Mobile Netw Appl (2007) 12:271–279
DOI 10.1007/s11036-007-0021-5

Hardware Organization to Achieve High-Speed Elliptic Curve Cryptography for Mobile Devices

Sining Liu · Brian King · Wei Wang

Published online: 28 September 2007

Abstract Elliptic curve cryptography (ECC) is recognized as a fast cryptography system and has many applications in security systems. In this paper, a novel sharing scheme is proposed to significantly reduce the number of field multiplications and the usage of lookup tables, providing high speed operations for both hardware and software realizations.

Keywords elliptic curve cryptography · cryptographic hardware organization · lookup table

1 Introduction

Elliptic curve cryptography (ECC) was proposed independently by Kolitz [12] and Miller [18] in 1985. Compared with other commonly used public key cryptosystems such as RSA and discrete logarithm-based cryptosystems, ECC is recognized as having a faster key exchange and signature generation. Furthermore, ECC requires smaller bandwidth, smaller key size and faster computation. The basic operations of ECC are "double" and "add", which consist of field multiplications, squares, additions and inverses.

One popularly used ECC cryptosystem is based on the finite field $GF(2^n)$. There are several different methods to implement such ECC systems. One way is to use the standard affine coordinate system. A more efficient method is to use projective point arithmetic to reduce the number of field inversions. The Lopez and Dahab method proposed in [16] is considered one of the best projective point methods in terms of performance. Since the Lopez and Dahab method requires several field multiplications to perform elliptic curve operations and since an efficient method to perform a $GF(2^n)$ field multiplication is to construct it as a lookup table (LT)-based field multiplications, LT generation and access is time-consuming. In order to achieve high-speed implementation of projective point ECC cryptosystems, it is important to reduce the number of multiplications and their LT usage.

This paper is an expanded version of our paper [15]. In this paper, a novel sharing scheme is proposed to reduce the number of field multiplications and the number of lookup tables required by projective point implementation of ECC. The proposed method significantly reduces the lookup table generation and access time thus providing high-speed operations for both hardware and software realizations.

2 Cryptography and mobile devices

Due to limited resources in mobile devices yet the need to possess the capabilities of public-key cryptography, we have seen standard bodies, manufacturers, and service providers utilize elliptic curve cryptography. In the construction of the *Wireless Application Protocol (WAP)* (the wireless Internet),[1] elliptic curve

S. Liu · B. King (✉) · W. Wang
ECE Department, Indiana University-Purdue University Indianapolis, Michigan SL 160, Indianapolis, IN 46202, USA
e-mail: briaking@gmail.com

[1] WAP has since been added to the Open Mobile Alliance (http://www.openmobilealliance.org/).

cryptography was added to the WTLS specification (the wireless version of TLS) [2]. The use of elliptic curve has been explored on many different mobile devices including PDA's, cell phones, and pagers, ECC has even been explored for use on active RFID devices [3].

The use of public-key cryptography by mobile devices could vary, possible uses include key-exchange, signature generation, signature verification, or content encryption (perhaps by using the hybrid cryptosystem ECIES [1]). Clearly if mobile devices are required to authenticate frequently then the implementation of the public-key cryptosystem can make substantial impact on the device and its resources. Cryptosystem choices could impact memory usage, bandwidth, and due to the computational complexity impact the battery power of the device. Frequent authentication could occur if the device is mobile within some type of wireless ad-hoc network, and is required to authenticate whenever it requests the resources of the wireless ad-hoc network. In another scenario, in Messerges and Dabbish [17] described a digital rights management system for mobile devices (3G phones, personal devices, etc.). In this scenario, in order to access content (very often content that is copyright protected), the user/device will need to authenticate. In another scenario, health-care information requires privacy protection, mobile devices used to transmit highly-sensitive healthcare information will need to protect that information, key-exchange and authentication protocols may be required. In general, as we rely more and more on mobile devices working in ad-hoc environments and wishing to access/transmit content that requires privacy and authentication, we will see a rise in the use of ECC.

3 Elliptic curve cryptography (ECC) over $GF(2^n)$

The essential task of ECC [4] is to compute the scalar multiple kP, where k is a scalar, and P is a point on the elliptic curve. This paper will focus on elliptic curves defined over $GF(2^n)$. The Weierstrass equation for a non-supersingular elliptic curve is:

$$y^2 + xy = x^3 + ax^2 + b \tag{1}$$

where $a, b \in GF(2^n)$ and $b \neq 0$.

The set of all points $P = (x, y)$, where $x, y \in GF(2^n)$, together with a point $\mathcal{O}$, the point of infinity, forms an additive abelian group G, in which $\mathcal{O}$ is the identity element. Here addition in G is defined by: for all $P \in G$

1. $P + \mathcal{O} = P$,
2. for $P = (x, y) \neq \mathcal{O}$, the negative of P satisfies $-P = (x, x + y)$
3. and for all $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, where $P_1 \neq \mathcal{O}$, $P_2 \neq \mathcal{O}$ then $P_1 + P_2 = P_3 = (x_3, y_3)$ where $x_3, y_3 \in GF(2^n)$ and,

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \tag{2}$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \tag{3}$$

Here $\lambda = x_1 + \frac{y_1}{x_1}$ when $P_1 = P_2$ and $\lambda = \frac{y_1 + y_2}{x_1 + x_2}$ when $P_1 \neq \pm P_2$. The computation of kP is performed by expressing k in a binary form $k = k_{J-1}k_{J-2}\ldots k_1k_0$ and applying the "double" and "add" method. That is,

$$kP = 2(\cdots 2((2k_{J-1}P) + k_{J-2}P) + \cdots) + k_0P.$$

The "double" requires two field multiplications, one square, and one inverse.[2] The "add" operation requires two field multiplications, 1 square, and 1 inverse.

