Available online at www.sciencedirect.com
ScienceDirect
Journal of Systems Architecture 53 (2007) 72–84
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

High-speed hardware implementations of Elliptic Curve Cryptography: A survey

Guerric Meurice de Dormale *,[1], Jean-Jacques Quisquater

UCL Crypto Group, Laboratoire de Microélectronique, Université catholique de Louvain, Place du Levant 3, B-1348 Louvain-la-Neuve, Belgium

Received 2 May 2006; received in revised form 31 August 2006; accepted 15 September 2006
Available online 2 November 2006

Abstract

For the last decade, Elliptic Curve Cryptography (ECC) has gained increasing acceptance in the industry and the academic community and has been the subject of several standards. This interest is mainly due to the high level of security with relatively small keys provided by ECC. To sustain the high throughput required by applications like network servers, high-speed implementations of public-key cryptosystems are needed. For that purpose, hardware-based accelerators are often the only solution reaching an acceptable performance-cost ratio. The fundamental question that arises is how to choose the appropriate efficiency–flexibility tradeoff.

In this survey, techniques for implementing Elliptic Curve Cryptography at a high-speed are explored. A classification of the work available in the open literature in function of the level of efficiency and flexibility is also proposed. In particular, the subjects of reconfigurable, dedicated, generator, versatile and general purpose scalar multipliers are addressed. Finally, some words about future work that should be tackled are provided.

Keywords: Public-key cryptography; Elliptic Curve Cryptography; High-speed hardware implementation; Efficiency–flexibility tradeoffs; Network applications

1. Introduction

Since their introduction in cryptography in 1985 by Koblitz [43] and Miller [51], elliptic curves have raised increasing interest. This rich mathematical tool, already known by number theorists for a long time, was used to set up new asymmetric schemes able to compete with the well established RSA. Such schemes allow many useful functionalities like digital signature, public-key encryption, key agreements,... For those needs, Elliptic Curve Cryptography (ECC) is indeed an attractive solution as the provided public-key scheme is currently one of the most secure per bit. As a result, highly desired properties are exhibited such as less processing power, storage, bandwidth and power consumption. Nowadays, this technology is well accepted in the industry and the academic community and has been the

* Corresponding author.
E-mail addresses: gmeurice@dice.ucl.ac.be (G. Meurice de Dormale), quisquater@dice.ucl.ac.be (J.-J. Quisquater).
[1] Supported by the Belgian fund for industrial and agricultural research (FRIA).

1383-7621/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2006.09.002

subject of several standards [3,36]. The customers list of the Certicom company gives a good overview of today's utilization of ECC in the industry. It includes the National Security Agency for government communications, Research in Motion for enterprise software, Unisys for banking applications and Alcatel for mobile security. It is also worth to mention that Microsoft uses ECC for its Digital Right Management.

In network applications like SSL and IPsec protocols, high-speed implementation of public-key cryptography can be required. For that purpose, hardware-based accelerators are either the only solution reaching an acceptable performance-cost ratio or even the only feasible solution. One of the main subjects of that framework deals with hardware accelerator for network servers where millions of heterogeneous client devices need to connect. Therefore, the fundamental question arises about the choice of an appropriate tradeoff between the *efficiency* (e.g. throughput, latency,...) and the *flexibility* (e.g. independency of parameters, data length, field type,...).

This survey presents the current research in the high-speed hardware implementation of Elliptic Curve Cryptography. In particular, a classification of the work available in the open literature according to the level of efficiency and flexibility is proposed. Hyper Elliptic Curve Cryptography and the pairings architectures are not covered and could be the topic of another article.

As for many high-speed applications the hardware is used together with a software-based controller, or Application Programming Interface (API), the hardware processor usually performs only the most computer-intensive operation: the scalar point multiplication. It is the core operation of many elliptic curve based signature, key agreement and encryption schemes including ECDSA, EC-KCDSA, ECIES, PSEC, ECMQV,... Therefore, this article is almost exclusively dedicated to scalar point multiplication. It is also important to notice that the problem of side channel attacks (cf. [71]) is not within the scope of this study. Indeed, scenarios involving high speed hardware usually do not suppose that *bad guys* have physically access to the device. Remotely accessible information like timing could be a problem (e.g. [14]) but it can be easily overcome using parallel hardware architecture.

