

基于 FPGA 的 ECC 快速算法研究及设计

陈俊杰, 孟李林, 袁 阳

(西安邮电大学 陕西省通信专用集成电路设计工程技术研究中心, 陕西 西安 710061)

摘 要: 椭圆曲线算法(ECC)的核心是点乘算法(KP), KP 性能决定了 ECC 的性能. 针对素数域点乘运算速度慢的问题, 提出了一种基于改进 NAF 的点乘并行调度算法. 在深入分析 Jacobian 射影坐标系下点加算法和倍点算法的基础上, 分别设计了点加并行运算算法和倍点并行运算算法. 基于 Cyclone IV 系列的 FPGA 开发平台实现了改进后 ECC 算法的硬件设计. 硬件测试结果表明: 完成一次点乘运算需要 111 860 个时钟周期. 与改进前算法相比, 运算速度提高了 40.3%. 如将改进后的点乘算法基于 ASIC 实现, 预估点乘算法性能可达到 72 393.6 次/s.

关键词: 椭圆曲线; 点乘算法; 素数有限域; 并行调度; FPGA

中图分类号: TP402

文献标识码: A

文章编号: 1000—7180(2016)08—0139—05

The Research and Design of ECC Fast Algorithm Based on FPGA

CHEN Jun-jie, MENG Li-lin, YUAN Yang

(Shaanxi Provincial Communications ASIC Design Engineering Technology Research Center,
Xi'an University of Posts and Telecommunications, Xi'an 710061, China)

Abstract: The core of elliptic curve cryptographic algorithm (ECC) is the point multiplication algorithm (KP), the performance of the KP determines the performance of the ECC. A new KP parallel scheduling algorithms based on improved NAF is presented in this paper for solving the problem of low speed of KP in prime finite field. By deeply analysing the point addition algorithm and the point doubling algorithm of the Jacobian projective coordinates, designed the point-addition parallel computing algorithm and the point-doubling parallel computing algorithm. The hardware design of the improved algorithm of ECC has completed based on the FPGA development platform of Cyclone IV family. The testing result of hardware shows that a point multiplication operation needs 111,860 clock cycles. Comparing with the previous algorithms, the computing speed improve 40.3%. If the improved algorithm is implemented based on ASIC, estimating the performance of point multiplication algorithm can reach 72,393.6 times/s.

Key words: elliptic curve; point multiplication algorithm; prime finite field; parallel scheduling; FPGA

1 引言

椭圆曲线密码算法(Elliptic Curve Cryptography, ECC)最早于 1985 年由 Koblitz^[1] 和 Miller^[2] 提出, 其安全性依赖于椭圆曲线离散对数问题的难解性^[3]. 与 RSA 公钥密码体制相比, ECC 具有密钥更短、安全性更高、运算速度更快等优势, 使其广泛应用在信息安全领域. ECC 的核心是点乘算法, 因此点乘算法的硬件设计及实现是目前 ECC 算法研

究的重点.

本文在分析已有的点乘算法硬件设计基础上, 针对素数域($GF(p)$)上点乘调度算法、点加算法和倍点算法进行分析研究, 提出算法改进及并行运算的设计实现方案.

2 椭圆曲线密码算法

2.1 素域上椭圆曲线

基于 $GF(p)$ ($p > 3$) 的椭圆曲线是一个 Weier-

strass 方程: $y^2 = x^3 + ax + b$, 其中 p 为素数, $x, y, a, b \in \text{GF}(p)$, 且使得该方程定义的点集和无穷远点 O 在点的加法下构成阿贝尔群, 在此阿贝尔群上即可构建椭圆曲线密码算法^[4]. ECC 的核心是点乘运算, 点乘运算可分解为点加运算和倍点运算. 素域上的椭圆曲线常用的表示形式有三种: 仿射坐标表示、标准射影坐标表示和 Jacobian 射影坐标表示. 不同坐标系下椭圆曲线点加运算和倍点运算的复杂度, 如表 1 所示^[5]. 表中 I、M 和 S 分别表示有限域中的求逆运算、乘法运算和平方运算.

