{G}_m=(P(S_i | S_j))_{(m+1)\times(m+1)}$. 例如:

$$\mathbf{G}_2 = \begin{bmatrix} 1/4 & 1/2 & 1 \\ 2/4 & 0 & 0 \\ 1/4 & 1/2 & 0 \end{bmatrix}, \mathbf{G}_3 = \begin{bmatrix} 1/8 & 1/4 & 1/2 & 1 \\ 3/8 & 0 & 0 & 0 \\ 3/8 & 2/4 & 0 & 0 \\ 1/8 & 1/4 & 1/2 & 0 \end{bmatrix},$$
$$\mathbf{G}_4 = \begin{bmatrix} 1/16 & 1/8 & 1/4 & 1/2 & 1 \\ 4/16 & 0 & 0 & 0 & 0 \\ 6/16 & 3/8 & 0 & 0 & 0 \\ 4/16 & 3/8 & 2/4 & 0 & 0 \\ 1/16 & 1/8 & 1/4 & 1/2 & 0 \end{bmatrix}.$$

可以证明该马尔可夫过程存在极限分布 $[P(S_0), P(S_1), \dots, P(S_m)]$, 其是线性方程组:

$$\begin{bmatrix} \mathbf{G}_m - \mathbf{I}_{m+1} \\ \vdots \\ 1 \quad \dots \quad 1 \end{bmatrix} \times [P(S_0), P(S_1), \dots, P(S_m)]^T = \begin{pmatrix} 0 & \dots & 0 & 1 \end{pmatrix}^T$$

的解, 其中 \mathbf{I}_{m+1} 是 $m+1$ 阶单位方阵. 解方程所得的 $P(S_0)$ 即是当 L 充分长时, 全“0”列出现的概率, 于是算法 6 平均要进行的点加运算次数即为 $(1-P(S_0))L$.

在文章的最后一部分中对于 $m \leq 8$ 的情形给出了具体值(事实上, 用带符号二进制表示方法时, 每一个 m 维的向量需要预计算的点数是 $(3^m - 1)/2$, 因而 $m > 6$ 的情形在实际应用中是不适合的).

5 结 论

在本文中, 主要给出了一种 k_1, k_2, \dots, k_m 的带符号二进制表示方法, 使得其全“0”列的数目尽可能多; 并且在此基础上给出了计算多标量乘的一个新算法. 下面将对于新算法和已有的各种算法进行比较(在 $m=1$ 时, 算法 5 即是传统的 NAF 算法).

表 1 中给出了当 $\{k_1, k_2, \dots, k_m\}$ 采用二进制表示法、NAF 表示法和本文提出的 T 形式表示方法时全“0”列出现的概率的比较(宽度由于最多相差 1, 与 k_j 的宽度(即 L)相比可忽略不计, 因此认为宽度都是相同的).

椭圆曲线点加运算和点倍运算的时间复杂度 T_A 和 T_D 有如下关系(近似)^[4]: $T_A = T_D$ (仿射坐标

表示)或者 $T_A = 2T_D$ (其它坐标表示). 而无论 JSF 编码算法或是 T 形式编码算法——即算法 5, 都只需要加法、模 4 或模 8 运算、比特位的位操作和整数的移位运算; 相比之下, 椭圆曲线上的点运算则至少需要 4 次域中元素的乘法^[4], 当域比较大时, 其比加法、模 4 或模 8 运算、比特位的位操作和整数的移位运算要慢得多. 因此在算法复杂度分析和比较的过程中, JSF 编码算法或是 T 形式编码算法的时间复杂度可以忽略不计. 由此可以得到表 2.

表 1 不同表示方法全 0 列的概率比较

m	一般表示方法 (1/2) ^{m}	NAF 表示方法 (2/3) ^{m}	算法 5 的输出 结果 $P(S_0)$
1	0.5	0.667	0.667
2	0.25	0.444	0.500
3	0.125	0.296	0.410
4	0.063	0.198	0.358
5	0.031	0.132	0.328
6	0.016	0.088	0.300
7	0.008	0.059	0.283
8	0.004	0.039	0.269

表 2 计算多标量乘的算法时间复杂度比较(1)

	(平均)总时间复杂度 T	$T_1 = T/(LT_D)$	$T_2 = T/(LT_D)$
		($T_A = T_D$)	($T_A = 2T_D$)
算法 1	$LT_D + (mL/2)T_A$	$1 + m/2$	$1 + m$
算法 2	$(L+1)T_D + (1-1/2^m)LT_A$	$2 - 1/2^m$	$3 - 1/2^{m-1}$
算法 3	$(L+1)T_D + (1-(2/3)^m)LT_A$	$2 - (2/3)^m$	$3 - 2 \cdot (2/3)^m$
算法 6	$(L+1)T_D + (1-P(S_0))LT_A$	$2 - P(S_0)$	$3 - 2 \cdot P(S_0)$

在下面的表 3 中即给出了在不同的 m 值下, 使用各种算法计算多标量乘时的时间复杂度具体值的比较(以点倍运算的时间复杂度为一个单位时间), 其中 $T_{i,j}$ 表示使用算法 j 时, T_i 的取值.