This survey intends to serve as an introduction to the ECC topic in a high-speed context, bringing interested readers quickly up to speed on developments from the last half-decade. The aim is to help them choosing an appropriate architecture in function of the desired level of efficiency and flexibility. For further information, older implementations can be found in [9] and an analysis about efficiency-security tradeoff in the context of general ECC engineering is available in [17].

The paper's structure is divided as follows: Section 2 recalls the mathematical background of elliptic curves. Then, Section 3 explains the different design principles, including coordinates, scalar multiplication algorithms and scheduling as well as finite field arithmetic. An overview of the current research works stands in Section 4. Specifically, reconfigurable, dedicated, generator, versatile and general purpose scalar multipliers are addressed. Representative hardware implementations of each of those categories is also exhibited. Finally, the conclusion and a few words about further works are given in Section 5.

2. Mathematical background

The underlying hard problem of ECC is the intractability of the elliptic curve discrete logarithm problem (ECDLP). As no sub-exponential algorithms are known to solve a generic instance of this problem, the key sizes are allowed to be much smaller than other public-key cryptosystems like RSA.

As this section presents highly standard material and due to space limitation, only a quick review about the background of elliptic curve is provided. For a thorough description of the topic, the reader is referred to the literature [13].

2.1. Elliptic curves

The Weierstraß equation of an elliptic curve E defined over $\mathbb{K}$ using affine coordinates is

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

where $a_1, a_2, a_3, a_4, a_5, a_6 \in \bar{\mathbb{K}}$. An elliptic curve E is then defined as the set of solutions of this equation in the affine plane, together with the point at infinity $\mathcal{O}$. Using several *admissible* changes of variables, simplified (or reduced) versions of this equation are used in practice.

For a non-singular curve equation defined over a field GF(p) and of characteristic $\neq 2, 3$, the reduced Weierstraß equation has the form

$$E : y^2 = x^3 + ax + b, \tag{1}$$

where $a,b \in \mathrm{GF}(p)$ and $4a^3 + 27$, $b^2 \neq 0$.

For a non-supersingular curve equation of characteristic 2 (defined over a field $\mathrm{GF}(2^m)$), the reduced Weierstraß equation has the form

$$E : y^2 + xy = x^3 + ax^2 + b, \tag{2}$$

where $a, b \in \mathrm{GF}(2^m)$, $b \neq 0$.

To ensure security of ECC, field sizes of at least 160-bit and prime extension for $\mathrm{GF}(2^m)$ are recommended [3,36]. Indeed, composite m's for $\mathrm{GF}(2^m)$ exhibit some weaknesses [29,49].

2.2. Group law

While using elliptic curve, the fundamental operations are the point addition and doubling. They are the building blocks of the scalar multiplication algorithms. Let $P = (x_1, y_1)$ be a point of the curve E defined either by Eqs. (1) or (2). Then, the negative of point P is $-P = (x_1, -y_1)$ for Eq. (1) and $-P = (x_1, x_1 + y_1)$ for Eq. (2).

The sum $P + Q$ of points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ (assuming that P, $Q \neq \mathcal{O}$) is point $R = (x_3, y_3)$ where:

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{cases} \text{ with}$$

$$\lambda = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & \text{when} \quad P \neq \pm Q \\ (3x_1^2 + a)/2y_1 & \text{when} \quad P = Q \end{cases}$$

for Eq. (1) and for Eq. (2):

$$\begin{cases} x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \end{cases} \text{ with}$$

$$\lambda = \begin{cases} (y_2 + y_1)/(x_2 + x_1) & \text{when} \quad P \neq \pm Q \\ x_1 + y_1/x_1 & \text{when} \quad P = Q \end{cases}$$

2.3. Scalar multiplication algorithms

The scalar multiplication $Q = P + P + \cdots + P = kP$ is basically performed by a binary algorithm (Algorithm 1), also called *double and add* method [33]. It is the analogous of the well-known *square and multiply* method. For a random k, the expected number of point additions is $l/2$, with $l = \lceil \log_2(k) \rceil$.

Algorithm 1. Left-to-right binary scalar multiplication

Input: $P \in E$, $k = (1, k_{l-2}, \ldots, k_1, k_0)_2$
Output: $Q = k \cdot P$
  $Q \leftarrow P$
  for i from $l - 2$ downto 0 **do**
    $Q \leftarrow 2Q$
    if $k_i = 1$ **then** $Q \leftarrow Q + P$

As the negative of a point can be easily computed, another possibility is to use an addition–subtraction method [33] (Algorithm 2). A particularly useful signed digit representation for k is thenon-adjacent form (NAF). For a random k, this algorithm decreases the expected number of point additions to $l/3$.

