ECC 椭圆曲线密码体制 C* Core 实现与优化*

钱　丹[1,2]，李　飞[1]，高献伟[2]，董秀则[2]，曾　辉[1]
(1. 成都信息工程学院 计算机学院，成都 610225；2. 北京电子科技学院 电子工程系，北京 100070)

摘　要：对 C* Core 国芯芯片中实现 ECC 椭圆曲线密码加密算法进行了深入研究，概述了 C* Core 芯片中存储特点，给出 C* Core 芯片中椭圆曲线中数据点表示方法，结合 ECES 加密协议，在 C* Core 芯片中成功实现二元域 F_2^m 中 NISI 推荐的五条椭圆曲线加密算法；然后依次对初始程序进行三种方式优化，重点阐述了改进 Montgomery 点乘算法，详细记录每次优化前后程序耗时；最后对比各阶段程序运行耗时，得出优化率。C* Core 芯片中 ECC 加密算法运行效率优化后总体平均提高近 90%。

关键词：国芯；椭圆曲线密码；加密算法；二元域；点乘；优化
中图分类号：TP368.1　　**文献标志码：**A　　**文章编号：**1001-3695(2012)06-2243-03
doi:10.3969/j.issn.1001-3695.2012.06.064

Implementation and optimization of ECC based on C* Core

QIAN Dan[1,2], LI Fei[1], GAO Xian-wei[2], DONG Xiu-ze[2], ZENG Hui[1]
(1. *School of Computer, Chengdu University of Information Technology, Chengdu* 610225, *China*; 2. *Dept. of Electronic Engineering, Beijing Electronic Science & Technology Institute, Beijing* 100070, *China*)

Abstract: This paper surveyed the in-depth analysis on the implement of elliptic curve cryptography encryption algorithm in C* Core chip. Firstly, it proposed the storage characteristics of C* Core chip, gave a method to express the data points in elliptic curve. Combined with the ECES encryption protocols, it successfully achieved ECC encryption algorithm under five elliptic curve recommended by NISI in F_2^m domain on C* Core chip; followed by three ways of optimizing the initial program, which focused on improving the Montgomery point multiplication algorithm, it recorded each time-consuming procedures before and after optimization; through optimization, ECC encryption algorithm in C* Core chip, the overall average increase in the efficiency of nearly 90%.

Key words: C* Core; elliptic curve cryptography(ECC); encryption algorithm; binary field; point multiplication; optimization

随着 C* Core 系列嵌入式芯片的诞生，基于此类芯片的安全应用与研究广受关注[1]。目前国内对 C* Core 芯片实现密码体制算法相关性能缺乏验证。针对此问题，本文借助 C* Core 芯片平台实现且优化椭圆曲线加密(ECC)算法，通过详细数据记录，获取 C* Core 芯片 ECC 程序执行相关具体性能。

1　C* Core 芯片中 ECC 算法实现

1.1　C* Core 数据存储特点

具体以 CCM3118 芯片为例[2]，其片内含 2 KB ROM、64 KB SRAM，数据存储默认采用大端(big-endian)方式，如图 1 所示，即字的最重要字节在地址低位[3]。

31			0	
字节0	字节1	字节2	字节3	字地址：0x00000000
字节4	字节5	字节6	字节7	字地址：0x00000004
字节8	字节9	字节A	字节B	字地址：0x00000008
字节C	字节D	字节E	字节F	字地址：0x0000000C

图1　内存数据组织

图 2 为数值采用大端方式在内存中存储的具体对应示意图，字地址 0x0000_0C80 对应数值 0X12344321，字地址 0X0000_0C84 对应数值 0X56788765。

字地址：0X000 0_0C80	00010010	00110100	01000011	00100001
字地址：0X000 0_0C84	10000111	01100101	01010110	01111000