3.1 Arithmetic in $GF(2^n)$

In the finite field $GF(2^n)$, a field addition is simply an XOR operation. The computation of the field square depends on the basis representation of the field. One can use a normal basis or a polynomial basis. Most implementations will use a polynomial basis. Throughout this paper we will use a polynomial basis to represent $GF(2^n)$ field elements. In a polynomial basis, the field square consists of two steps. Supposing $u = u_{n-1}X^{n-1} + \cdots + u_1X + u_0$, where $u_i \in \{0, 1\}$, we have $u^2 = u_{n-1}X^{2n-2} + u_{n-2}X^{2n-4} + \cdots + u_1X^2 + u_0$. One can first "insert zeros" into the bitstring $u_{n-1}\ldots u_1u_0$, which yields $u_{n-1}0\cdots0u_10u_0$. Then, a modular reduction algorithm is called to return the final result. The insertion of zeros and reduction steps can be implemented very efficiently in both software and hardware. The field multiplication is usually much slower than the field square and requires more resources than the field square and field addition. Depending on the implementation, a field multiplication can six-to-ten times longer than a field square [10]. A common method to implement an efficient field multiplication is to use a lookup table (LT) [7]. The product of $\alpha, \beta \in GF(2^n)$ is generally computed as the LT stores 2^g products of α with scalars $b = 0, 1, \cdots, 2^g - 1$. Then, β can be shifted as g-bit groups instead of a bit-by-bit computation. Each group will require one access of the LT and one shift-and-add operation. After $\lceil \frac{n}{g} \rceil$ of such LT accesses and shift-and-add operations, the product of this field multiplication is obtained. Although there exists efficient methods to

[2] These requirements reflect the most efficient add and double when using affine coordinates.

compute inverses in $GF(2^n)$ [8, 20], the computation of an inverse will take significantly more time than a multiplication, an efficient implementation of a field inverse may take anywhere from eight to ten times longer than a field multiplication [10].

3.2 ECC based on affine and projective point arithmetic

There are several ways to compute the scalar multiple kP. One is to use the traditional affine arithmetic. An alternative is to use the projective point coordinates. Using the affine arithmetic to compute kP, many "double" or "add" operations are used, each requiring a field inverse; whereas, using the projective point coordinates, a complete computation of kP only requires one inverse operation. As the field inverse has been benchmarked to be significantly slower than a field multiplication, the projective point method is more efficient than the affine method, because the number of field inverses required has been reduced to one [6, 9, 14].

Projective Point ECC: A projective plane over a field $\mathcal{F}$ can be defined by fixing positive integers α, β and creating an equivalence relation $(x, y, z) \sim (X, Y, Z)$ if $(x, y, z) = (\lambda^{\alpha} X, \lambda^{\beta} Y, \lambda Z)$, where $\lambda \in \mathcal{F}$, $\lambda \neq 0$. Each affine point (x, y) can be mapped to the projective plane by $\phi : (x, y) \rightarrow (x, y, 1)$. Then the $Image(\phi)$ comprises all equivalence classes (X, Y, Z), $Z \neq 0$. Each equivalence class in the $Image(\phi)$ can be mapped back to the affine point by $x = \frac{X}{Z^{\alpha}}$, $y = \frac{Y}{Z^{\beta}}$.

There exist many different mappings of projective points, amongst which are three popular ones for ECC applications: (1) the homogeneous projective point transformation [19], where $x = \frac{X}{Z}$, $y = \frac{Y}{Z}$, (2) the Jacobian method [5], where $x = \frac{X}{Z^2}$, $y = \frac{Y}{Z^3}$, and (3) the Lopez and Dahab method [16], where $x = \frac{X}{Z}$, $y = \frac{Y}{Z^2}$. Table 1 describes the computational requirements for each of these projective point methods in terms of field operations of multiplication and square. As shown in Table 1, the Lopez and Dahab method requires the least number of multiplications and is considered the most efficient projective point method amongst the three. The reason is that the number of multiplications will be the dominant factor to determine the complexity and speed of these projective point ECC methods.

Recall from Section 3.1, the product of $\alpha, \beta \in GF(2^n)$ is computed using a LT scheme. The LT stores 2^g products of α, and β is shifted as g-bit groups, instead of a bit-by-bit computation. Each group will require one access of the LT and one shift-and-add operation.

Table 1 Comparison of computational requirements

		No. of mult.	No. of squares
Homogeneous	Double	7	5
	Add	13	1
Jacobian	Double	5	5
	Add	11	4
Lopez and Dahab	Double	5	5
	Add	10	5

After $\lceil \frac{n}{g} \rceil$ of such LT accesses and shift-and-add operations, the product of this field multiplication is obtained. For example, in $GF(2^{163})$, the parameter $g = 4$ is typically chosen and one field multiplication will require 41 iterations of lookup table access. Also, note that a multiplication will require generation of a LT, which is another nontrivial overhead. Since the field multiplication is the most time-consuming operation in a projective point implementation, the time required by the field multiplications will determine the speed for projective point ECC systems.