表 1 素域上椭圆曲线运算复杂度

	仿射坐标	标准射影坐标	Jacobian 射影坐标
点加	I+2M+S	13M+S	12M+4S
倍点	I+2M+2S	8M+5S	4M+6S

2.2 椭圆曲线点乘

椭圆曲线上的一点 P 与整数 k 相乘被称为标量乘或点乘, 它被定义为 k 个 P 的相加, 即 $kP = P + P + \dots + P + P$. 点乘由点加和倍点运算构成, 而点加运算和倍点运算可分解为模乘、模加/减和模逆运算, 如图 1 所示. 在有限域内模逆运算耗时是模乘运算的 70 倍, 则由表 1 可知: 素域上 Jacobian 射影坐标的点乘算法运算时钟周期最少. 采用 Jacobian 射影坐标点乘运算, 需要将仿射坐标转换为 Jacobian 射影坐标. 仿射坐标 (x, y) 转换到 Jacobian 射影坐标 (X, Y, Z) 的方式^[3]为 $(x, y) \rightarrow (X = x, Y = y, Z = 1)$; Jacobian 射影坐标 (X, Y, Z) 转换到仿射坐标 (x, y) 的方式为 $(X, Y, Z) \rightarrow (x = X/Z^2, y = Y/Z^3)$.

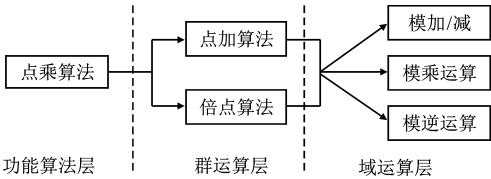


图 1 点乘算法的层次结构

3 点乘算法分析及改进

3.1 点乘算法分析

常用的点乘算法包括以下几种: 加减点乘算法、固定窗口加减点乘算法、w_ary 点乘算法、符号位 w_ary 点乘算法、蒙哥马利点乘算法. 这五种点乘算法中, 蒙哥马利点乘算法计算复杂度小, 不需要预计算, 不需要预存储, 运算效率最高^[3].

在点乘运算之前, 使用 NAF(Non-Adjacent Form) 对 k 值进行重编码可以提高点乘运算速度.

$\text{NAF}(k) = (k_{t-1}, k_{t-2}, \dots, k_1, k_0)_{\text{NAF}}$, 其中 t 为 k 二进制表示时的位数, $k_i \in \{0, \pm 1\}$ 且 $k_{t-1} \neq 0$. 重编码前, 点乘运算平均需要 $t-1$ 次倍点运算和 $t/2$ 次点加运算. 而使用 NAF 对 k 重编码后, 点加运算的平均次数减少至 $t/3$ ^[7]. 所以本文采用 NAF 编码思想进行椭圆曲线点乘的计算^[8-9], 其算法描述如算法 1 和算法 2 所示.

算法 1 基于 NAF 的点乘调度算法

输入: $k, P \in \text{GF}(p)$
输出: $Q = kP$
(1) 计算 $\text{NAF}(k) = \sum_{i=0}^{t-1} k_i 2^i$
(2) $Q \leftarrow \infty$
(3) i 从 0 到 $t-1$, 重复执行:
 (3.1) $Q \leftarrow 2Q$
 (3.2) if($k_i = 1$) $Q \leftarrow Q + P$
 (3.3) if($k_i = -1$) $Q \leftarrow Q - P$
(4): 返回 Q

算法 2 NAF 编码

输入: 正整数 k
输出: $\text{NAF}(k)$
(1) $i \leftarrow 0$
(2) if($k \geq 1$) 重复如下步骤:
 (2.1) if($k[0] = 1$) $\{k_i \leftarrow 2 - (k \bmod 4), k \leftarrow k - k_i\}$
 (2.2) else $k_i \leftarrow 0$
 (2.3) $k \leftarrow k/2, i \leftarrow i + 1$
(3) 返回 $\text{NAF}(k) = (k_{t-1}, \dots, k_0)_{\text{NAF}}$

3.2 点乘算法改进

算法 1 采用先倍点后点加的串行运算方式, 运算速度慢. 深入分析算法 1 的运算流程可以得出: Jacobian 射影坐标下倍点运算和点加运算虽然在数据上相互影响, 但这种相互影响是有规律的. 倍点运算输入数据是上一轮倍点运算结果, 点加运算输入数据是上一轮点加运算结果和本轮倍点运算结果. 因此, 本轮点加运算和下一轮倍点运算同时运算不影响最终计算结果的正确性.