Algorithm 2. Left-to-right binary NAF scalar multiplication

Input: $P \in E$, $k = (k_l, k_{l-1}, \ldots, k_0)_2$, $k_i \in \{-1, 0, 1\}$
Output: $Q = k \cdot P$
  $Q \leftarrow 0$
  for i from l downto 0 **do**
    $Q \leftarrow 2Q$
    if $k_i = 1$ **then** $Q \leftarrow Q + P$
    if $k_i = -1$ **then** $Q \leftarrow Q - P$

Another popular choice is the Montgomery ladder algorithm [53] (Algorithm 3). It is based on the following observation: the x-coordinate of the sum of two points, whose difference is known, can be computed in terms of the x-coordinates of the involved points. As a result, the computation of the y-coordinate is not needed and resources are saved. A way to recover this coordinate at the end of the scalar multiplication was later introduced by [58]. Based on that result, López and Dahab [45] proposed a version for $\mathrm{GF}(2^m)$ with a way to recover the y-coordinate.

Algorithm 3. Montgomery ladder scalar multiplication

Input: $P \in E$, $k = (1, k_{l-2}, \ldots, k_1, k_0)_2$
Output: $x(P_1) = x(k \cdot P)$
  $P_1 \leftarrow P$, $P_2 \leftarrow 2P$
  for i from $l - 2$ downto 0 **do**
    if $k_i = 1$ **then** $x(P_1) \leftarrow x(P_1 + P_2)$, $x(P_2) \leftarrow x(2P_2)$
    else $x(P_2) \leftarrow x(P_1 + P_2), x(P_1) \leftarrow x(2P_1)$

A survey about methods for the scalar multiplication can be found in [30].

2.4. Choice of coordinates

To avoid the costly inversion needed by the computation of λ (cf. Section 2.2), other coordinate systems can be used. For example, the use of homogeneous projective coordinates maps the representation of the point P from (x, y) to (X, Y, Z) with $x = X/Z$ and $y = Y/Z$. More multiplications and squarings are required to perform the addition and doubling of points, but the inversion is deferred until the end of the whole computation.

This final conversion back to affine coordinates is required for inter-operability reasons and as elliptic curve-based schemes work with this kind of coordinates. Without randomization, this conversion is also needed for security reasons [54].

3. Design principles

High-speed is the major optimization goal for the applications covered by this article. The required resources and the power consumption are therefore of minor importance. Nevertheless, the designer must be sure that integration of for instance large multipliers, results in a significant speedup. In the most common applications, a low latency is preferred to a high throughput. The main reason is that clients expect a fast response to their queries and do not care about methods used by the server to cope with traffic throughput. To compare with secret key cryptography, this context is equivalent to an AES implementation using a chaining mode and with a single dataset at a time. This assumption is non negligible for the different implementation choices and affects the performances of systems.

Most implementations deal only with scalar multiplication. Other operations necessary for the elliptic curve-based protocols are left to the software controller. As a result, this paper treats only the scalar multiplication operation. However, it could be useful to spend a little piece of hardware to support complex operations in software like GF(p) inversion for ECDSA.

Arithmetic in GF(2^m) offers some implementation advantages on hardware platforms. Indeed, the addition is only a bitwise XOR and there is no carry propagation. The squaring is also a simple linear operation. Consequently, almost all the papers dealing with high-speed hardware implementations use this kind of field. Implementations based on GF(2^m) will therefore be emphasized in this article.

While designing high-speed ECC architecture, one of the main concerns is the choice of appropriate coordinates related with the scalar multiplication algorithm. When very high speed is required, another important subject is the scheduling of scalar multiplication operations. A third important choice is the kind of multiplier used with curves over GF(2^m). Finally, an appropriate inversion algorithm for final conversion to the affine coordinates has to be chosen.

3.1. Coordinates and scalar multiplication algorithms

The choice of system coordinates together with a scalar multiplication algorithm is the first problem to deal with. It depends on applications but most of the time, the base point P is not considered to be known in advance. First of all, affine coordinates should not be used since the complexity of a high-speed inverter is much higher than a fast multiplier. Therefore, projective or mixed projective coordinates are usually employed.

