Balanced point operations for side-channel protection of elliptic curve cryptography

L. Batina, N. Mentens, B. Preneel and I. Verbauwhede

Abstract: The authors propose balanced algorithms for elliptic curve cryptography (ECC). The authors make the point addition and doubling balanced; that is, they are implemented as identical sequences of operations. As an example the authors implement an ECC point multiplication algorithm, using the approach of Montgomery, for which a single power trace does not expose the Hamming weight nor the bits of the secret key. Nevertheless, their field-programmable gate array implementation is also compact and efficient. The proposed multiplier for the finite field operations is digit serial and scalable to arbitrary bit-lengths. The method calculates the result by splitting the multiplication into two separate processes. The architecture presented compares favourably with designs presented in the literature. Furthermore, the power consumption graphs show the new implementation has an improved side-channel resistance.

1 Introduction

The best-known and most commonly used public-key cryptosystems (PKCs) are based on factoring (RSA) and on the discrete logarithm problem (Diffie-Hellman, ElGamal, Schnorr, DSA) [1]. They allow secure communications over insecure channels without prior agreement of a shared secret; they also enable efficient and compact digital signatures. Another alternative for PKC is elliptic curve cryptography (ECC), which was proposed in the mid-1980s by Miller [2] and Koblitz [3].

For ECC, two types of finite fields are being considered, i.e. binary and prime fields. A field $\text{GF}(2^n)$ offers far more options as there are many choices for bases, irreducible polynomials, composite fields etc. There exist several ways to accelerate this curve-based arithmetic. Following a bottom-up approach these are speeding up the finite-field arithmetic (especially multiplication and inversion), choosing a ‘good’ representation (i.e. coordinates that are more efficient) and accelerating a scalar multiplication operation.

In this article we propose an algorithm for multiplication in binary fields and we describe an efficient systolic array architecture for the multiplication. The proposed method performs two parts of the multiplication (from LSB and from MSB) in parallel. Furthermore, we consider the approach of Montgomery [4] for scalar multiplication. This uses a representation where computations are performed on the x -coordinate only. According to López and Dahab [5], the Montgomery representation requires less memory and offers better protection against side-channel attacks. The same conclusion with respect to side-channel protection was drawn by Joye and Yen [6]. They also observed its benefit for a parallel computation. Menezes and

Vanstone observed that the benefit in storage is at considerable expense of speed [7]. From an algorithmic point of view, Stam concluded that it is less efficient than other known methods and that in the binary case it can hardly be recommended [8]. However, our conclusion is just the opposite, at least for hardware implementations. This method can benefit from independent calculations for point operations that can therefore be performed fully in parallel by means of two multipliers. Furthermore, we have optimised the formulae for the point operations to have exactly the same number of field multiplications for point addition and doubling. The field multiplications are performed in corresponding steps in both point operations. Similar work has been done by Fischer *et al.* [9] and Izu and Takagi [10] in $\text{GF}(p)$. However, their point operations are not fully balanced as in our case. We are convinced that our approach also offers an improved resistance against side-channel attacks compared with unbalanced methods. We provide some evidence for this in the form of power consumption graphs for what are considered to be ‘side-channel vulnerable’ operations.

The remainder of this paper is organised as follows. Section 2 provides the necessary mathematical background for ECC in $\text{GF}(2^n)$, the method of Montgomery for point multiplication and modular multiplication (MM) in binary fields. In Section 3 previous work is discussed and some relevant hardware implementations of ECC in $\text{GF}(2^n)$ are briefly reviewed. Section 4 gives algorithms and details of the new implementation. In Section 5 the results of our field-programmable gate array (FPGA) implementation of the ECC processor are presented, including a comparison with other relevant work. Section 6 addresses the security with respect to side-channel attacks. Section 7 concludes the paper and points to future work.

2 Elliptic curves over $\text{GF}(2^n)$

ECC relies on a group structure induced on an elliptic curve. A set of points on an elliptic curve (with one special point added, the so-called point at infinity $\mathcal{O}$) together with the so-called chord-and-tangent rule has

© IEE 2005

IEE Proceedings online no. 20055019

doi:10.1049/ip-ifs: 20055019

Paper received 15 July 2005

The authors are with Katholieke Universiteit Leuven, ESAT/SCD-COSIC, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium.

E-mail: Lejla.Batina@esat.kuleuven.ac.be

the structure of an abelian group. Here we consider a finite field of characteristic 2, i.e. $\text{GF}(2^n)$. The point or scalar multiplication is the basic operation for cryptographic protocols; it is easily performed via repeated group operations. At the next (lower) level are the point operations, which are closely related to the coordinates used to represent the points. The lowest level consists of finite field operations such as addition, subtraction, multiplication and inversion required to perform the group operations (Fig. 1).