Based on the above analysis and Table 1, the Lopez and Dahab method is considered as the best existing projective point ECC method. However, it still operates at considerably low speed. For $n = 163$ and $g = 4$, assuming key k has an equal number of ones and zeros, to perform the kP computation will require over 60,000 LT accesses and over 1,600 LT generations each requiring 2^4 entries. Therefore, it is critically important to reduce the number of field multiplications and to improve the efficiency of LT usage.

4 Projective point ECC based on look-up table sharing

In this section, we propose a novel LT sharing technique to improve the Lopez and Dahab method.

4.1 Lopez and Dahab algorithm

We briefly describe the Lopez and Dahab algorithm. P is an affine point (x_1, y_1) and is transformed to the projective point $(x_1, y_1, 1)$ when using the projective point method. An internal variable $Q = (x_2, y_2, z_2)$ is used to store intermediate results. $R = (x_3, y_3, z_3)$ represents an output. Doubling a point, $R = 2Q$, is given by:

$$z_3 = z_2^2 x_2^2$$

$$x_3 = x_2^4 + b z_2^4$$

$$y_3 = b\, z_2^4 z_3 + x_3\left(a z_3 + y_2^2 + b z_2^4\right) \tag{4}$$

Adding two points, $R = P + Q$ where $P \neq \pm Q$, is given by:

$$
\begin{aligned}
z_3 &= z_2^2(z_2x_1 + x_2)^2 \\
x_3 &= \left(z_2^2y_1 + y_2\right)^2 + z_2\left(z_2x_1 + x_2\right)\left(z_2^2y_1 + y_2\right) \\
&\quad + z_2\left(z_2x_1 + x_2\right)^3 + az_2^2\left(z_2x_1 + x_2\right)^2 \\
y_3 &= z_2\left(z_2x_1 + x_2\right)\left(z_2^2y_1 + y_2\right)\left(z_3x_1 + x_3\right) \\
&\quad + z_3x_3 + z_3^2y_1 \qquad (5)
\end{aligned}
$$

Thus the "double" and "add" algorithms will require 5 and 10 field multiplications, respectively, as shown in Table 1. Observe that we can assume that the ECC curve parameter a is either 0 or 1.[3]

4.2 Proposed lookup table sharing method

We now apply a novel LT sharing technique to improve the Lopez and Dahab method. The following notations are needed. $k = k_{J-1}k_{J-2}\ldots k_1k_0$ is a scalar where J is the key size. The size of J is typically the same as that of n or very close, we will assume $J = n$ hereafter. $\mathtt{LT}(\alpha)$ denotes the lookup table for a field element α such that $\mathtt{LT}(\alpha) = \{b\alpha | b = 0, \cdots, 2^g - 1\}$. Calling the procedure "`make LT`(α)" will create $\mathtt{LT}(\alpha)$ in the designated memory region. *Reg* is a shared n-bit register, and *Cache* is a shared memory region to store a LT. *Static* and *Dynamic* are memory regions to store static and dynamic LTs, respectively. Using these notations, the proposed "double", "add" and kP algorithms are shown in Algorithms 1, 2 and 3, respectively.

Our proposed "double" (Algorithm 1) uses four multiplications and three LTs. This is a significant reduction from the Lopez and Dahab double algorithm (4), which requires five multiplications, i.e., $b \cdot A$, $z_2 \cdot x_2$, $z_3 \cdot B$, $z_3 \cdot x_3$, and $x_3 \cdot C$, and 5 LTs. The reduction is achieved as follows. First, since b is a constant, by using a static LT(b), the $b \cdot A$ operation does not require a dynamic LT generation. Then, in step 4, the product $z_2 \cdot x_2$ is in fact the $z_3 \cdot x_3$ generated in the previous step. When $k_{i+1} = 1$, this product comes from the previous "add". When $k_{i+1} = 0$, this product comes from the previous "double". We assign a register *Reg* to enable sharing of this value. The initial value of *Reg* is x_2 since z_2 is 1. Afterwards, this register will be updated by $z_3 \cdot x_3$ generated, which represents $z_2 \cdot x_2$ in the next usage of Algorithm 1. Thus, we never need to compute $z_2 \cdot x_2$ and one multiplication is saved. The rest of the multiplications: $z_3 \cdot B$, $z_3 \cdot x_3$, and $x_3 \cdot C$ will require *Cache* and LT(x_3) created dynamically. *Cache* is used twice as in lines 16 and 17. Based on the above analysis, only three LTs and 4 multiplications are required in this proposed "double" algorithm.