图2　数据内存存储

C* Core 数据存储格式决定了 ECC 数据表示的特殊性。本文选用 NISI 推荐的五个 $GF(2^m)$ 上随机椭圆曲线作为实现对象，具体有以下五域：$GF(2^{163})$、$GF(2^{233})$、$GF(2^{283})$、$GF(2^{409})$、$GF(2^{571})$。具体椭圆曲线上长数据的表示以 $m=163$ 为例，假设有数据为 0x7fffffff 2013ffff 472f8a6f 22031dff 43cfefff，可定义一含 6 数据单元的无符号整型数组 $a[6]$ 表示，字长 32 位。具体对应关系如表 1 所示，其中 $a[0]$ 对应二元域多项式 162 ~ 160 位，其他域曲线上点表示原理类似。

1.2　椭圆曲线密码算法实现

本文 ECC 加/解密实现采用 ECES 加密协议[4]，工作过程如图 3 所示。其中：G 点为公钥，J 为基点，S 为私钥，K 为用户 E 产生的随机数。

收稿日期：2011-10-12；**修回日期：**2011-11-19　　**基金项目：**四川省科技基金资助项目(2008GG0007)

作者简介：钱丹(1986-)，男，安徽无为人，硕士研究生，主要研究方向为信息安全(yiandeboke@sina.com)；李飞(1966-)，男，湖南常德人，教授，硕导，主要研究方向为信息安全；高献伟(1970-)，男，湖南岳阳人，教授，硕导，主要研究方向为密码工程；董秀则(1979-)，男，山东莒县人，讲师，硕士，主要研究方向为密码工程；曾辉(1984-)，男，河南驻马店人，硕士研究生，主要研究方向为信息安全.

表 1　ECC 长数据表示

数组	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
数值(0x)		ffff	2013	472f	2203	43cf
	7	ffff	ffff	8a6f	1dff	efff
范围/位	162～160	159～128	127～96	95～64	62～32	31～0

图3　ECC加/解密过程示意图

系统构建后产生密钥步骤[5]：

a)在区间[1,$n-1$]内利用 C* Core 中真随机发生器随机选取一整数,利用 S 数组表示。

b)利用从右向左的二进制点乘方法计算 G 点(G_x,G_y),其中 $G=SJ$。

c)此 G 点即为图 3 中的公钥。

d)整数 S 即为本文用户 D 的私钥。

图 4 为程序实现实例截图,由明文数组 d_ming 与解密明文数组 ming 相等可知,程序运行正确。

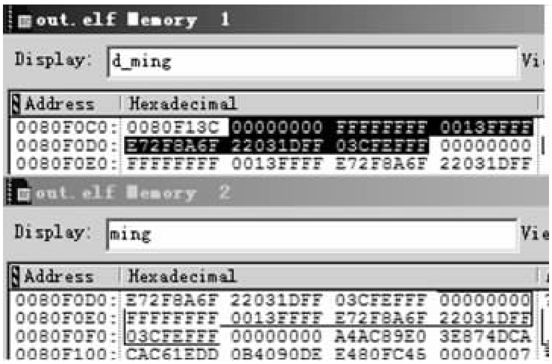

图4　ECC初始程序实现验证图

d_ming 首地址为 0X0080_F0C4,ming 数组首地址为 0X0080_F0DC,两数组内容相等,ECC 加解密过程实现。表 2 为初始程序分别在 $GF(2^m)$ 中五种域下加解密一次的耗时。