We introduce some notation. Let $P_4 = (x_4, y_4) = P_2 - P_1$ and $P_5 = (x_5, y_5) = 2P_1$ with $P_3 = P_1 + P_2$. The point P_4 is included because the method for point multiplication, as introduced by Montgomery, is defined by the fact that to add two points their difference should be known (whereas the y -coordinate is not needed). The formulae for point addition and doubling from [11] can be rewritten using that $P_1 = (x_1, y_1) \in E$. For $P_1 \neq P_2$ we get

$$\begin{aligned} x_3 &= \left(\frac{y_1 + y_2}{x_1 + x_2} \right)^2 + \frac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a \\ y_3 &= \left(\frac{y_1 + y_2}{x_1 + x_2} \right) (x_1 + x_3) + x_3 + y_1 \end{aligned} \quad (1)$$

If $P_1 = P_2$,

$$\begin{aligned} x_5 &= x_1^2 + \frac{b}{x_1^2} \\ y_5 &= x_1^2 + \left(x_1 + \frac{y_1}{x_1} \right) x_5 + x_5 \end{aligned}$$

For P_4 we get from Blake *et al.* [11, Lemma III.2]

$$x_4 = \left(\frac{y_1 + y_2 + x_2}{x_1 + x_2} \right)^2 + \frac{y_1 + y_2 + x_2}{x_1 + x_2} + x_1 + x_2 + a$$

We will use the observation that the x -coordinate of P_5 does not include the y -coordinate of P_1 . Also the x -coordinate of the sum of P_3 and P_4 can be expressed with the x -coordinate only. More precisely, we have

Lemma 2.1: [5] *The x -coordinates of the points $P_3 = (x_3, y_3) = P_1 + P_2$ and $P_4 = (x_4, y_4) = P_2 - P_1$ on an elliptic curve (1) satisfy*

$$x_3 + x_4 = \frac{x_1 \cdot x_2}{(x_1 + x_2)^2}$$

Proof: It follows directly from the formula for addition and the curve equation. $\square$

3 Previous work on hardware implementations of ECC

This Section lists some relevant previous work on ECC architectures for binary fields. There are many papers

Fig. 1 Scheme of the hierarchy for ECC operations

[12–16] dealing with this topic but very few efficient hardware implementations present a completely generic solution which allows an arbitrary choice for all parameters: field size, digit-length, irreducible polynomial, elliptic curve parameters, coordinates etc. We opt for a completely generic implementation as security criteria are changing frequently. Nevertheless, we did not need any optimisation to boost performance, which would be possible by fixing some parameters, e.g. special curves, sparse polynomials etc.

As field multiplication is the most crucial aspect for efficient hardware implementations the previous work on finite fields multipliers should be also considered. The first bit-serial multiplier for finite fields was discussed by Beth and Gollmann [17]. This multiplier uses convolution and reduction modulo an irreducible polynomial and takes n clock cycles to compute a multiplication. Relevant algorithms and architectures for multiplication in $\text{GF}(2^n)$ have been proposed in [17–23].

In 2000 Orlando and Paar proposed a scalable elliptic curve processor (ECP) architecture which operates over finite fields $\text{GF}(2^m)$ [13]. The architecture is scalable, with a separated squarer (bit-parallel). Goodman and Chandrakasan proposed a cryptographic processor in [14] which performs a variety of algorithms for PKC applications. Multiplication is performed with a bit-serial multiplier using the Montgomery modular multiplication (MMM) [24]. Gura *et al.* [15] have introduced a programmable hardware accelerator for ECC over $\text{GF}(2^n)$ which can handle arbitrary field sizes up to 255. The multiplier they use is a digit-serial shift-and-add multiplier. For a detailed survey of finite field multipliers and processors for PKC see [25].

4 A new hardware implementation

In this Section we describe our new hardware implementation. We follow the top-down approach and for each step we elaborate our choice.

4.1 Montgomery method for point multiplication in $\text{GF}(2^n)$

For the point multiplication we chose the method of Montgomery that maintains the relationship $P_2 - P_1$ as invariant [4]. The idea of Montgomery dealt with speeding up the calculation of only the x -coordinate of the result. More precisely, to add two points their difference is used as an input parameter while the y -coordinate is not used in the algorithm. This fact is justified because cryptographic applications rarely use the y -coordinate. The algorithm to be used (Algorithm 1)

Algorithm 1: Algorithm for point multiplication

Require: an integer $k > 0$ and a point P

Ensure: $x(kP)$

```

1:  $k \leftarrow k_{l-1}, \dots, k_1, k_0$ 
2:  $P_1 \leftarrow P, P_2 \leftarrow 2P.$ 
3: for  $i$  from  $l-2$  downto 0 do
4:   If  $k_i = 1$  then
5:      $x(P_1) \leftarrow x(P_1 + P_2), x(P_2) \leftarrow x(2P_2)$ 
6:   Else
7:      $x(P_2) \leftarrow x(P_1 + P_2), x(P_1) \leftarrow x(2P_1)$ 
8:   end for
9: Return  $x(P_1)$ 

```

is a variant of the binary method and was considered by López and Dahab. They have also introduced an option for recovering the y -coordinate [5].