基于以上分析, 可以将算法 1 中步骤(3)的串行运算方式, 改进为并行运算方式, 改进后点乘算法运算时钟周期数可减少 33%. 具体改进如算法 3 所示.

算法 3 基于 NAF 的点乘调度算法

输入: $k, P \in \text{GF}(p)$
输出: $Q = kP$
(1) 计算 $\text{NAF}(k) = \sum_{i=0}^{t-1} k_i 2^i$
(2) $Q_0 \leftarrow P$
 if($k_0 = -1$) $Q_A \leftarrow -P$

else $Q_A \leftarrow P$
 (3) $Q_D \leftarrow 2Q_D$
 (4) i 从 1 到 $t-2$, 重复执行:
 if($k_i = 1$) { $Q_D \leftarrow 2Q_D$, $Q_A \leftarrow Q_A + Q_D$ }
 else if($k_i = -1$) { $Q_D \leftarrow 2Q_D$, $Q_A \leftarrow Q_A - Q_D$ }
 else $Q_D \leftarrow 2Q_D$
 (5) $Q_A \leftarrow Q_A + Q_D$, 返回 Q_A

算法 2 中存在不足. 由于执行步骤(2.1)时, k_0 的值为 1, 因此, 无论 k_1 取什么值, k_1 和 k_0 的计算结果都是 0. 由于 k_1 和 k_0 都为 0, 所以步骤(2.3) 必将连续执行两次, 即左移两位. 本设计对步骤(2.1) 的计算结果直接左移两位, 使两次左移运算在一个步骤内完成, 从而提高编码效率. 具体改进如算法 4 所示.

算法 4 改进的 NAF 编码

输入: 正整数 k

输出: NAF(k)

(1) $i \leftarrow 0$
 (2) if($k \geq 1$) 重复如下步骤:
 (2.1) if($k[0] = 1$)
 { $k_i \leftarrow 2 - (k \bmod 4)$, $k \leftarrow k - k_i$
 $k_{i+1} \leftarrow 0$, $k \leftarrow k/4$
 $i \leftarrow i + 1$ }
 (2.2) else
 { $k_i \leftarrow 0$, $k \leftarrow k/2$
 $i \leftarrow i + 1$ }
 (3) 返回 NAF(k) = $(k_{i-1}, \dots, k_0)_{\text{NAF}}$

4 Jacobian 射影坐标倍点和点加算法优化设计

4.1 倍点并行运算算法优化设计

倍点运算定义如下: 设点 $P_1 = (x_1, y_1, z_1)$, $P_2 = (x_2, y_2, z_2)$, $P_1 = P_2$, $P_3 = P_1 + P_2 = (x_3, y_3, z_3) \neq 0$, 则:

$$\lambda_1 = 3x_1^2 + az_1^4, \lambda_2 = 4x_1y_1^2, \lambda_3 = 8y_1^4,$$

$$x_3 = \lambda_1^2 - 2\lambda_2, y_3 = \lambda_1(\lambda_2 - x_3) - \lambda_3, z_3 = 2y_1z_1;$$

由上述定义可知, 倍点运算由一系列的模乘、模加/减运算构成, 共包括 10 次模乘和 13 次模加/减. 将这些基本运算按照数据的依赖关系, 在时间和空间两个维度合理地排列组合, 可以得到倍点运算过程中没有数据依赖关系的变量, 如图 2 所示. 图 2 中位于同一根节点下的各个分支节点为无数据依赖关系的变量. 由于没有数据依赖关系的变量同时运算, 不影响最终计算结果的正确性, 因此倍点运算过程中存在着模乘和模加/减同时运算的可能.

首先从时间维度分析, 由于模加/减运算较模乘

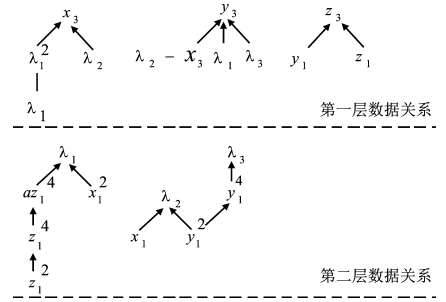


图 2 倍点数据的依赖关系

运算耗时短, 所以当模乘和模加/减数据没有依赖关系时, 并且在模乘运算将计算结果存入相应寄存器之前, 可同时进行模加/减运算.