Algorithm 1 EC Double $R = 2Q$

1:	$A \longleftarrow z_2^2$	
2:	$A \longleftarrow A^2$	
3:	$A \longleftarrow b \cdot A$	{use static LT(b)}
4:	$C \longleftarrow Reg$	{ *Reg* stores $z_2 \cdot x_2$ }
5:	$z_3 \longleftarrow C^2$	
6:	$B \longleftarrow x_2^2$	
7:	$B \longleftarrow B^2$	
8:	$x_3 \longleftarrow A + B$	
9:	$C \longleftarrow y_2^2$	
10:	$C \longleftarrow C + A$	
11:	**if** $a = 1$ **then**	
12:	$C \longleftarrow C + z_3$	
13:	**else**	
14:	$C \longleftarrow C$	
15:	*Cache* $\longleftarrow$ make LT(z_3)	
16:	$B \longleftarrow z_3 \cdot B$	{use *Cache*}
17:	$Reg \longleftarrow z_3 \cdot x_3$	{use *Cache*}
18:	$A \longleftarrow Reg + B$	
19:	$B \longleftarrow x_3 \cdot C$	{use *Dynamic*)}
20:	$y_3 \longleftarrow A + B$	

The Lopez and Dahab "add" algorithm (4) requires 10 multiplications and we reduced this to 9 by simply reorganizing the order and by using the fact that one can adopt the principle that ECC parameter a is either 0 or 1 [11] (see earlier comment). The reorganized algorithm (Algorithm 2) requires only five LTs: *Cache*, LT(y_1), LT(E), LT(z_3), and LT(F). LT(y_1) is static and used for two multiplications (steps 7 and 8). *Cache* is generated in the previous double (step 15 of Algorithm 1) and used for two multiplications here (steps 1 and 3 of Algorithm 2). Only the other three LTs are generated dynamically: LT(E) and LT(z_3) are used for two multiplications and LT(F) is used once. Using these Algorithms 1 and 2 a number of times, the final result of kP is obtained as shown in Algorithm 3.

To summarize, our "double" scheme uses four multiplications with two dynamic LTs and one static LT, and our "add" scheme uses nine multiplications with three dynamic LTs and one static LT. These dynamic LTs can be reused in one physical location. Also, Register *Reg* and LT *Cache* are shared by the current and the previous steps of both "double" and "add". Totally, we use one dynamic LT, two static LTs, one *Cache* LT and

[3] If $a \neq 0$ or 1, then the curve E is isomorphic to an elliptic curve for which the ECC parameter a belongs to $\{0, 1\}$. Thus one performs the scalar multiple in this curve and then make the transformation back to the original curve. This would require only one field multiplication [11].

Algorithm 2 EC Add $R = P + Q$

1: $A \longleftarrow z_2 \cdot x_1$ {use *Cache*= current LT(z_2)}
2: $A \longleftarrow A + x_2$
3: $E \longleftarrow z_2 \cdot A$ {use *Cache*= current LT(z_2)}
4: $D \longleftarrow E^2$
5: $B \longleftarrow z_2^2$
6: $C \longleftarrow D^2$
7: $B \longleftarrow y_1 \cdot B$ {use static LT(y_1)}
8: $C \longleftarrow y_1 \cdot C$ {use static LT(y_1)}
9: $B \longleftarrow B + y_2$
10: $F \longleftarrow B^2$
11: $A \longleftarrow A^2$
12: *Dynamic* $\longleftarrow$ make LT(E)
13: $B \longleftarrow E \cdot B$ {use LT(E)}
14: $G \longleftarrow E \cdot A$ {use LT(E)}
15: $F \longleftarrow F + B + G$
16: **if** $a = 1$ **then**
17: $x_3 \longleftarrow F + D$
18: **else**
19: $x_3 \longleftarrow F$
20: $z_3 \longleftarrow D$
21: *Cache* $\longleftarrow$ make LT(z_3)
22: $Reg \longleftarrow z_3 \cdot x_3$ {use LT(z_3)}
23: $F \longleftarrow z_3 \cdot x_1$ {use LT(z_3)}
24: $F \longleftarrow F + x_3$
25: $F \longleftarrow F \cdot B$ {use *Dynamic*)}
26: $y_3 \longleftarrow F + C + Reg$

Algorithm 3 EC scalar multiple kP based on LT sharing

1: *Static I* $\longleftarrow$ make LT(b)
2: *Static II* $\longleftarrow$ make LT(y_1)
3: $Q = (x_1, y_1, 1)$ {$Q \longleftarrow P$}
4: $Reg \longleftarrow x_1$
5: $i \longleftarrow n - 2$ {here we assume $k_{n-1} = 1$}
6: **while** $i \geq 0$ **do**
7: $Q \longleftarrow 2Q$
8: **if** $k_i = 1$ **then**
9: $Q \longleftarrow Q + P$
10: $i \longleftarrow i - 1$
11: $temp \longleftarrow z_2^{-1}$
12: $\text{Output}_x \longleftarrow x_2 \cdot temp$
13: $\text{Output}_y \longleftarrow y_2 \cdot temp^2$

a register *Reg*. The reduction in terms of the number of multiplications will speed up the whole process. The LT and register sharing will significantly reduce the time-consuming LT generation and access time. Thus, high-speed performance can be achieved for both hardware implementation and software implementation.

Based on the proposed algorithms, the block diagram of the proposed ECC system structure is shown in Fig. 1. It consists of a double module and an add module, as well as *Reg* and *Cache* shared between them. In an implementation, a LT takes up $2^g \cdot n$ bits of memory and *Reg* requires only n bits of memory.