The advantage of this algorithm is that it calculates one point addition and one doubling in each step. In this way the loop operations do not depend on the exponent, which could offer an increased resistance against timing and other side-channel attacks. In addition, we noticed that the algorithm requires fewer registers than other hardware solutions. Nevertheless, the performance is not much affected. We discuss this in more details in what follows.

4.2 Point addition and doubling

In this part we move one level lower, i.e. to point operations. This is where our design is improved with respect to other proposals. Namely, point operations (add and double) are in principle different, which can be explored from the viewpoint of side-channel analysis. Some authors have tried already to balance these two operations in order to improve side-channel resistance. Chevallier-Mames *et al.* [26] presented a balanced algorithm for ECC over binary fields in the case of affine coordinates. We mention here the work of Brier and Joye [27], who suggested two approaches to achieve uniformity of point operations. However, both approaches result in some penalty in speed. For the formulae of López and Dahab in $\text{GF}(2^n)$ the operation count is $A : D = 5M : 6M$. Here, A and D are the point operations and M is a field multiplication. We remind the reader that field addition in hardware for $\text{GF}(2^n)$ is just a simple bitwise XOR operation and therefore is not taken into account.

As already mentioned, we deal with projective coordinates to avoid expensive inversions in hardware. Let us consider the formulae for point operations in the case of simple projective coordinates, i.e. $x_i = (X_i/Z_i)$, $i=1, 2$. The results of point doubling and point addition, i.e. $X_5 = X(P_5)$ and $X_3 = X(P_3) = X(P_1 + P_2)$, respectively, are calculated as

$$\begin{aligned} X_5 &= X_1^4 + bZ_1^4 \\ Z_5 &= X_1^2 \cdot Z_1^2 \\ X_3 &= (X_1 \cdot Z_2 + X_2 \cdot Z_1)^2 \\ Z_3 &= x_4 Z_3 + (X_1 \cdot X_2) \cdot (Z_1 \cdot Z_2). \end{aligned} \quad (2)$$

It is easy to see that point doubling and addition would require six and five multiplications, respectively. We slightly rewrote the formulae in order to have six multiplications for both point operations. We had to add one more multiplication in the point addition, so we used the following formula:

$$X_1 Z_2 + X_2 Z_1 = (X_1 + X_2)(Z_1 + Z_2) - X_1 Z_1 - X_2 Z_2$$

which follows from the Karatsuba-like approach. In the case of Karatsuba's algorithm a formula for multiplication reduces the problem of multiplying $2n$ -bit numbers to three multiplications of n -bit numbers. Here we need one extra multiplication so we compute $X_1 Z_2 + X_2 Z_1$ with three multiplications. The next property we want is to have balanced field multiplications in each step of the point operation algorithms (this is not the case for the algorithm of [5]). In this way the two multipliers will work fully in parallel while the exponent is scanned bit by bit. Then for each bit one

addition and one doubling are performed, with fully synchronised multipliers. The algorithms for point addition and doubling are given in Algorithm 2.

Algorithm 2: EC point addition and doubling

Require: X_i, Z_i for $i = 1, \dots, 4$; $x_i = \frac{X_i}{Z_i}$, $x_4 = x(P_1 - P_2)$	Require: $c \in \text{GF}(2^n)$, $c = b^{2^{n-1}}$, X_1, Z_1 where $x_1 = \frac{X_1}{Z_1}$
Ensure: $X(P_1 + P_2) = X(P_3) = X_3, Z_3$	Ensure: $X(2P_1) = X(P_5) = X_5, Z_5$
1. $X_3 \leftarrow X_1 + X_2, Z_3 \leftarrow Z_1 + Z_2$	1. $T_1 \leftarrow c + Z_1, T_1 \leftarrow T_1 + Z_1$
2. $Z_3 \leftarrow X_3 \cdot Z_3$	2. $Z_5 \leftarrow Z_1^2$
3. $T_2 \leftarrow X_1 Z_1$	3. $X_5 \leftarrow X_1^2$
4. $X_3 \leftarrow X_2 Z_2$	4. $T_1 \leftarrow Z_5 T_1$
5. $Z_3 \leftarrow Z_3 + T_2 + X_3$	5. $T_1 \leftarrow T_1 + X_1 + X_1$
6. $Z_3 \leftarrow Z_3^2$	6. $Z_5 \leftarrow X_5 Z_5$
7. $T_1 \leftarrow x_4 Z_3$	7. $T_1 \leftarrow T_1^2$
8. $X_3 \leftarrow T_2 X_3$	8. $X_5 \leftarrow X_5^2$
9. $X_3 \leftarrow X_3 + T_1$	9. $X_5 \leftarrow X_5 + T_1$