Without precomputation, the fastest known algorithms for scalar multiplication are the Montgomery Ladder version of López and Dahab (LD) for GF(2^m) and the binary NAF method with mixed coordinates for GF(p). The relevant amount of operations required by the GF(2^m) algorithm are $6lM + 5lS + 2S + (I + M)$, where M stands for a modular Multiplication, S for a modular Squaring, I for a modular Inversion and l for the bit-length of the key. The cost of S will be discussed later. For GF(p), the approximate running time using Jacobian mixed coordinates is $\frac{l}{3}(8M + 3S) + l(4M + 4S) + (I + S + M)$.

Though fixed base point is usually not assumed, a speed-up for curves based on GF(p) can be achieved by on the fly precomputation. The reader is referred to [30,42,33] for details about precomputation and fixed point scalar multiplication.

3.2. Scalar multiplication scheduling

For very high-speed implementations, while large and fast multipliers are used, the time needed to perform additions and squarings could not be negligible at all. As a result, data dependency can be an issue and the scheduling of scalar multiplication operations must be carefully handled. In particular, the designer must ensure that the multipliers are (almost) never left idle.

For large multipliers, interesting operating frequency is only achieved if some pipelining stages

are added. Hence, in order to keep the pipeline always filled, the output of a multiplier should not be directly used as the input of the same multiplier. A careful analysis of the scheduling of the LD algorithm can be found in [4].

3.3. Multiplier and squarer

As the architectures are usually built around high speed multiplier(s), the choice of the kind of multiplier has a great impact on the whole structure and the performances. Unlike GF(p), there are radically different ways to perform a modular multiplication in GF(2^m). In the open literature, most of the designs use either an Optimal Normal Basis (ONB) multiplier [1] or a Polynomial Basis (PB) multiplier [74,46].

With the use of ONB, the squaring operation is only a bit rotate shift. Nevertheless, multiplication is more complex than in PB. While a squaring in PB is not as cheap as in ONB, the use of irreducible trinomials and pentanomials can drastically reduce the complexity. These kinds of polynomials are recommended by the NIST and SECG [56,64] and lead to efficient bit-parallel modular reduction. For example, a modular squaring with a trinomial requires at most $\lfloor \frac{m+k-1}{2} \rfloor$ bit additions [79].[2] With respect to large multipliers, the area of a dedicated bit-parallel squarer is negligible. This operation can therefore be performed in one clock cycle, as does the addition.

As there is three times less type-II ONB fields (with a prime extension) of practical sizes (160 to 256-bit) and because the complexity of a practical PB squaring is low, PB could be preferred. An interesting discussion on this topic can be found in [78,80].

It should be added that, as prime extensions are recommended, type-I ONB should not be used. As the complexity of a type-I ONB is smaller than a type-II ONB, comparisons between architectures using type-I ONB and PB are biased.

3.4. Final inversion

While projective coordinates are used, a conversion to affine coordinates is required at the end of the whole computation. A dedicated circuit based on the extended Euclidean algorithm could be employed [21,20]. Nevertheless, methods based on little Fermat's theorem are currently preferred as the required arithmetic units for the modular exponentiation are already available.[3] For GF(2^m), the number of multiplications can be reduced thanks to the multiplication chain technique of Itoh and Tsujii [38]. The required number of multiplications needed becomes $\lfloor \log_2(m-1) \rfloor + W(m-1) - 1$, where $W()$ stands for the hamming weight function. Especially if a dedicated squaring unit is available, a fast modular inversion can be achieved with this method.

4. Related works

During the last half-decade, many papers about high-speed ECC have been published in the open literature. The aim of this section is to list the main kind of implementations related to this survey and to give an overview of the most relevant architectures. As explained in the introduction, early implementations can be found in [9]. In particular, the following categories for the scalar multipliers highlight the different level of efficiency and flexibility:

Reconfigurable and dedicated ASIC deals with architecture using only one field size and one hardwired irreducible polynomial (or modulo). This polynomial is of course mutable through reconfiguration of reconfigurable platforms.

Generator concerns architectures automatically synthesizable with scripts, needing only the parameters of the system.

Versatile handles architectures accommodating various field sizes and irreducible polynomials (or modulo) without reconfiguration.

General purpose treats generic architectures with an arithmetic unit independent on the field size and the irreducible polynomial (or modulo).

Multi-scalar deals with computation of different scalar multiplications at the same time.