表 2　初始程序运行耗时

	$m=163$	$m=233$	$m=283$	$m=409$	$m=571$
时间/s	4.978	10.512	19.458	57.258	78.589

2　基于 C* Core 的 ECC 椭圆曲线优化

2.1　一次优化——基于 Montgomery 点乘算法

ECC 加密算法耗时集中在点乘计算中,利用 Montgomery 点乘算法相对于传统点乘算法可大幅度节省计算用时。

Montgomery 基本思想[6]:令 $Q_1=(x_1,y_1)$ 和 $Q_2=(x_2,y_2)$,$Q_1 != Q_2$,再令 $Q_1+Q_2=(x_3,y_3)$,$Q_1-Q_2=(x_4,y_4)$,则可得

$$x_3=x_4+\frac{x_2}{x_1+x_2}+\left(\frac{x_2}{x_1+x_2}\right)^2 \tag{1}$$

故 x_3 坐标可以由 Q_1、Q_2 和 x_4 坐标计算。在以上确定 KJ 算法中的 v 迭代仅计算 $T_j=[LJ,(L+1)J]$的 x 坐标,其中 L 是 K 的最左 v 位所表示的整数。若 K 的最左($v-1$)位为 0 或 1,则 $T_{v-1}=[2LJ,(2L+1)J,(2L+2)J]$,如图 5 所示。

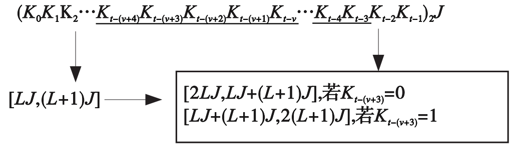

图5　Montgomery方法点乘的一次迭代

u 次迭代后,对于 $L=(k_0\cdots k_{t-(j+4)})_2$,$LJ$ 和($L+1$)J 的 x 坐标便知道了。$u+1$ 次迭代需要一次倍点和一次加运算,以便为 $L'=(k_0\cdots k_{t-(u+4)})_2$ 寻找 L、J 和($L'+1$)J 的 x 坐标。此算法运行的优点一是不需要额外存储器;二是在主循环的每一次迭代中完成相同操作,增强了抵抗时间分析攻击和能量分析攻击的能力[7]。表 3 为 Montgomery 点乘与从右向左法点乘用时对照表。

表 3　两种点乘算法耗时对照

	耗时/ms				
	$m=163$	$m=233$	$m=283$	$m=409$	$m=571$
从右向左法	878	2075	4 008	10 153	15 036
Montgomery	290	679	988	2 348	6 074

表 4 为采用 Montgomery 算法后 ECC 加密程序简称一次优化程序执行耗时表。

表 4　一次优化后程序执行耗时

	$m=163$	$m=233$	$m=283$	$m=409$	$m=571$
时间/s	1.165	2.639	3.944	9.623	24.034

2.2　二次优化——基于改进的 Montgomery 点乘算法

对 Montgomery 点乘研究发现,计算中需定义两个变量,两次求逆。本节对 Montgomery 点乘给予改进,改进后 ECC 加密程序为二次优化程序。

算法　改进 Montgomery 点乘。

输入:$(k_0,k_1\cdots k_{t-1})_2$ 且 $k_0=1$, $J=(x,y)\in E(F_2^m)$,J 为基点坐标。

输出:KJ。

a)$x_1\leftarrow x$,$Z_1\leftarrow 1$,$Z_2\leftarrow x^2$,$x_2\leftarrow Z_2^2+b$。

b)对于 i 从 $t-2$ 到 0,重复执行:

(a)若 $K_i=1$,则 $x_3\leftarrow x_1Z_2$,$y_3\leftarrow x_3x_2Z_1$,$Z_1\leftarrow x_2Z_1$,$Z_1\leftarrow(x_3+Z_1)^2$,$x_1\leftarrow x\,Z_1+y_3$;

$x_3\leftarrow x_2^2$,$y_3\leftarrow Z_2^2$,$Z_2\leftarrow x_3y_3$,$x_2\leftarrow x_3^2+b\,y_3^2$。

(b)否则 $x_3\leftarrow x_2Z_1$,$y_3\leftarrow x_3x_1Z_2$,$Z_2\leftarrow x_1Z_2$,$Z_2\leftarrow(x_3+Z_2)^2$,$x_2\leftarrow xZ_2+y_3$;