Each point operation requires exactly six multiplications which are also balanced with respect to the nine steps in Algorithm 2. In Step 5 of the point doubling a redundant operation is inserted to balance the field additions. The required number of intermediate n -bit registers is three for both cases. More precisely, the following lemma holds:

Lemma 4.1: When Algorithm 1 deploys Algorithm 2 we get the following number of operations in $\text{GF}(2^n)$:

$$\begin{aligned} \#inversions &= 1 \\ \#multiplications &= 12 \lfloor \log_2 k \rfloor + 13 \\ \#additions &= 6 \lfloor \log_2 k \rfloor + 7 \end{aligned}$$

Note that the 12 multiplications require only the time for 6 multiplications since 2 multiplications are performed in parallel in every iteration of the main loop. In this way we improved the formulae of López and Dahab in order to have fully balanced point operations to counterfeit simple side-channel attacks.

4.3 An algorithm for field multiplication

The standard way to compute the product $c(x) = a(x)b(x) \bmod f(x)$ is using convolution, which we refer to as the classical algorithm. Another possible way to calculate the product of two polynomials in $\text{GF}(2^n)$ is Montgomery's multiplication algorithm as proposed in [18]. Here, we define the MMM as $\text{MMM}[a(x), b(x)] := a(x) \cdot b(x) \cdot r^{-1}(x) \bmod f(x)$. Before a sequence of operations can be started, all operands have to be converted to the form $a(x)r(x) \bmod f(x)$, the so-called M -residue of the operand by multiplication with $r(x) = x^n$.

Our circuit implements Algorithm 3. The combined MM algorithm includes two parts, classical and Montgomery, each of which is a systolic array. The parts look quite similar as their cells are performing similar operations, i.e. multiplication and XOR. The difference is that they shift in opposite directions and they start from opposite parts of the loop. While the classical multiplier starts the shift-and-add process from the MSB of one of the operands and shifts the cumulative result left, the Montgomery-based multiplier starts at the LSB and shifts the result right. Those

Algorithm 3: Digit-serial MM in GF(2ⁿ)

Require: polynomials $a(x)$, $b(x)$ and $f(x)$, $F_0'(x)$

Ensure: $Res(x) = a(x) \cdot b(x) \cdot x^{-\lceil \frac{s}{2} \rceil} \bmod f(x)$

```

1:  $Res(x) = 0$ ,  $Res_C(x) = 0$ ,  $Res_M(x) = 0$ ,
2: for  $i$  from 0 to  $\lceil \frac{s}{2} \rceil - 1$  do
3:  $Res_C(x) \leftarrow Res_{C_{i-1}} \cdot f(x) + Res_C(x) \cdot x + A_{s-i} \cdot b(x)$ 
4:  $Res_M(x) \leftarrow Res_M(x) + A_i \cdot b(x)$ 
5:  $M(x) \leftarrow Res_{M_0} \cdot F_0'(x) \pmod{x^w}$ 
6:  $Res_M(x) \leftarrow Res_M(x) + M(x) \cdot f(x)$ 
7:  $Res_M(x) \leftarrow Res_M(x) \text{div } x^w$ 
8: end for
9:  $Res(x) = Res_C(x) + Res_M(x)$ 
10: Return  $Res(x)$ 

```

two arrays process the operand $a(x)$ from different sides and they stop after exactly $\lceil s/2 \rceil$ cycles (here s is the number of digits). The classical part still has to perform a shift over $\lceil s/2 \rceil$ bits, but this is taken care of by the conversion of the M -residue of the result. More precisely, the M -residue is of the form $a(x)b(x)r^{-1}(x) \bmod f(x)$ where $r(x) = x^{\lceil s/2 \rceil}$. The multiplication is completed by calculating MMM [$Res(x)$, 1].

The idea of combining two algorithms together was mentioned for the first time in [28] and the schematic of the multiplier was presented in [29].

A schematic of the multiplier is presented in Fig. 2. A_i , B_i and F_i are the coefficients of $a(x)$, $b(x)$ and $f(x)$, respectively. The outputs $c(x)$ become inputs to the systolic arrays in the next clock cycle. Finally, the result of the multiplication is obtained by XOR-ing the outputs of both systolic arrays.