This structure represents slightly more memory usage than the original Lopez Dahab method, but this is the trade-off in order to achieve a high-speed cryptosystem. If the mobile device utilizes a fixed curve than the *Static I* LT (used for $LT(b)$ in the EC Double) can be precomputed and stored in ROM. Further if the computation is using the base point P of the fixed elliptic curve then the other static table *Static II* could be stored in ROM as well. Assuming that all static tables are stored in RAM, then the hardware organization for the ECC consists of four LT's which is $2^g \cdot n$ bits of memory and *Reg* which requires n bits of memory. The four LT's in our organization are: *Cache, Static I, Static II,* and *Dynamic*. The *Cache* LT is used to generate a LT in the EC double and then we reuse this same LT in the EC add (whenever the EC add is required). The *Dynamic* LT is used to store a dynamically generated LT. By examining the algorithms (Algorithms 1 and Algorithm 2) closely it is clear that we could reduce the number of LT's by one. That is, if we re-order a few steps in Algorithm 1, then we could use the same memory location for the *Cache* LT and *Dynamic* LT. The only conflict for using the same memory location for *Dynamic* and *Cache* occur when we execute the EC Double and then execute the EC add. The conflict occurs as follows: In Algorithm 1-line15 the contents of the *Cache* is set. These contents are used in Algorithm 1-line16, Algorithm 1-line17, Algorithm 2-line1,

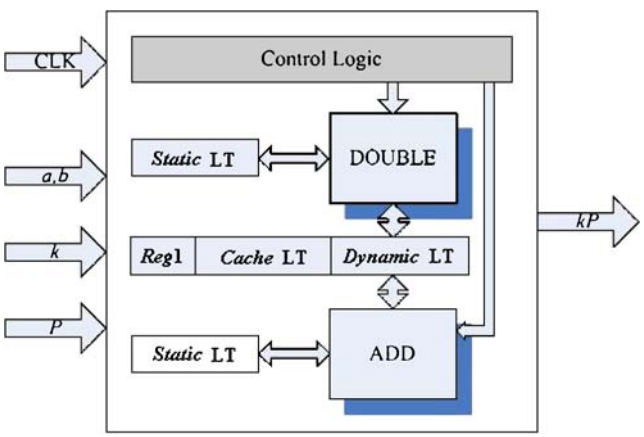

Figure 1 The proposed ECC hardware organization

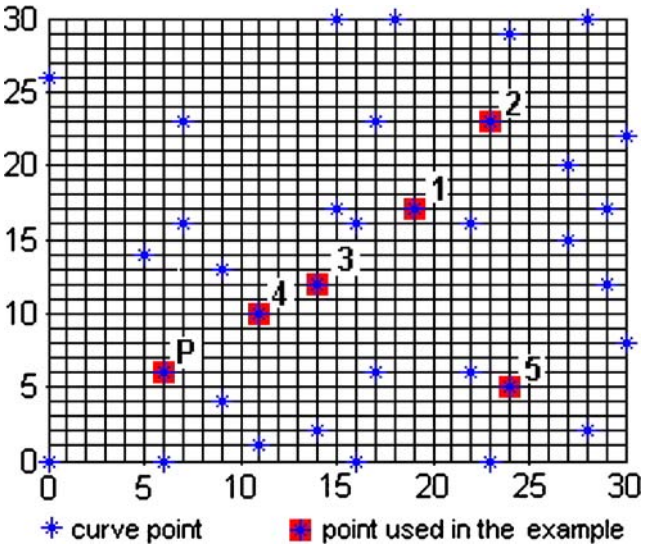

Figure 2 EC points and path of computation. Recall that we are using integers to represent polynomials. For example, $14 = (1110)_2 = X^3 + X^2 + X$

and Algorithm 2-line3. The *Dynamic* LT is used in Algorithm 1-line19. Observe that both multiplicands Algorithm 1-line19 are independent of the calculations made in Algorithm 1-line16 and Algorithm 1-line17. Therefore by performing Algorithm 1-line19, prior to Algorithm 1-line15, Algorithm 1-line16, Algorithm 1-line17, and Algorithm 1-line18 we could use the same memory location for *Dynamic* and *Cache*. If we did this we would have to change Algorithm 1–line19 to $F \longleftarrow x_3 \cdot C$. Algorithm 1-line20 would change to $y_3 \longleftarrow A + F$.

15' $F \longleftarrow x_3 \cdot C$ {use *Dynamic*)}
16' *Cache* $\longleftarrow$ make LT(z_3)
17' $B \longleftarrow z_3 \cdot B$ {use *Cache*}
18' $Reg \longleftarrow z_3 \cdot x_3$ {use *Cache*}
19' $A \longleftarrow Reg + B$
20' $y_3 \longleftarrow A + F$

Altogether the memory resource requirements for our hardware organization is (we highlight only the memory need for the ECC operations, omitting an analysis of the memory requirements to implement the field operations that are needed in the ECC operation): three LT's, *Reg*, and A, B, C, D, E, F requiring $3 \cdot 2^g \cdot n + 7 \cdot n$ bits of memory. In the original Lopez Dahab algorithm using a LT, we would require $2^g \cdot n + 7 \cdot n$ bits of memory, thus we see for a very slight increase in memory $2 \cdot 2^g \cdot n$ bits, we gain significant performance improvements. The performance analysis of our examples are provided in Section 5.