$x_3\leftarrow x_1^2$,$y_3\leftarrow Z_1^2$,$Z_1\leftarrow x_3y_3$,$x_1\leftarrow x_3^2+by_3^2$。

c) $y_3\leftarrow(xZ_1Z_2)^{-1}$,$x_3\leftarrow(x_1xZ_2y_3)$。

d) $x_1\leftarrow(xZ_1+x_1)$,$x_2\leftarrow(xZ_2+x_2)$,$x_1\leftarrow x_1x_2$,$x_2\leftarrow(x_2+y)Z_1Z_2$,$x_1\leftarrow x_1+x_2$,$x_2\leftarrow x+x_3$,$y_3\leftarrow x_1x_2y_3+y$。

e)返回(x_3,y_3)。

算法优化后,无中间变量,可节省内存资源与耗时。令整数 $K=(k_0,k_1,\cdots,k_{t-1})_2$,记 $l=\lceil\log_2K\rceil$,整个算法共需要 $GF(2^m)$ 中一次求逆、$6l+10$ 次乘法、$3l+7$ 次加法和 $5l+3$ 个平方运算。表 5 为改进 Montgomery 算法用时,表 6 为对应的二次优化后完整程序运行用时。

表 5　改进 Montgomery 点乘算法耗时

	$m=163$	$m=233$	$m=283$	$m=409$	$m=571$
耗时/ms	243	582	846	1 460	3 899

表 6　二次优化后程序执行耗时

	$m=163$	$m=233$	$m=183$	$m=409$	$m=571$
耗时/s	0.982	2.262	3.385	5.986	15.419

2.3　三次综合优化

通过优化二元域平方等算法也可有效节约程序运行耗时，以 $m=163$ 曲线为例，在改进 Montgomery 算法中共一次点乘需二元域乘法 $161\times7+12=1\ 139$ 次，二元域平方 $161\times5=805$ 次。本节首先采用创建查找表法进行平方计算优化。传统二元域计算如图 6 所示，令二进制多项式为 $a(z)\ =\ a_{m-1}z^{m-1}+a_{m-2}z^{m-2}+\ \cdots+a_2z^2+\ a_1\,z+a_0$，则 $a(z)^2$ 计算通过往 $a(z)$ 二进制表示中相邻位之间插入 0 即可。

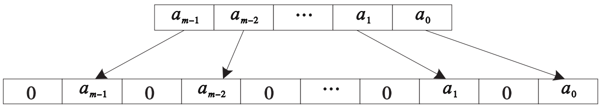

图6　二进制多项式平方

上述方法原理是采用对应字长数值循环移位异或方式计算，此法优点是内存占用较少，但运算速度并不理想。可通过对每一字节的平方预计算，制成表 T 共含 2^8 个 16 位数据，从而把 8 位多项式变换扩充为 16 位对应的多项式，查找表如表 7 所示。

表 7　二进制平方查找

十进制	二进制	平方后二进制	十六进制
0	0	0000000000000000	0X 0
1	1	0000000000000001	0X 1
2	10	0000000000000100	0X 4
⋮	⋮	⋮	⋮
254	11111110	1010101010101000	0X5554
255	11111111	1010101010101010	0X5555

查找表 T 共 256 项，制作过程可通过观察十六进制表示寻找规律，扩展过程总是以 0、1、4、5 循环进行，具体不再详述。基于查找表的多项式平方算法如下。

算法　查找表法求多项式平方。

输入：次数低于 m 的多项式 $a(z)$。

输出：$c(z)=a(z)^2$。

a) 对于 i 从 0 到 $t-1$，t 为数组 a 的度，重复执行：

(a) 令 $A[i]\ =\ (u_3,u_2,u_1,u_0)$，其中 u_i 是一个字节；

(b) $c[2i]\leftarrow(T(u_3),T(u_2))$，$c[2i+1]\leftarrow(T(u_1),T(u_0))$。

b) 返回 c。

实验证明利用查找表二进制平方算法可有效降低平方算法用时，如表 8 所示。

表 8　二进制平方运行用时对照

	$m=163$	$m=233$	$m=283$	$m=409$	$m=571$
插入法/us	16	21	22	24	34
查表法/us	5	10	13	9	26