In Figure 3, one processing element (PE) of the array is depicted. There is the so-called regular PE, while the boundary PEs are called leftmost and rightmost PE. They have a slightly different structure which is shown in Fig. 3b (leftmost PE) and 3c (rightmost PE).

After conversion from the M -residue to the normal representation, we get the result. Namely, the following lemma holds.

Lemma 4.2: The result of Algorithm 3 is in the Montgomery domain, i.e. $Res(x) = a(x) \cdot b(x)r^{-1}(x)$ where $r(x) = x^{\lceil \frac{s}{2} \rceil}$.

Proof: Here $n = sw$, i.e. the bit representation can be written in s words of length w . After $\lceil s/2 \rceil$ steps in

Algorithm 3, the left (Res_C) part calculated has been shifted to the right for $\lceil s/2 \rceil \cdot w$ bits. At the same time, the right (Res_M) part has been shifted to the left for the same number of bits due to the division with x^w . This results in a partial result of the form $Res_M = a(x)b(x)x^{-\lceil s/2 \rceil \cdot w}$. The Res_C part still has to be shifted over the remaining $\lceil \frac{s}{2} \rceil$ bits, which is adjusted due to the remaining conversion from Montgomery to normal representation. $\square$

In Algorithm 3 $A_i(x)$ represents one digit of the polynomial $a(x)$. The addition step is performed by multiplication of $A_i(x)$ with $b(x)$. In order to be able to perform Step 7 in Algorithm 3 a multiple of $f(x)$ has to be added which results in $Res_M(x)$ being divisible by x^w . So, for $Res_M(x) \neq 0 \bmod x^w$, we need to find $M(x)$ such that $Res_M(x) + M(x)f(x) = 0 \bmod x^w$. $M(x)$ is calculated from the following equation: $M(x) = Res_{M_0} \cdot F_0^{-1}(x) \bmod x^w$. It is easy to see that $F_0^{-1}(x) \bmod x^w \equiv F_0'(x)$, which follows from the relations for Montgomery's parameter $r(x)$ [18]. Also, here $Res_{M_0}(x)$ and $F_0(x)$ are the least significant words of $Res_M(x)$ and $f(x)$, respectively.

4.4 A prototype FPGA architecture

Our ECP is shown in Fig. 4. The operation blocks on each level from top to bottom are as follows:

- Level 1: Main controller
- Level 2:
 - (1) Normal to Montgomery representation conversion (NtoM)
 - (2) Affine to projective coordinates conversion (AtoP)
 - (3) EC point multiplication (PM)
 - (4) Projective to affine coordinates conversion (PtoA)
 - (5) Montgomery to normal representation conversion (MtoN)
- Level 3: EC point doubling (PD) and EC point addition (PA)
- Level 4: Modular addition (MA), MM and modular inversion (MI)

The main control finite state machine (FSM) first commands the NtoM to convert all the inputs from normal to Montgomery representation and the AtoP to convert the coordinates from affine to projective. The NtoM and the AtoP both use the MM to perform these conversions. The AtoP also needs the MA to do some precalculations, see (2). When all conversions are

Fig. 2 Schematic of the multiplier. The classical and the Montgomery parts are calculated in parallel and the result is XORed afterwards

Fig. 3 The regular PE of the systolic array, the leftmost PE of the systolic array, the rightmost PE of the systolic array

finished the main controller orchestrates the PM block to start the point multiplication by invoking the PD and the PA in parallel. It writes the resulting $X1$ and $Z1$ to its output registers. The FSM inside the PM orchestrates these operations. Due to the parallelism of the point operations both the PD and the PA use an MA and an MM. The next step after point multiplication is conversion from projective to affine representation using the MI, which invokes the MM to perform the MI using Fermat's theorem. Finally, the affine coordinates are converted from Montgomery to normal representation using the MM.

The flowchart of the FSM inside the point multiplication block is shown in Fig. 5. When the START signal is set, the bits of k are evaluated from MSB to LSB resulting in the assignment of new values for P1 and P2. These values depend on the key-bit k_i . When all bits have been evaluated, an internal counter gives an END signal. The result of the last P1 calculation is written to the output register and the VALID output is set.

5 Results

The results of our design on a Xilinx Virtex XCV800 FPGA are given in Table 1.

The formula for the latency that is used in Table 1 is: $\text{latency} = [21s] + [6s(n-2) + 3s] + [18sn + 4n]$, where the three parts of the formula correspond to the calculations of the main conversion operations, the MI and the point multiplication, respectively.