其次从空间的维度分析, 由于倍点并行运算时, 同一时刻有多个 256 bit 数据位宽的变量参与不同的运算, 因此这些变量不适合采用 RAM 存储, 应该采用专用的寄存器组存储. 结合图 2 和倍点运算定义, 可以得出倍点并行运算需设计 7 个复用寄存器: $x_3, y_3, z_3, \lambda_1, \lambda_2, T_1, T_2$, 从而完成倍点并行运算算法设计, 如表 2 所示.

表 2 中的倍点并行运算算法只需 10 次模乘运算和 1 次模减运算的时间, 共节省了 12 次模加/减运算的时间, 即倍点运算时钟周期数减少了 9.4%.

表 2 倍点并行运算算法

运算 步骤	模乘	模加/减
1	$T_2 = x_1^2$	
2	$T_2 = z_1^2$	$T_1 = 3T_2$
3	$T_2 = z_1^4$	
4	$T_2 = az_1^4$	
5	$T_2 = y_1^2$	$\lambda_1 = T_1 + T_2$
6	$T_1 = x_1y_1^2$	
7	$T_1 = \lambda_1^2$	$\lambda_2 = 4T_1$
8	$T_1 = y_1^4$	$T_2 = 2\lambda_2 \rightarrow x_3 = T_1 - T_2$
9	$T_1 = y_1z_1$	$T_2 = \lambda_2 - x_3 \rightarrow x_3 = 8T_1$
10	$T_1 = \lambda_1(\lambda_2 - x_3)$	$z_3 = 2T_1$
11		$y_3 = T_1 - x_3$

4.2 点加并行运算算法优化设计

点加运算定义如下: 设点 $P_1 = (x_1, y_1, z_1)$, $P_2 = (x_2, y_2, z_2)$, $P_1 \neq P_2$, $P_3 = P_1 + P_2 = (x_3, y_3, z_3) \neq 0$, 则:

$$\lambda_1 = x_1z_2^2, \lambda_2 = x_2z_1^2, \lambda_3 = \lambda_1 - \lambda_2, \lambda_7 = \lambda_1 + \lambda_2, \\ \lambda_4 = y_1z_2^3, \lambda_5 = y_1z_1^3, \lambda_6 = \lambda_4 - \lambda_5, \lambda_8 = \lambda_4 + \lambda_5, x_3 = \\ \lambda_6^2 - \lambda_7\lambda_3^2, \lambda_9 = \lambda_7\lambda_3^2 - 2x_3, z_3 = z_1z_2\lambda_3, y_3 = (\lambda_9\lambda_6 -$$

$\lambda_8\lambda_3^3)/2$.

由上述定义可知,点加运算由 17 次模乘和 8 次模加 / 减构成. 参照倍点并行运算算法设计原理,可以得到点加运算过程中没有数据依赖关系的变量,根据这些变量和点加算法定义,可以得出点加并行运算需设计 10 个复用寄存器: $x_1, y_1, z_1, x_3, y_3, z_3, T_1, T_2, T_3, T_4$,从而完成点加并行运算算法设计,如表 3 所示.

表 3 中的点加并行运算算法只需 17 次模乘运算的时间,共节省了 8 次模加 / 减运算的时间,即点加运算时钟周期数减少了 3.7%.

表 3 点加并行运算算法

运算 步骤	模乘	模加 / 减
1	$T_4 = z_1^2$	
2	$T_1 = x_1 z_2^2$	
3	$T_3 = z_1^2$	
4	$T_2 = x_2 z_1^2$	
5	$T_4 = z_2^3$	$x_1 = \lambda_1 - \lambda_2 \rightarrow x_3 = \lambda_1 + \lambda_2$
6	$T_1 = y_1 z_2^3$	
7	$T_3 = z_1^3$	
8	$T_2 = y_2 z_1^3$	
9	$T_3 = \lambda_3^2$	$y_1 = \lambda_4 - \lambda_5 \rightarrow y_3 = \lambda_4 + \lambda_5$
10	$T_2 = \lambda_7 \lambda_3^2$	
11	$T_1 = \lambda_6^2$	
12	$T_3 = \lambda_3^3$	$x_3 = T_1 - T_2$
13	$T_2 = \lambda_8 \lambda_3^3$	$T_4 = 2x_3 \rightarrow T_4 = T_2 - T_4$
14	$T_1 = \lambda_9 \lambda_6$	
15	$T_3 = z_1 z_2$	$y_3 = T_1 - T_2$
16	$z_3 = z_1 z_2 \lambda_3$	
17	$y_3 = \lambda_3 (1/2)$	

