第15卷 第8期 2015年3月
1671—1815(2015)08-0095-04

科 学 技 术 与 工 程
Science Technology and Engineering

Vol. 15 No. 8 Mar. 2015

计算机技术

一种改进的 2^w-ary 快速标量乘算法

张 洁
(兰州交通大学电子与信息工程学院,兰州 730070)

摘 要 标量乘是椭圆曲线密码体制(ECC)的基本运算,也是最耗时和极易受到攻击的运算之一。针对利用重编码和R-L技术实现2^w-ary快速标量乘算法的不足,对该算法进行了改进。在保持抗SPA的前提下,改进算法使用绝对值,避免了负数做数组下标的使用,减少了50%的存储量,节约了赋值后的消除操作。理论分析表明,改进算法优于原算法,算法的运算量降低2^w次。数字验证表明,改进算法比原算法快约17%。

关键词 椭圆曲线密码 倍乘 点加 侧信道攻击
中图法分类号 TP18; **文献标志码** A

1985年,Miller和Koblitz各自独立地将椭圆曲线应用于密码系统,由于椭圆曲线密码系统(elliptic curve cryptography, ECC)[1]与其他的公钥密码系统相比,具有安全性高、计算量小、处理速度快、存储空间占用小、带宽要求低等特点,因此受到越来越多的关注。在ECC的实现中,最基本、最耗时的运算是标量乘(即已知域内整数n和椭圆曲线上一点P,求nP的运算)。计算标量乘的算法主要有二进制法(展开、窗口、带符号)、k进制法、Frobenius自同态法以及利用复乘的GLV方法等。而经典标量乘算法易遭受侧信道攻击(side channel attack, SCA)。SCA是1996年P. Kocher提出的一种利用加密过程中的计算时间或能量消耗来分析秘密消息的攻击方法,主要分为两类,即简单能量分析(simple power analysis, SPA)[2]和差分能量分析(differential power analysis, DPA)[3]。由于SPA攻击易于实施,危害性大,因此仅考虑SPA攻击。

由于不同的操作对能量的消耗不同,SPA攻击者就是通过分析设备上一次密码操作所消耗的能量差异获得有用的密码信息。为了抵抗SPA攻击,倍加[4,5]、Montgomery[6]、指数重编码[6,7]方法被提出,然而倍加算法容易遭受安全错误攻击[8],Montgomery算法因为指数操作需要更多的计算成本,指数重编码算法因为指数循环中的数字0和非0也有指数算法的脆弱性[9]。由于2^w-ary标量乘法运算效率较高,而且在当前标量乘的实现中大多采用此算法,为此文献[8,9]研究了2^w-ary标量乘法抵抗SPA的攻击技术。文献[8]把变量转换成$\{-2^w,1,2,\cdots,2^{w-1}\}$,该方法由于使用从左至右的标量乘法算法,在主操作执行前,必须预计算重编码的标量[9]。文献[9]使用从右至左的标量乘法算法,没有预计算的成本,具有抗SPA能力;但是文献[9]使用负数做数组的下标,在实际使用中需要转换,另外无穷远点的计算仍然存在,算法的效率不高。由于无穷远点的分布是随机的,因此不存在攻击威胁。为此,对文献[9]改进,贡献有三个方面:避免了负数做数组下标时的再次转换操作;存储量减少了50%;在保持抗SPA的前提下,使标量加法的运算量减少2^{w-1}次。

2014年11月2日收到
作者简介:张 洁,女,硕士,讲师。研究方向:软件工程。E-mail:6529670@qq.com。

1 ECC预备知识

椭圆曲线是指由Weierstrss方程所确定的平面曲线,如式(1)。

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad (1)$$

根据椭圆曲线域的特征和J不变量的不同,可以将式(1)表示成五种不同的形式。尽管如此,五种椭圆曲线的点加和倍点计算的方式却是相同的,即椭圆曲线上位于同一直线的3个点之和被映射到无穷远点(垂直于x轴的直线第3个点为无穷远点),为此,简便起见,下面仅以特征大于3的曲线为例说明点加和倍点计算的方法。

特征大于3时,曲线的形式如式(2)。

$$y^2 = x^3 + ax + b \quad (2)$$

椭圆曲线上的点是一个由式(2)联系的二维向量,即(x,y)。假设用O表示无穷远点,P、Q、R表示同一直线上3个不同的点,其坐标分别是(x_1,y_1)、(x_2,y_2)、(x_3,y_3),则$P+Q+(-R)=O$。下面给