One of the consequences of the scalability of the design is that the minimal clock period does not depend on the bit-length n ; it depends only on the digit-length w . This can be observed in Table 1.

Table 2 presents a broader comparison with other architectures. We found that comparing designs is hard since these designs have been optimised for different goals, have been implemented on different platforms and have chosen different options for bases, coordinates, irreducible polynomials etc. Moreover, some of these solutions are not scalable: in these designs, some parameters are fixed (such as a special polynomial,

Fig. 4 Architecture of the ECP

Fig. 5 Flowchart of the FSM inside the ECPM

special reduction etc.), which boosts the performance. Therefore, we have included only the solutions that are either scalable [14, 15] or believed to be the state of the art in ECC hardware implementations. The purpose of this table is mainly to reference the prior methodology in this area. We give this comparison as a proof that a scalable and side-channel-secure design can also lead to a solution that is competitive in performance.

6 Side-channel security

Implementations of cryptographic algorithms should be resistant to side-channel attacks such as timing [30],

power [31] and electromagnetic radiation [32, 33] analysis attacks. These attacks present a realistic threat for wireless applications and have been demonstrated to be very effective against smart cards without specific countermeasures. The latency of the proposed algorithm does not depend on the Hamming weight of the exponent, which makes it very suitable for defending against timing attacks. Indeed, timing attacks can explore all steps with non-constant execution time; conditional instructions are a typical target. Also in that case power traces may reveal some secret information that would allow the attacker to perform simple power analysis, i.e. an SPA attack. For example, a typical

double-and-add algorithm [1], which executes the point doubling and addition operations if the i -th bit of the exponent (k_i) = 1 and otherwise (for k_i = 0) performs only doubling, is not SPA resistant.

Also, the computational difference between the point operations is a typical target of an attacker [27, 34]. To prevent that, cryptographic algorithms should be implemented as sequences of operations that are indistinguishable through simple side-channel analysis. Chevallier-Mames *et al.* define this property as the side-channel atomicity [26]. In their view, SPA-resistant algorithms should consist of so-called side-channel

Table 1: Implementation results for bit-lengths of $n=179$ and $n=211$ and comparison of bit-serial and digit-serial multipliers

n	Results for ECC point multiplication					
	No. slices		Period (ns)		Latency (ms)	
	179	211	179	211	179	211
$w=1$	10626	13863	19.165	19.159	2.479	3.438
$w=4$	11433	14660	19.298	19.301	1.886	2.614
$w=8$	11622	14950	20.050	20.043	1.008	1.391
$w=16$	11881	15367	20.961	20.971	0.557	0.763

Table 2: Comparison of different architectures for multiplication in $GF(2^n)$

Impl.	Fin. field	FPGA	Frequency (MHz)	Latency (ms)	Hardware compl.
[14]	$GF(2^{160})$	ASIC	50.0	Est. 5	n. a.
[13]	$GF(2^{167})$	XCV400E	76.7	Est. 0.84	6,513 Gates, 501 reg.
[15] ^a	$GF(2^{163})$	XCV2000E	66.4	Est. 0.586	14241 LUTs, 2990 FFs
[16]	$GF(2^{163})$	XCV1000E	22.1	7.4	Not known
[*] ^b	$GF(2^{179})$	XCV800	47.7	0.991	7,788 Slices

^aThe performance of [15] is for so-called known curves because of a special reduction. For generic curves the performance is much slower.

^bThis work.

atomic blocks which are algorithm specific. In our implementation, there exist side-channel atomic blocks on different levels of the ECC hierarchy. Following the top-down approach, these are point addition/doubling and multiplication/squaring. For that purpose we use the same multiplier for both multiplication and squaring, although there exist more efficient architectures for squaring. Here we propose a fully balanced point multiplication that performs the same operations for every loop of the algorithm. More precisely, Algorithm 2 for point operations executes exactly one field multiplication in corresponding steps. It is not the case in general, as doubling and addition are two different operations, as in most of the standard references on ECC [11, 35]. Considering our new Algorithm 2, point operations result in two not so distinguishable patterns (Figs 6 and 7). In both Figures a total of six field multiplications can be observed. Furthermore, both operations (double and add) are performed in parallel for each step in Algorithm 1.

The systolic array architecture also allows more parallelism of the multiplication process, which will make power analysis more difficult. These first results, although proving SPA and timing resistance, are just the first step towards a side-channel-resistant design. In future work this algorithm should be more carefully examined with respect to security against side-channel attacks; however, we are convinced that our design approach offers an increased resistance to these attacks. Moreover, for more advanced attacks such as (first- or higher-order) differential power analysis and differential electromagnetic analysis, other countermeasures are required as well.