Example 1 We construct a small scalar multiple example to illustrate the flow of the proposed algorithms and the benefits of the proposed sharing techniques. The elliptic curve is chosen to be $y^2 + xy = x^3 + x^2 + 3$ mod $F(x) = x^5 + x^2 + 1$. Note that the coefficients are integer representation of polynomials, i.e., "3"$= (11)_2 = X + 1$, and we will adopt this convention throughout this work. There are 38 points on the curve, shown as stars in Fig. 2. The point of infinity $\mathcal{O}$ is illustrated by (0,0). $P = (x_1, y_1) = (6, 6)$ is a generator. Let $k = 18$ (or "10010"), we will see how kP is computed by using the proposed algorithms 1, 2 and 3. For $k = 18$, Algorithm 3 will follow such a sequence : "double-double-double-add-double", and the resulting points are plotted in Fig. 2 using squares, each with a number indicating its temporal sequence. Table 2 illustrates how the LTs and the shared register *Reg* evolve during the execution of Algorithm 3. The results presented are consistent with those given by the Lopez and Dahab algorithm, which confirms the correctness of our algorithm.

As noted, in certain cases the first two operations in Table 2 (line 1, 2) can be skipped if the LTs are precomputed and stored in ROM. This holds for those applications that the curve is fixed and P is a base point stipulated by some standards. Even if the above assumption does not hold, the cost for computing the two LTs is trivial because they are computed only once in each computation of kP. From the table, it is also

Table 2 Contents of *Reg* and LTs during computing kP

Line index	Static I	Static II	Reg	Cache	Dynamic	Output Point[a]
1	LT(3)	–[b]	–	–	–	–
2	LT(3)	LT(6)	–	–	–	–
4	LT(3)	LT(6)	6	–	–	–
7	LT(3)	LT(6)	1	LT(20)	–	(19, 17)
7	LT(3)	LT(6)	23	LT(1)	–	(23, 23)
7	LT(3)	LT(6)	15	LT(24)	–	(14, 12)
9	LT(3)	LT(6)	24	LT(22)	LT(29)	(11, 10)
7	LT(3)	LT(6)	14	LT(7)	LT(29)	(24, 5)

[a]Converted back to normal coordinates
[b]Value uninitialized

seen that in these steps, the "double" algorithm uses and updates *Reg* and *Cache* but does not use *Dynamic*, whereas the add algorithm utilizes and updates all of them including the *Dynamic*. This demonstrates that the "add" accessed the dynamic LT more frequently than the "double".

5 Performance analysis

5.1 Performance analysis and comparison

The above analysis demonstrates the reduction of multiplications in "double" and "add". Using this result, we now compare the proposed algorithms with the Lopez and Dahab method for the complete ECC system in terms of a scalar multiple kP. It is noted that in Algorithm 3, when $k_i = 1$, an EC Add is necessary; when $k_i = 0$, this EC Add will be skipped (steps 8 and 9). Therefore, for the cases of $GF(2^n)$ and k is uniformly distributed, the number of EC Double will be n and that of EC Add is $[\frac{n}{2}]$. This technique is applied to both Lopez and Dahab and the proposed method. Therefore, as shown in Table 3, the number of multiplications required by Lopez and Dahab is approximately $5n + [\frac{n}{2}] \cdot 10$, while that of the proposed method is $4n + [\frac{n}{2}] \cdot 9$. This leads to 15% reduction in terms of number of multiplications, thus offering higher operating speed. In order to achieve this reduction, the proposed method will use one dynamic LT, two static LTs, one *Cache* LT and one *Reg*, while the Lopez and Dahab method requires a dynamic LT.

A more significant improvement to achieve high-speed performances is in terms of the number of LT generations and accesses, in another word, average execution time. Recall that we used 2^g to represent the number of entries in a LT, where g corresponds to the number of operand-bits examined at a time. Therefore, $\lceil \frac{n}{g} \rceil$ represents the number of LT access for a given field multiplication. If we measure the execution time of a scalar multiple kP solely in terms of field multiplications, then execution time equals LT generation time plus the number of LT access × time per access, i.e.,

$$\begin{aligned}\text{Exec. Time} &= \text{Total Time of Field Mult.} \\ &= \left\{ \begin{pmatrix} \text{no. of} \\ \text{mult.} \end{pmatrix} \cdot \left\lceil \frac{n}{g} \right\rceil \right\} \Delta_1 + \left\{ \begin{pmatrix} \text{no. of} \\ \text{LT Gen.} \end{pmatrix} \cdot 2^g \right\} \Delta_2\end{aligned}$$

where $\lceil \frac{n}{g} \rceil$ is no. of LT access per mult., Δ_1 is the time constant for every LT access, 2^g is the number of entries per LT, and Δ_2 is the average time to compute one entry in the LT. For an FPGA, both Δ_1 and Δ_2 equal to one clock period.