5 点乘算法的硬件设计

基于 Altera 公司 Cyclone IV EP4CGX75CF23C6 芯片的开放平台实现了改进后的点乘算法. 由于 FPGA 芯片资源有限,实现速度与面积的平衡是硬件设计的关键. 点乘运算的核心是模乘,根据分析和测试发现:选用该 FPGA 芯片中 128 bit 乘法器 IP 核设计 256 bit 模乘运算单元,能够较好地实现面积和速度的平衡.

5.1 点加和倍点优化算法的硬件设计

本文提出的点加、倍点并行运算算法,支持模乘和模加/减并行运算. 首先,256 bit 模乘运算单元基于 128 bit 乘法器 IP 核,采用 2 级流水线设计实现. 其次,针对点加和倍点中模加/减运算特点,分别设计了 Add_Sub_ALU_A 和 Add_Sub_ALU_D 两种运算器,这两种运算器能够处理所有的模加/减计算表达式.

其中 mux_a_s 信号用来控制运算类型,mod_num 信号用来控制运算次数. 模加/减运算的这种设计方式,比在域运算层分别设计各域运算单元更加节省硬件资源,同时大大降低群运算层控制电路的设计复杂度,其中控制电路只需控制 mux_a_s, mod_num 两个信号,不需要设计额外的控制状态机.

最后,为该算法各步骤的计算表达式设计专用的硬件调度结构,形成一个全局串行局部并行的硬件结构,如图 3 所示,从而提高点加和倍点的运算效率.

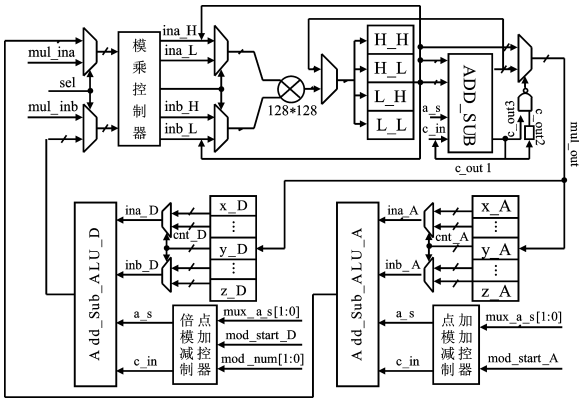


图 3 点加和倍点并行运算算法的硬件设计

5.2 快速点乘算法的硬件设计

点乘运算由点加和倍点运算构成,为了实现点加和倍点并行运算,针对点加和倍点模块分别设计两组应答信号 Start_DBL、Done_DBL 和 Start_ADD、Done_ADD,通过这两组应答信号和一个逻辑与门实现点加和倍点运算的并行调度,如图 6 所示. 这种设计方式比协处理器方式更加节省硬件资源,同时大大降低了控制电路的设计复杂度.

图 4 中其他电路模块分别是:Dtpre 为数据转换电路、NAF(k)为重编码电路、PtoA 为坐标转换电路、Inv 为快速模逆电路; $x_D, y_D, z_D, x_A, y_A, z_A$ 为这些电路模块的复用寄存器.