7 Conclusions and future work

A complete ECC processor for binary fields has been presented. An FPGA implementation has been fully described including a bit/digit-serial multiplier that combines two previously known multiplication methods. The proposed architecture is a systolic array that allows for good performance in speed and more parallelism in operation, which is also beneficial for side-channel security. Furthermore, we have proposed a fully balanced point multiplication algorithm that performs

Fig. 6 Power consumption trace of a PA as in Algorithm 2 (six field multiplications are easily detected)

Fig. 7 Power consumption trace of a PD as in Algorithm 2 (also includes six multiplications)

the same operations for every loop of the algorithm. By using this approach we proved that this so-called side-channel-aware design is the first step towards side-channel resistance. However, it is clear that additional countermeasures might be required.

8 Acknowledgements

Lejla Batina and Nele Mentens are funded by research grants of the Katholieke Universiteit Leuven, Belgium. This work was supported by FWO projects (G.0450.04 and G.0141.03) and by the EU IST FP6 projects SCARD and ECRYPT.

9 References

- Menezes, A., van Oorschot, P., and Vanstone, S.: "Handbook of Applied Cryptography" (CRC Press, 1997)
- Miller, V.: "Uses of elliptic curves in cryptography". In Williams, H.C. (Ed.), Proc. Advances in Cryptology: CRYPTO'85, Santa Barbara, CA, August, 1985, *LNCS* **218** (Springer-Verlag, 1985), pp. 417–426
- Koblitz, N.: "Elliptic curve cryptosystem". *Math. Comp.*, 1987, **48**, pp. 203–209
- Montgomery, P.: "Speeding the pollard and elliptic curve methods of factorization". *Math. Comput.*, 1987, **48**, pp. 243–264
- López, J., and Dahab, R.: "Fast multiplication on elliptic curves over $GF(2^m)$ ". In Koç, Ç.K., and Paar, C. (Eds), Proc. of 1st Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES), Worcester, MA, USA, August, 1999, *LNCS* **1717** (Springer-Verlag, 1999), pp. 316–327
- Joye, M., and Yen, S.-M.: "The montgomery powering ladder". In Kaliski, B.S., Jr, Koç, Ç.K., and Paar, C. (Eds), Proc. 4th Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES), San Francisco Bay, USA, August, 2002, *LNCS* **2523** (Springer-Verlag, 2002), pp. 291–302
- Menezes, A., and Vanstone, S.: "Elliptic curve cryptosystems and their implementations". *J. Cryptol.*, 1993, **6**, pp. 209–224
- Stam, M.: "Speeding up Subgroup Cryptosystems". (PhD thesis, Technische Universiteit Eindhoven)
- Fischer, W., Giraud, C., Knudsen, E.W., and Seifert, J.-P.: "Parallel scalar multiplication on general elliptic curves over $\mathbb{F}_p$ hedged against non-differential side-channel attacks", IACR ePrint archive: 2002/007 Report 2002/007, 2002
- Izu, T., and Takagi, T.: "A fast parallel elliptic curve multiplication resistant against side channel attacks". In Naccache, D., and Paillier, P. (Eds), Proc. Int. Workshop on Practice and Theory in Public Key Cryptosystems (PKC 2002) *LNCS* **3027**, Springer-Verlag, 2002 pp. 280–296
- Blake, I., Seroussi, G., and Smart, N.P.: "Elliptic curves in cryptography". London Mathematical Society Lecture Note Series (Cambridge University Press, 1999)
- Sutikno, S., Effendi, R., and Surya, A.: "Design and implementation of arithmetic processor $GF(2^{155})$ for elliptic curve cryptosystems". Proc. 1998 IEEE Asia-Pacific Conf. on Circuits and Systems (APCCAS'98), Chiangmai, Thailand, November 1998, pp. 647–650
- Orlando, G., and Paar, C.: "A high-performance reconfigurable elliptic curve processor for $GF(2^m)$ ". In Koç, Ç.K., and Paar, C. (Eds), Proc. 2nd Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES), Worcester, MA, June 2000, *LNCS* **1965** (Springer-Verlag, 2000), pp. 41–56
- Goodman, J., and Chandrakasan, A.P.: "An energy-efficient reconfigurable public-key cryptography processor". *IEEE J. Solid-St. Circ.*, 2001, **36** (11), pp. 1808–1820
- Gura, N., Shantz, S.C., Eberle, H., Finchelstein, D., Gupta, S., Gupta, V., and Stebila, D.: "An end-to-end systems approach to elliptic curve cryptography". In Kaliski, B., Jr, Koç, Ç.K., and Paar, C. (Eds), Proc. 4th Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES), San Francisco Bay, USA, August 2002, **2523**
- Kitsos, P., Theodoridis, G., and Koufopavlou, O.: "An efficient reconfigurable multiplier architecture for Galois Field $GF(2^m)$ ". *Microelectr. J.*, 2003, **34**, pp. 975–980
- Beth, T., and Gollmann, D.: "Algorithm engineering for public key algorithm". *IEEE J. Sel. Area. Comm.*, 1989, **7** (4), pp. 458–465
- Koç, Ç.K., and Acar, T.: "Montgomery multiplication in $GF(2^k)$ ". *Design. Code. Cryptogr.*, **14**, pp. 57–69
- Wei, S.-W.: "VLSI architectures for computing exponentiations, multiplicative inverses, and divisions in $GF(2^m)$ ". *IEEE Trans. Circuits II*, 1997, **44** (10), pp. 847–855
- Tsai, W.-C., and Wang, S.-J.: "A systolic architecture for elliptic curve cryptosystems". Proc. ICSP2000, Beijing, China, August 2000, pp. 591–597
- Song, L., and Parhi, K.K.: "Low-energy digit-serial/parallel finite field multipliers". *J. VLSI Signal Proc.*, 1998, **19**, pp. 149–166
- Mentens, N., Örs, S.B., Preneel, B., and Vandewalle, J.: "An FPGA implementation of an elliptic curve processor over $GF(2^m)$ ". Proc. GLSVLSI 2004, Boston, MA, April 2004, 4 pp
- Wu, H.: "Montgomery multiplier and squarer for a class of finite fields". *IEEE Trans. Comput.*, 2002, **51** (5), pp. 521–529
- Montgomery, P.: "Modular multiplication without trial division". *Math. Comput.*, 1985, **44**, pp. 519–521
- Batina, L., Örs, S.B., Preneel, B., and Vandewalle, J.: "Hardware architectures for public key cryptography". *Integration*, 2003, **34**, (1–2), pp. 1–64
- Chevallier-Mames, B., Ciet, M., and Joye, M.: "Low-cost solutions for preventing Simple Side-Channel Analysis: Side-Channel Atomicity". *IEEE Trans. Comput.*, 2004, **53** (6), pp. 760–768
- Brier, E., and Joye, M.: "Weierstrass elliptic curves and side-channel attacks" In Naccache, D., and Paillier, P. (Eds), Proc. PKC'02, *LNCS* **2274**, (Springer-Verlag, 2002) pp. 335–345
- Potgieter, M.J.: "A hardware implementation of the group operations necessary for implementing an elliptic curve cryptosystem over a characteristic two finite field". Final report of project EPR400. (Technical University Eindhoven, 2002)
- Batina, L., Mentens, N., Örs, S.B., and Preneel, B.: "Serial multiplier architectures over $GF(2^n)$ for elliptic curve cryptosystems".