此外，在程序中适时地利用嵌入汇编代码优化，也可提高程序执行效率。例如，在 C* Core 指令集中 cmpnei 命令执行效率高于 cmphs 命令，其两者功能相同，作替换后可节省程序执行时间；通过打开循环语句可减少数据压入堆栈的操作，也能较大幅度提高程序执行效率。本文在二元域乘法程序中即采用打开二重循环语句方法进行优化。

通过查找表平方算法与代码综合优化，程序点乘算法运行效率得到了较大幅度提高，所得程序为三次优化程序。表 9 为三次优化后改进 Montgomery 算法点乘算法用时。

表 9　三次优化后点乘算法用时

	用时/ms				
	$m=163$	$m=233$	$m=283$	$m=409$	$m=571$
改进 Montgomery	66	204	304	541	1 955

3　结束语

本文相关程序在 C* Core 芯片中运行，主频为 130 MHz，所得精确时间为在可编程中断定时器（PIT）获取。表 10 反映了五个不同域（$GF(2^{163})$、$GF(2^{233})$、$GF(2^{283})$、$GF(2^{409})$、$GF(2^{571})$）下 ECC 椭圆曲线程序运行用时。

表 10　ECC 执行用时对照

	用时/s				
	$m=163$	$m=233$	$m=283$	$m=409$	$m=571$
初始程序	4.978	10.512	19.458	57.258	78.589
一次优化	1.165	2.639	3.944	9.623	24.034
二次优化	0.982	2.262	3.385	5.986	15.419
三次优化	0.268	0.801	1.222	2.226	7.768

通过对比表中数据可知，各域下程序优化效果明显。图 7 为五个域下各程序耗时分布显示，形象地反映了初始程序及各次优化后的耗时分布。

通过图 7 对比可知，首次优化效率最高，平均达 75% 之多，图 8 为各域下三次优化程序优化率，其中不同曲线分别对应三次优化，图中纵坐标为提高的效率百分比，横坐标为域，具体效率见图中标注。

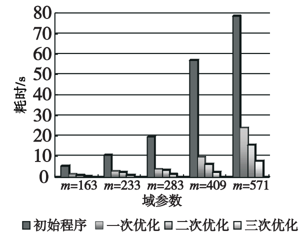

图7　ECC各域程序耗时

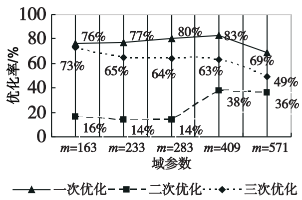

图8　ECC加密算法实现各域优化率

参考文献：

[1]　王宜怀，朱巧明，郑茳. C* Core 与 M* Core 的嵌入式系统[M]. 北京：清华大学出版社，2006：10-19.

[2]　苏州国芯科技有限公司. CCM3118DQ advance information rev. 3[R]. 2010.

[3]　钱丹，李飞，路而红，等. 基于 C* Core 动态内存分配方案与实现[J]. 电子设计工程，2011，19(13)：33-35.

[4]　周玉洁，冯登国. 公开密钥密码算法及其快速实现[M]. 北京：国防工业出版社，2002：96-97.

[5]　乔纳森·卡茨，耶胡达·林德尔. 现代密码学原理与协议[M]. 任伟，译. 北京：国防工业出版社，2011：78-80.

[6]　HANKERSON D, MENEZES A, VANSTONE S. Guide to elliptic curve cryp-tography[M]. New York：Springer-Verlag，2004：76-86.

[7]　KONHEIM A G. 计算机安全与密码学[M]. 唐明，译. 北京：电子工业出版社，2010：378.