表 3 计算多标量乘的算法时间复杂度比较(2)

m	T_1					T_2				
	$T_{1,1}$	$T_{1,2}$	$T_{1,3}$	$T_{1,6}$	$T_{1,6}/T_{1,3}/\%$	$T_{2,1}$	$T_{2,2}$	$T_{2,3}$	$T_{2,6}$	$T_{2,6}/T_{2,3}/\%$
1	1.500	1.500	1.333	1.333	100.0	2.000	2.000	1.667	1.667	100.0
2	2.000	1.750	1.556	1.500	96.4	3.000	2.500	2.111	2.000	94.7
3	2.500	1.875	1.704	1.590	93.3	4.000	2.750	2.407	2.180	90.6
4	3.000	1.938	1.802	1.642	91.1	5.000	2.875	2.605	2.284	87.7
5	3.500	1.969	1.868	1.672	89.5	6.000	2.938	2.737	2.344	85.6
6	4.000	1.984	1.912	1.700	88.9	7.000	2.969	2.824	2.400	85.0
7	4.500	1.992	1.941	1.717	88.5	8.000	2.984	2.883	2.434	84.4
8	5.000	1.996	1.961	1.731	88.3	9.000	2.992	2.922	2.462	84.3

图 2 与图 3 中给出了在 m 取不同值时各种算法的时间复杂度比较(为了图表清晰, 不考虑过于慢的算法 1).

可以看出, 本文提出的算法与已有的各种算法相比, 是有相当的优势的: 在 $m \geq 3$ 时, 多标量乘的时间复杂度减少了 7%~15%.

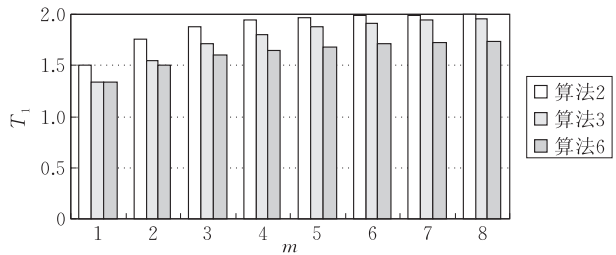


图 2 T_1

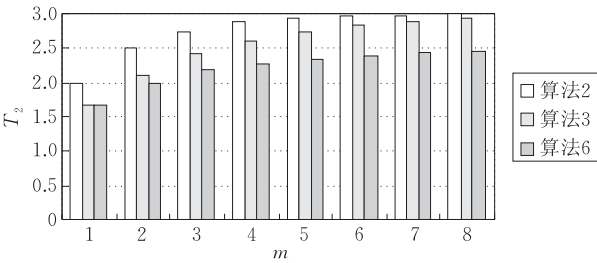


图 3 T_2

参 考 文 献

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

[2] Miller V. Uses of elliptic curve in cryptography//Proceedings of the CRYPTO'85, LNCS218. New York: Springer Verlag, 1986: 417-426

[3] Silverman H. The Arithmetic of Elliptic Curves. GTM106. New York: Springer-Verlag, 1986

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

[5] Chen T-S, Huang K-H, Chung Yu-Fang. Digital multi-signature scheme based on the elliptic curve cryptosystem. Journal of Computer Science and Technology, 19(4): 57-inside back cover

[6] Menexes A J, Oorschot P C, Vanstone S A. Handbook of Applied Cryptography. Boca Raton: CRC Press, c1997

[7] Lim C H, Lee P J. More flexible exponentiation with pre-computation//Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology, LNCS389. New York: Springer Verlag, 1994: 95-107

[8] Clavier C, Joye M. Universal exponentiation algorithm//Proceedings of the CHES2001, LNCS2162. New York: Springer Verlag, 2001: 300-308

[9] ElGamal T. A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory, 1985, IT-31(4): 469-472

[10] Solinas J. Low-weight binary representations for pairs of integers. Centre for Applied Cryptographic Research, University of Waterloo, Ontario, Canada: Technical Report CORR 2001-41, 2001

[11] Morain F, Olivos J. Speeding up the computations on an elliptic curve using addition-subtraction chains. Theoretical Informatics and Applications, 1990, 24(6): 531-543



LIU Duo, born in 1978, Ph. D. . His current research interests include network security, elliptic curve cryptology, combinational algorithm, etc.

DAI Yi-Qi, born in 1946, professor, Ph. D. supervisor. His main research interests are information security, network security, cryptology and combinational algorithm.

Background

Elliptic curve cryptosystem (ECC) was proposed independently in 1985 by Koblitz and Miller. Recently, many efficient algorithms and implementation techniques related to ECC have been proposed.

The basic operation of ECC is scalar multiplication, which uses a given point P and an integer k to compute kP . The multi-scalar multiplication, which computes the point $k_1P_1 + k_2P_2 + \dots + k_mP_m$ for the given point P_1, P_2, \dots, P_m and positive integers k_1, k_2, \dots, k_m , also plays an important role in ECC, for it can be used in the following three cases.

- (1) In some protocols, such as the verify step of ECDSA (elliptic curve digital signature algorithm), multi-scalar multiplication on elliptic curve is required;
- (2) In some fast scalar multiplication algorithms, the computation of scalar multiplication is converted to the computation of multi-scalar multiplication;
- (3) In some countermeasure designs of scalar multiplication to resist Simple Power Analysis (SPA), an integer is decomposed into multiple integers, and thus scalar multiplication can be securely implemented as multi-scalar multiplication.

The most trivial method of computing the point $k_1P_1 + k_2P_2 + \dots + k_mP_m$ is as follows: compute the m scalar multiplications $k_1P_1, k_2P_2, \dots, k_mP_m$ respectively, and then sum the m points up. However, the efficiency of this method is low. Thereupon, the trick named as simultaneous scalar multiplication was proposed, based on which some new multi-scalar multiplication were designed.

Solinas introduced the Joint Sparse Form (JSF) presentation technique for pair of integers, which is a special kind of joint signed binary presentation, and proposed a new multi-scalar multiplication algorithm for a pair of integers using JSF. An open problem, (the joint sparse form for) Three or More Terms, was also put forward in his work. This paper gives a solution to it by proposing a new encoding algorithm, which transforms multiple integers into a new kind of signed binary representation of them.

On this basis, a new elliptic curve multi-scalar multiplication algorithm is proposed. Its analysis and comparisons to traditional methods are given.