出基于式(2)的点加和倍点的计算公式。

已知 P、Q,计算 $P+Q=R$ 点加的公式如式(3)。

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{cases}, \lambda = \frac{y_2 - y_1}{x_2 - x_1} (x_2 \neq x_1) \tag{3}$$

已知 P,计算 $2P$ 点的公式如式(4)。

$$\begin{cases} x_3 = \lambda^2 - 2x_1 \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{cases}, \lambda = \frac{3x_1^2 + a}{2y_1} (y_1 \neq 0) \tag{4}$$

2 算法的改进

假设 k 表示密钥,$l = \left\lfloor \frac{\lg_2 k}{w} \right\rfloor$ 表示 2^w-ary 算法的长度,则 2^w-ary 算法的表示如式(5)。

$$k = \sum_{i=0}^{l} k_i 2^{iw} = \sum_{i=0}^{l} k_i' 2^{iw} \tag{5}$$

式(5)中 $k_i \in \{-2^w, 1, 2, \cdots, 2^{w-1}\}$;$(i = 0, \cdots, 2^{w-1})$ 为重编码,$k_i' \in \{0, 1, 2, \cdots, 2^{w-1}\}$;$(i = 0, \cdots, 2^{w-1})$ 为原编码,则原编码与重编码的转换如式(6)。

$$\begin{cases} k_i = k_i' + c_i, (c_0 = 0) \\ c_{i+1} = \begin{cases} 1, & k_i = 0 \text{ or } 2^{w-1} < k_i < 2^w \\ 0, & \text{其他} \end{cases} \end{cases} \tag{6}$$

c_{i+1} 的值利用文献[9]的方法计算求得,如式(7)。

$$c \leftarrow 1 - [(t + 2^w - 1 \gg w] + [(t + 2^{w-1} - 1 \gg w] - (t \gg w) \tag{7}$$

为了后文比较分析,这里给出文献[9]算法,见算法1。

算法1 规则 2^w-ary R-L 椭圆曲线标量乘

输入:$P \in E$, $k = \sum_{i=0}^{l} k_i' 2^{iw}$, $k_i' \in \{0, 1, 2, \cdots, 2^{w-1}\}$, $0 < k_l < 2^{w-1} + 1/w$

输出:$kP \in E$

(1) $B[i] \leftarrow P$ for $i = 0, \pm 1, \pm 2, \cdots, 2^{w-1}, \pm 2^w$

(2) $c \leftarrow 0$

(3) for $i = 0$ to l step 1

(a) $t \leftarrow k_i + c$

(b) $c \leftarrow 1 - ((t + 2^w - 1 \gg w) + ((t + 2^{w-1} - 1 \gg w) - (t \gg w)$

(c) $k_i \leftarrow 1 - (c \gg w)$

(d) $B[k_i] \leftarrow B[k_i] + B[0]$

(e) $B[0] \leftarrow 2^w B[0]$

(4) $B[i] \leftarrow B[i] - B[-i]$ for $i = 1, \cdots, 2^{w-1} - 1, 2^w$

(5) $B[2^{w-1}] \leftarrow B[2^{w-1}] - P$

(6) $T \leftarrow B[2^{w-1}] + 2B[2^w], S \leftarrow T$

(7) for $i = 2^{w-1} - 1$ to 1 step -1

(a) $T \leftarrow T + B[i]$

(b) $S \leftarrow S + T$

(8) return S

在算法1中,算法的第3步计算的下标有正有负,因为当 $c = 1$ 时,$k[i] = t - (c \ll w) < 0$,数组下标出现了负数。算法的第5步和第6步结果仍然会成为无穷远点,所以程序并没有消除无穷远点带来的隐患。由于无穷远点的应用是随机的,能量攻击不能分析出无穷远点的分布,因此,先赋值后消除的操作是没有必要的,另外,根据 k_i 值的对称性,只需算法1的一半就可以达到目标。对算法1的改进,其思路是使用绝对值避免了负数做数组下标的使用,也使存储量减少了50%,依据无穷远点分布的随机性,省略了赋值后的消除操作。算法的描述见算法2。

算法2 改进规则 2^w-ary R-L 椭圆曲线标量乘

输入:$P \in E$, $k = \sum_{i=0}^{l-1} k_i' 2^{iw}$,

$k_i' \in \{0, 1, 2, \cdots, 2^{w-1}, \pm 2^w\}$

输出:$kP \in E$

(1) $B[i] \leftarrow O$ for $i = 1, 2, \cdots, 2^{w-1}, 2^w$

(2) $B[0] = P, c \leftarrow 0$

(3) for $i = 0$ to l step 1

(a) $t \leftarrow k_i + c$

(b) $c \leftarrow 1 - ((t + 2^w - 1 \gg w) + ((t + 2^{w-1} - 1 \gg w) - (t \gg w)$