In this work we only consider the case when the entire LT is generated. Using the Lopez and Dahab method, $5n + [\frac{n}{2}] \cdot 10$ multiplications will require $5n + [\frac{n}{2}] \cdot 10$ LT generations. Thus, the execution time of the Lopez and Dahab scalar multiple algorithm is

$$\left(5n + 10\left[\frac{n}{2}\right]\right)\left(\left\lceil \frac{n}{g} \right\rceil \Delta_1 + 2^g \Delta_2\right) \tag{6}$$

The proposed method involves $4n + [\frac{n}{2}] \cdot 9$ multiplications. Since $2n$ multiplications in EC Double and $[\frac{n}{2}] \cdot 6$ in EC Add are implemented using static LT or shared cache and register, these operations do not require LT generations. Thus, only $(2n + [\frac{n}{2}] \cdot 3)$ LTs are generated, leading to a reduction of about 45% of LT generation overhead. Furthermore, the reduction in terms of number of multiplications and LT generations will in turn reduce the LT access time. The execution time of our scalar multiple algorithm is

$$\left\lceil \frac{n}{g} \right\rceil \left(4n + 9\left[\frac{n}{2}\right]\right) \Delta_1 + 2^g \left(2n + 3\left[\frac{n}{2}\right]\right) \Delta_2 \tag{7}$$

Table 3 provides a brief comparison between our algorithm and the Lopez and Dahab algorithm.

In this study, we assume that polynomial additions and squares are fast and can be ignored at the presence of polynomial multiplications. For a memory-constrained system such as a smart card, a small LT with $g = 4$ is usually chosen since a small LT requires modest amount of memory. Using software implementation, the addressing overhead for implementing $g = 4$ is less than those using other numbers that does

Table 3 Comparison of different implementation schemes

	No. of mult.	No. of LTs	No. of LT gen.	Total exec. time
Lopez and Dahab	$5n + [\frac{n}{2}] \cdot 10$	1 dyn. LT	$5n + [\frac{n}{2}] \cdot 10$	$(5n + 10[\frac{n}{2}]) \cdot (\lceil \frac{n}{g} \rceil \Delta_1 + 2^g \Delta_2)$
Proposed design	$4n + [\frac{n}{2}] \cdot 9$	4 LTs 1 Reg	$2n + [\frac{n}{2}] \cdot 3$	$\lceil \frac{n}{g} \rceil (4n + 9[\frac{n}{2}]) \Delta_1 + 2^g (2n + 3[\frac{n}{2}]) \Delta_2$

Table 4 FPGA implementation results for kP

	Hardware	Speed (ms)	Power (mW)
Lopez and Dahab	890 CLBs and 1 BRAM	2.3	890
Proposed design	910 CLBs and 1 BRAM	1.9	923

not represent bytes of half-bytes. Using hardware implementations, there is not such limitation since the address bus can be any wordlength. Therefore, g can be any number. For example, in certain hardware-based cryptosystems, a LT where one uses a g between 5 and 8. It is noted that there exists another strategy to choose g large [7]. In that case, if $\frac{n}{g} \ll 2^g$, it is more practical to construct a basis of the LT instead of the entire LT. However, this strategy is beyond the scope of our paper, and readers are referred to [7] for a detailed discussion.

6 FPGA implementation

As a case study, a 163-bit ECC with $g = 4$ is implemented using Xilinx FPGA technology. The proposed look-up-table sharing ECC system can offer efficient and high-speed FPGA realization. The proposed system only consists of "shift-add" logic and LT. Both "shift-add" and LT can be easily implemented using FPGA. The core generator tool in Xilinx ISE platform has several shift-add and look-up-table blocks available, which are used in our implementation. The implementation results are shown in Table 4 and compared with the Lopez and Dahab method. In the table, the number of configurable logic blocks (CLBs) shows logic resource usage. The size of a Block RAM (BRAM) in Virtex-II chip is 18K-bit, which can accommodate more than four 2^4 by 163-bit LTs. The proposed method is 20% faster while requiring similar hardware resource and power consumption. Since speed is one main challenge of the mobile phones and PDAs applications and high speed gain can be used to obtain low power consumption by lowering the supply voltage, this proposed system will be very useful for the consumer electronics industry [13].

6.1 Applications for mobile devices

Mobile devices such as PDAs and mobile phones, open the door to a slew of new commercial applications and services. Many secured applications use public key cryptography for providing authentication, integrity, non-repudiation and encryption. ECC (elliptic curve cryptography) is a particularly suitable public-key system for mobile devices as it has smaller key size and more robustness to different attacks compared with other cryptosystems [21–23]. In order to achieve wide applications of ECC in mobile devices, ECC implementation needs to satisfy the requirement of mobile devices. Due to the evolution of IC industry, the modern mobile devices can accommodate more and more memory elements with reduced cost. For the last forty years, the size of the IC has been scaled down by 50% each year according to the Moore's Law. The current IC technology for mobile phone and PDA applications is in the nanometer scale, allowing billions of transistors per IC. Thus, the area requirement of ICs is not critical. Instead, the speed and power consumption become the major concerns. The proposed reorganization of ECC will follow this trend. By introducing a modest amount of LUTs, the speed of ECC is significantly improved. Also, by reducing the number of multiplications, the ECC algorithm itself is simplified. This speed gain is useful for improving performance of security systems in mobile devices. Furthermore, one can tradeoff the speed gain to save power consumption. For example, we design this high speed ECC. Then, we can reduce the supply voltage of the chip and the speed will slow down. The speed is still acceptable but the power consumption that is related to square of voltages will be significantly reduced. Therefore, the applications of the proposed high-speed ECC will have a huge impact in improving both speed and power consumption of security system for mobile applications.

7 Conclusion

In this paper, a novel lookup table sharing scheme and hardware organization for ECC has been proposed, our work reduces the number of field multiplications and the usage of lookup tables required by projective point implementation of ECC. The proposed method significantly reduces the execution time for scalar multiple kP, thus providing high-speed operations. It is expected that the proposed lookup table sharing techniques and high speed ECC can be widely used in many security systems for mobile applications.