- Proc. 12th IEEE MELECON 2004, Dubrovnik, Croatia, May, 2004, 4 pp
- 30 Kocher, P.: "Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems". In Koblitz, N., (Ed.), Proc. Advances in Cryptology: CRYPTO'96, Santa Barbara, August, 1996, *LNCs* **1109** (Springer-Verlag, 1996), pp. 104–113
 - 31 Kocher, P., Jaffe, J., Jun, B., "Differential power analysis" In Wiener, M. (Ed.), Proc. Advances in Cryptology: CRYPTO'99, Santa Barbara, CA, August 1999, *LNCs* **1666** (Springer-Verlag, 1999) pp. 388–397
 - 32 Quisquater, J.-J., and Samyde, D.: "Electromagnetic analysis (ema): measures and counter-measures for smart cards". In Attali, I., and Jensen, T.P. (Eds), Proc. Smart Card Programming and Security (E-smart 2001), Cannes, French Riviera, September, 2001, *LNCs* **2140** (Springer-Verlag, 2001), pp. 200–210
 - 33 Gandolfi, K., Mourtel, C., and Olivier, F.: "Electromagnetic analysis: concrete results". In Koç, Ç.K., Naccache, D., and Paar, C. (Eds), Proc. 3rd Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES), Paris, France, May, 2001, *LNCs* **2162** (Springer-Verlag, 2001), pp. 255–265
 - 34 Trichina, E., and Bellezza, A.: "Implementation of elliptic curve cryptography with built-in counter measures against side channel attacks". In Kaliski, B.S., Jr, Koç, Ç.K., and Paar, C. (Eds), Proc. 4th Int Workshop on Cryptographic Hardware and Embedded Systems (CHES), San Francisco Bay, USA, *LNCs* **2535**, (Springer-Verlag, 2002), pp. 98–113
 - 35 IEEE P1363: 'Standard specifications for public key cryptography', 1999