(c) $k_i \leftarrow 1 - (c \gg w)$

(d) if $|k_i| < 2^{w-1}$ or $k_i = 2^w$ then $B[k_i] \leftarrow B[k_i] + B[0]$ else $B[|k_i|] \leftarrow B[|k_i|] - B[0]$

(e) $B[0] \leftarrow 2^w B[0]$ 2.

(4) $T \leftarrow B[2^{w-1}] + 2B[2^w], S \leftarrow T$

(5) for $i = 2^{w-1} - 1$ to 1 step -1

(a) $T \leftarrow T + B[i]$

(b) $S \leftarrow S + T$

(6) return S

3 算法的比较分析

3.1 算法的存储量和运算量比较

算法1需要 $2^w + 2$ 个点的存储,算法2需 $2^{w-1} + 1$ 个点的存储,因此算法2比算法1节约了50%的存储量。

通过求解算法的运算量,来对比算法 1 和算法 2 的优劣。为了便于对比,将算法 1 和算法 2 运算量比较的结果列在表 1 中,如表 1 所示。

表 1　算法 1 和算法 2 运算量比较

Table 1　Computation comparison of algorithm 1 and 2

算法	程序序号	倍点次数	点加次数
算法 1	3	$w(l+1)$	$l+1$
	4	0	2^{w-1}
	6	1	1
	7	0	2^w-2
算法 2	3	$w(l+1)$	$l+1$
	5	0	2^w-2

从式(3)和式(4)对比发现,计算倍点比计算点加要快。简单起见,把算法 1 和算法 2 的倍点操作全部看成点加操作,则算法 1 总的运算量为 $T_1=(w+1)(l+1)+2^w+2^{w-1}+1$,算法 2 总的运算量为 $T_2=(w+1)(l+1)+2^w-1$。实际窗口尺寸一般大于 3,因为 $T_1-T_2=2^{w-1}-2>0$,所以算法 1 的运算量大于算法 2 的运算量,亦即算法 2 优于算法 1。

3.2　数字验证

假设窗口尺寸为 $w=3$,现在要计算 kP,其中 $k=321=1+5\times64$。也假设 k 的原始子项系数已计算出并存储在数组 $k[i](i=0,1,\cdots,2^w-1)$ 中,即 $k_0'=1,k_1^1=0,k_2'=5,k_3'=0$。下面分析算法 1 和算法 2 的执行过程。

在算法 1 中,计算得出的窗口值分别是 $k_0=1$,$k_1=-8,k_2=-2$。算法的主要执行过程如下:

程序的第 3 步执行结果为 $B[1]=2P,B[-8]=9P,B[-2]=65P,B[1]=514P$;程序的第 4 步执行结果为 $B[1]=513P$, $B[2]=-64P,B[3]=O$, $B[8]=-8P$;程序的第 5 步执行结果为 $B[4]=O$;程序的第 6 步执行结果为 $T=O+(-16P),S=-16P$;程序的第 7 步执行过程为,$T=T+B[3]=-16,S=S+T=-32P,T=T+B[2]=-80P,S=-112P,T=T+B[1]=433P,S=321P$,与所求解的目标值一致。另外,假设一个点占 1 个存储,则算法 1 所需的存储量为 9。

算法 2 的主要执行过程为:程序的第 3 步执行结果为 $B[1]=P,B[8]=-8P,B[2]=-64P,B[1]=513P$;程序的第 4 步执行结果为 $S=T=-16P$;程序的第 5 步执行过程为,$T=T+B[3]=-16P,S=S+T=-32P,T=T+B[2]=-80P,S=S+T=-112P,T=T+B[1]=433P,S=S+T=321P$。与所求解的目标值一致,因此算法 2 也是正确的。算法 2 需要的存储量为 4,约为算法 1 的 50%。下面再来看乘数 321 在两种算法中的运算量,见表 2。表 2 表明,算法 2 比算法 1 快 17%。

表 2　乘数为 321 的计算过程比较

Table 2　Calculation process comparison of multiplier as 321

算法	程序序号	倍点次数	点加次数	总运算量
算法 1	3	12	4	29
	4	0	4	
	5	0	1	
	6	1	1	
	7	0	6	
算法 2	3	12	4	24
	4	1	1	
	5	0	6	