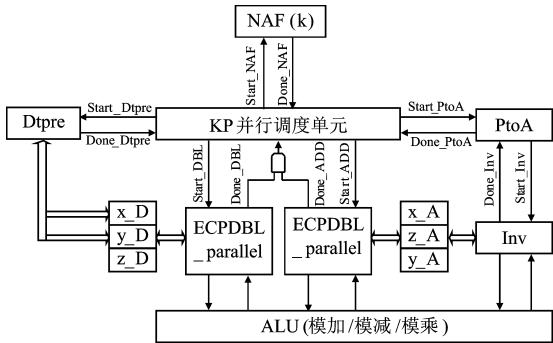


图 4 快速点乘算法的硬件设计

6 性能分析与比较

电路设计采用 verilog HDL 语言实现,完成一次 256 bit 点乘算法需要 111 860 个时钟周期,在 Cyclone IV EP4CGX75CF23C6 上完成电路测试.表 4 给出本文与其他文献的比较结果.

表 4 中,文献[4]采用高速 32 位协处理器设计,但由于点乘算过程中,每次迭代后都要进行大量的数据搬移,严重限制了点乘运算速度,导致点乘运算时钟周期数多.文献[10]基于 ASIC 实现,针对模乘运算设计专用的 3 级流水脉动算术阵列,但其采用串行方式处理,导致运算时钟周期数过多.由表中数

据可知,本文设计的点乘算法运算速度较快,同时较好的实现了面积和速度的平衡,在已查阅的文献中,完成一次点乘运算所需时钟周期数最少.

本文设计受选用 FPGA 芯片的限制,在时钟频率上比较慢,影响了点乘运算性能,但所设计的算法具有明显优势,运算时钟周期数最少.若选用高端 FPGA 器件实现,则点加运算和倍点运算可分别设计为四路并行运算结构,同时选用 256 bit 乘法器 IP 核,可以使点乘运算性能提高约 12 倍,即 9 049.2 次/s.若选用 ASIC 实现,当时钟频率达到文献[10]的水平时,点乘运算性能还能再提高约 8 倍,即 72 393.6 次/s.

表 4 点乘算法硬件实现比较

文献	时间	密钥长度/b	器件(工艺/ μm)	时钟频率/ MHz	点乘性能/ (次/s)	运算时钟 周期数(T)	域类型
[9]	2008	163	Virtex2 XC2V3000	70	436.6	160 329	二元域
[9]	2008	163	Virtex2 XC2V3000	70	236.4	296 108	素域
[4]	2010	256	CycloneII EP2C3	180	773.5	232 704	二元域
[10]	2011	256	ASIC(0.18 μm)	671	1176	570 578	素域
[11]	2013	233	StratixIII EP3SL1	276.24	617.2	447 569	二元域
本文	2015	256	Cyclone IV GX75C6	83.51	754.1	111 860	素域

7 结束语

本文针对传统点乘运算速度慢的缺点.首先,在传统点乘调度算法的基础上,提出一种点乘并行调度算法,支持点加运算和倍点运算同时执行,大大提高点乘的运算效率.其次,对 Jacobian 射影坐标下的点加算法和倍点算法进行改进,分别设计了一种点加并行运算算法和倍点并行运算算法,并根据算法各步骤的计算表达式设计专用的硬件调度结构,支持模乘和模加/减并行运算.最后,改进传统 NAF 编码算法,提高了编码效率.硬件测试结果表明,与已查阅的 FPGA 和 ASIC 实现相比,本文提出的点乘算法不仅运算时钟周期数最少,而且硬件实现的整体性能显著提高,能够应用于信息安全领域.

参考文献:

[1] Koblitz N. Elliptic curve cryptosystems[J]. Mathematics of Computation,1987,48(177):203-209.
[2] Miller V. Use of Elliptic Curves in Cryptography[C] // Advances in Cryptology Crypto’85, Lecture Notes in Computer Science. Heidelberg: Springer, 1986: 417-426.

[3] Xining Cui, Jingwei Yang. An FPGA based processor for elliptic curve cryptography[C]// Computer Science and Information Processing (CSIP). Xi’an: IEEE, 2012:343-349.
[4] 何德彪,陈建华. 高速椭圆曲线密码协处理器的设计与实现[J]. 华南理工大学学报(自然科学版),2010,38(5):90-94.
[5] 国家密码管理局. GM/T 0003-2012 SM2 椭圆曲线公钥密码算法[S]. 北京:国家密码管理局,2012.
[6] Yongnan Li, Limin Xiao, Zifeng Wang, et al. High performance point-multiplication for conic curves cryptosystem based on standard NAF algorithm and chinese remainder theorem [C] // Information Science and Applications (ICISA). Jeju Island: IEEE, 2011: 1-8.
[7] Nagai Y, Shirase M, Izu I. Elliptic curve scalar multiplication with a bijective transform [C] // Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS). Birmingham: IEEE, 2014:280-286.
[8] Al-Otaibi A, Al-Somani T F, Beckett P. Efficient elliptic curve parallel scalar multiplication methods[C] // Computer Engineering & Systems (ICCES). Cairo: IEEE, 2013:116-123.