References

1. Institute for Electrical and Electronics Engineers (IEEE) (2000) Standard 1363-2000. Standard Specifications for Public Key Cryptography, January 2000

2. Wireless Application Protocol Forum (2007) WAP Wireless Transport Layer Security (WTLS) Specification, http://www1.wapforum.org/tech/documents/WAP-261-WTLS-20010406-a.pdf
3. Batina L, Guajardo J, Kerins T, Mentens N, Tuyls P, Verbauwhede I (2006) An elliptic curve processor suitable for RFID-tags. (IACR eprint, July 2006) http://eprint.iacr.org/2006/227.pdf
4. Blake IF, Smart N, Seroussi G (1999) Elliptic curves in cryptography. London mathematical society lecture note series, Cambridge University Press
5. Chudnovsky DV, Chudnovsky GV (1986) Sequences of numbers generated by addition in formal groups and new primality and factorization tests. Adv Appl Math 7:385–434
6. Gaudry P, Hess F, Smart N (2000) Constructive and destructive facets of weil descent on elliptic curves. HP Labs Technical Report no. HPL-2000-10. http://www.hpl.hp.com/techreports/2000/HPL-2000-10.html
7. Hasan MA (2000) Look-up table-based large finite field multiplication in memory constrained cryptosystems. IEEE Trans Comput 49(7):749–758
8. Hasan MA (2001) Efficient computation of multiplicative inverses for cryptographic applications. In: Proceedings of the 15th IEEE symposium on computer arithmetic, pp 66–72
9. Jacobson M, Menezes A, Stein A (2001) Solving elliptic curve discrete logarithm problems using weil descent. Combinatorics and Optimization Research Report, 31 May 2001, http://www.cacr.math.uwaterloo.ca
10. King B (2001) An improved implementation of elliptic curves over $GF(2^n)$ when using projective point arithmetic. Selected areas of cryptography, Springer-Verlag, pp 134–150
11. King B (2004) A point compression method for elliptic curves defined over $GF(2^n)$. Workshop on public key cryptography, pp 333–345
12. Koblitz N (1987) Elliptic curve cryptosystems. Math Comput 48(177):203–209
13. Leong PHW, Leung IKH (2002) A microcoded elliptic curve processor using FPGA technology. IEEE Trans on Very Large Scale Integr (VLSI) Syst 10(5):550–559, October
14. Lidl R, Niederreiter H (1997) Finite fields, 2nd edn. Cambridge Univ. Press
15. Liu S, King B, Wang W (2006) A CRT-RSA algorithm secure against hardware fault attacks. The 2nd IEEE international symposium on dependable, autonomic and secure computing (DASC'06)
16. Lopez J, Dahab R (1999) Improved algorithms for elliptic curve arithmetic in $GF(2^n)$. Selected areas in cryptography '98, SAC'98. LNCS 1556, Springer, pp 201–212
17. Messerges TS, Dabbish EA (2003) Digital rights management in a 3G mobile phone and beyond. In: Proceedings of the 3rd ACM workshop on digital rights management, pp 27–38
18. Miller VS (1985) Use of elliptic curves in cryptography. Advances in cryptology CRYPTO 1985, Springer-Verlag, New York, pp 417–420
19. Mullin AGR, Vanstone S (1991) On the development of a fast elliptic curve processor chip. Advances in cryptology – crypto '91, Springer, pp 482–487
20. Schroeppel R, Orman H, O'Malley SW, Spatscheck O (1995) Fast key exchange with elliptic curve systems. In: Advances in cryptology – CRYPTO '95. Lecture notes in computer science, vol 963. Springer, pp 43–56
21. Seroussi G (1998) Compact representation of elliptic curve points over F_{2^n}. HP Labs Technical Reports http://www.hpl.hp.com/techreports/98/HPL-98-94R1.html, pp 1–6
22. Silverman J (1986) The arithmetic of elliptic curves. Springer, New York
23. Singele D, Preneel B (2005) The wireless application protocol (WAP). Int J Netw Secur 1(3):161–165

Brian King received a Ph.D. in mathematics (1990) and a Ph.D. in Computer Science (2000). He is currently an assistant professor of Electrical and Computer Engineering at Indiana Univ. Purdue Univ. Indianapolis (IUPUI). Prior to joining IUPUI he worked in the Security Technologies La at Motorola Research Labs. His research interests include: wireless security, cryptography, threshold cryptography and low-complexity cryptosystems.

Wei Wang received his Ph. D degree in 2002 from Concordia University, Montreal, QC, Canada, both in Electrical and Computer Engineering. From 2000 to 2002, he served as an ASIC and FPGA design engineer in EMS technologies, Montreal, QC, Canada. From 2002 to 2004, he was an assistant professor in the Department of Electrical and Computer Engineering, the University of Western Ontario, London, ON, Canada. From 2004, he joined the Department of Electrical and Computer Engineering, Indiana University - Purdue University Indianapolis (IUPUI) as an assistant professor. His main research interests are nanoelectronics, VLSI, DSP, cryptography, digital design, ASIC and FPGA design, and computer arithmetic. He has over 70 journal and conference publications in these areas.