4　结束语

使用绝对值避免了负数做数组下标的使用,略去了初值的消除操作,计算量降低。理论分析表明改进算法优于文献[9]算法,算法的运算量比文献[9]算法降低 2^{w-1} 次,算法的存储量比文献[9]减少 50%。数字验证表明,改进算法比算法比文献[9]算法快约 17%。

参　考　文　献

1　Koblitz N. Elliptic curve cryptosystems. Mathematics of Computation, 1987;48(177):203—209

2　Kocher P C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. Lecture Notes in Computer Science, 1996;1109:104—113

3　Kocher P C, Jaffe J, Jun B. Differential power analysis. Lecture Notes in Computer Science, 1999;1666:388—397

4　Coron J. Resistance against differential power analysis for elliptic curve cryptosystems. Lecture Notes in Computer Science, 1999; 1717:292—302

5　Boscher A, Naciri R, Prouff E. CRT RSA algorithm protected against fault attacks. Lecture Notes in Computer Science, 2007;4462:229—243

6　Joye M, Tunstall M. Exponent recoding and regular exponentiation algorithms. Lecture Notes in Computer Science, 2009; 5580: 334—349

7　Vuillaume C, Okeya K. Flexible exponentiation with resistance to side channel attacks. Lecture Notes in Computer Science, 2006; 3989: 268—283

8　Möller B. Securing elliptic curve point multiplication against side channel attacks. Lecture Notes in Computer Science, 2001;2200: 324—334

9　Baek Y J. Scalar recoding and regular 2^w-ary right-to-left EC scalar multiplication algorithm. Information Processing Letters, 2013;113 : 357—360.

An Improved 2^w-ary Fast Scalar Multiplication Algorithm

ZHANG Jie
(College of Electronic and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730070, P. R. China)

[**Abstract**] The scalar multiplication is the basic operations of the elliptic curve cryptography (ECC), and is also the most time consuming and vulnerable operation. According to the flaw on using recoding and R-L for fast computing scalar multiplication algorithm, the orignal algorithm is improved. Under the premise of the resisting SPA, the improved algorithm has some advantages, such as to avoid the negative number to use in an array subscript, to reduce the amount of 50% storage, to delete some operations about elimination after assignment. The theoretical analysis indicates that the operation of the improved algorithm is less 2^w than original algorithm. The digital verification shows that the operation complexity of improved algorithm reduce to 17% of original algorithm.

[**Key words**] elliptic curve cryptosystem scalar multiplication scalar addition side channel attack

(上接第 89 页)

13 王 凌,刘 波. 微粒群优化与调度算法. 北京:清华大学出版社,2008:16—19
Wang Ling, Liu Bo. Particle swarm optimization and scheduling Igorithms. Beijing: Tsinghua University Press, 2008: 16—19

14 胡海波,黄友锐. 混合粒子群算法优化分数阶 PID 控制参数研究. 计算机应用,2009;29(9)9:2483—2486
Hu Haibo, Huang Yourui. Research of fractional order PID controller using hybrid particle swarm optimization. Journal of Computer Applications, 2009; (29)9: 2483—2486

15 史 峰,王 辉,胡 斐,等. MATLAB 智能算法 30 个案例分析. 北京:北京航空航天大学出版社,2013:138—139
Shi Feng, Wang Hui, Hu Fei. Intelligent algorithm MATLAB 30 case analysis. Beijing: Beijing University of Aeronautics and Astronautics Press, 2013: 138—139

Fractional Order PI^{λ} Controller of Double Closed Loop DC Motor Speed Control

ZHANG Hai-ming[1], MIAO Zhong-cui[2*], ZHAO Jing-qiong[2]
(Lanzhou Jiaotong University School of Automation & Electrical Engineering[1], School of Mechanical and Electrical Engineering[2], Lanzhou 730070, P. R. China)

[**Abstract**] Fractional order $PI^{\lambda}D^{\mu}$ is a relatively new and high control performance controller. The Fractional order PI^{λ} was applied to the double closed-loop DC speed regulation system of speed controller, using particles swarm optimization algorithm to determine the parameters of the controller. The stability, follow performance and disturbance resistance of the fractional order PI^{λ} is superior to integer order PI which was demonstrated by Matlab/Simulink. The effectiveness of particle swarm optimization algorithm parameters setting was verified either.

[**Key words**] frational order PI^{λ} particle swarm double closed-loop speed control following performance