表 3 检测结果对比表 %

样本类型	SVDD		PSO-SVDD	
	检测准确率	误报率	检测准确率	误报率
Normal	87. 24	0. 23	92. 30	0. 14
DoS	86. 56	0. 15	95. 14	0. 07
R2L	78. 09	0. 26	82. 47	0. 16
U2R	85. 74	0. 17	89. 53	0. 09
Probe	81. 27	0. 29	93. 61	0. 13

为比较计算量大小,表 4 给出了两种算法的时间对比情况,从结果可看出本文算法所需的训练和测试时间大大减少,这是由于采用 PSO 特征优选后,参与入侵检测的数据量大大减少,因此计算量较小.

表 4 检测速度对比 s

样本类型	SVDD		PSO-SVDD	
	训练	测试	训练	测试
Normal	51	23	25	11
DoS	52	26.	27	12
R2L	57	22	22	9
U2R	56	24	21	14
Probe	54	24	20	10

4 结束语

本文从入侵检测特征提取和分类模型建立两方面,提出了基于粒子群优化的支持向量描述算法的入侵检测方法,该方法降低了冗余数据对分类器性

能的影响. 通过特征提取降低特征数据的维数,降低了计算的复杂程度,节省了计算时间且提高了分类器的性能. 在以后的工作中,将进一步研究算法的适应性和实时性.

参考文献:

[1] 饶鲜,董春曦,杨绍全. 基于支持向量机的入侵检测系统[J]. 软件学报,2003,14(4):798-802.

[2] 刘宇,朱随江,刘宝旭. 采用改进 PTSVM 的入侵检测研究[J]. 计算机工程与应用,2012,48(5):1-3,74.

[3] Tax D M J, Duin R P W. Support vector data description[J]. Machine Learning,2004,54(1):45-66.

[4] Mahbod Tavallace, Ebrahim Bagheri, Wei Lu, et al. A Detailed Analysis of the KDD CUP 99 Data Set[C]// Proceeding of the Setonal IEEE International Conference on Computational Intelligence for Security and Defense Applications. [s. l.]: IEEE,2009:1-10.

[5] Dd_tools (Version 2. 0. 0 by July 2013[EB/OL]. [2015-10-27]. http://prlab.tudelft.nl/david-tax/dd_tools.html, 2013.

[6] 金欣磊,马龙华,吴铁军,等. 基于随机过程的 PSO 收敛性分析[J]. 自动化学报,2007,33(12):1263-1268.

[7] 梁海军. 人工蜂群优化支持向量机算法在网络安全中的应用[J]. 微电子学与计算机,2013,30(2):95-98.

作者简介:

魏振伟 男,(1981-),硕士,工程师. 研究方向为检测与诊断,模式识别. E-mail:ygszzia@126. com

刘 飞 男,(1981-),博士研究生. 研究方向为计算机系统安全.

(上接第 143 页)

[9] 王健. 椭圆曲线加密体质的双有限域算法及其硬件实现[D]. 北京:北京大学,2008.

[10] 陈传鹏,覃中平. 改进的素数域椭圆曲线密码处理器[J]. 武汉大学学报(工学版),2011,44(1):124-127.

[11] Urbano-Molano F A, Trujillo-Olaya V, Velasco-Medina J. Design of an Elliptic Curve Cryptoprocessor using Optimal Normal Basis over GF(2²³³)[C]//Circuits and Systems(LASCAS). Cusco: IEEE,2013:1-4.

作者简介:

陈俊杰 男,(1991-),硕士. 研究方向为计算机系统结构与 VLSI. E-mail:chenjj519@163. com.

孟李林 男,(1962-),教授. 研究方向为计算机系统结构,嵌入式系统.

袁 阳 男,(1990-),硕士. 研究方向为集成电路系统设计.