Dedicated ASIC and reconfigurable scalar multiplier are assimilated as there is no real difference in their design principles. As there are few papers about GF(p) implementations, this topic will not be considered in this article apart from GF(2^m). It should be noted that no paper about implementation of multi-scalar multiplication has been found.

[2] Some results are incorrect in this paper.

[3] It should be emphasized that this algorithm is fully sequential. Consequently, data dependencies may affect the performances of the multiplier.

References are nevertheless given in order to stimulate work in this way.

Some interesting references about the arithmetic operators are not included. However, it is not the aim of this paper to cover every method to perform finite field arithmetic. Further references will be found in the referenced papers.

4.1. Reconfigurable and dedicated ASIC

While very high performances are required, the flexibility could be sacrificed at the advantage of data throughput and chip area. The following works deal with hardwired field sizes and irreducible polynomials (or modulo). For ASIC implementations, the circuit is specialized forever. Nevertheless, while using reconfigurable platforms like FPGAs, the functionality of the system can be modified through reconfiguration. Until now, proposed applications have only supported complete reconfiguration. Future implementations allowing on-the-fly reconfiguration of a dedicated modular reduction circuit could be an efficient solution and should help to recover the loss of flexibility. Of course, the circuit must be reconfigured fast enough to not annihilate the advantage of a hardwired approach: the speed.

A fully hardwired asynchronous ASIC is presented in [34]. It is based on an ONB multiplier. This work is refined in [26] and FPGA implementations were also provided. Another ASIC implementation, based on synchronous polynomial basis multiplier, can be found in [75].

Designs for curves defined over fields GF(p) are explained in [61,62,8,48]. The first three architectures use systolic array Montgomery modular multipliers while [48] uses Montgomery multiplication based on parallel schoolbook multipliers. A systolic array Montgomery modular multiplier is also used in [50] but for GF(2^m) curves.

Other papers using reconfigurable hardware can be read in [55,7]. They focus on an efficient hardware/software partition for the scalar multiplication. Reconfigurable architectures based on digit-serial polynomial basis multiplier can be found in [60, 47,70,4]. The paper [4] also mentions ASIC results. It seems to be the fastest implementation known. In the paper [66], Hessian elliptic curves [72] are used in order to extract more parallelism. The same design using the LD algorithm is presented in [67]. Nevertheless, the low frequency of this design shows that large multipliers must be handled with care. More about Karatsuba multipliers can be found in [65,22,28].

Following the design principles, [4] seems the best representative of a high-speed architecture using a hardwired irreducible polynomial, a single field size and an unknown base point P. It is a pipelined and parallel structure, based on a large Most Significant Digit (MSD)-first multiplier and an accurate scheduling of the LD algorithm.

Firstly, as shown in Figs. 1 and 2, the multiplier and the arithmetic Squaring/Addition unit are used in parallel. A pipelined multiplier is then needed in order to achieve a high operating frequency. It is presented in Fig. 3, with w as the word size, currently equal to one quarter of the field size.

Secondly, the scheduling of the LD algorithm is modified in order to prevent idle cycles from happening. Those lost cycles come from the load/store operations and the pipelined behavior of the multiplier. It is really important to avoid them in order to fully benefit from the capability of large multipliers. The scheduling is shown in Fig. 2, where M_p stands for the cycles needed to perform a single multiplication. The triangle indicates the end of a multiplication [4]. In particular, the output of a multiplication can not be used directly as the input of the next multiplication.

Assuming a multiplication is achieved in 4 cycles and a LD algorithm free of idle cycle, a full computation only needs $24(m-1)$ cycles. Including the rough $3m/2$ cycles needed to convert the final result back to affine coordinates, $25.5(m-1)$ cycles are required for the scalar multiplication. While it is currently the best result presented in the open literature it can nevertheless be improved. Indeed, a 3-cycle Karatsuba multiplier could be used to probably further reduce the area–time complexity of the design. A computing time of $19.5(m-1)$ will then be expected.

4.2. Generator

This section addresses generators of elliptic curve processors. For designs manipulating bitstream of

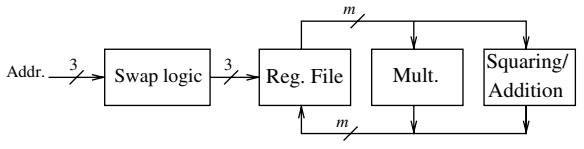

Fig. 1. Reconfigurable and dedicated ASIC global architecture.

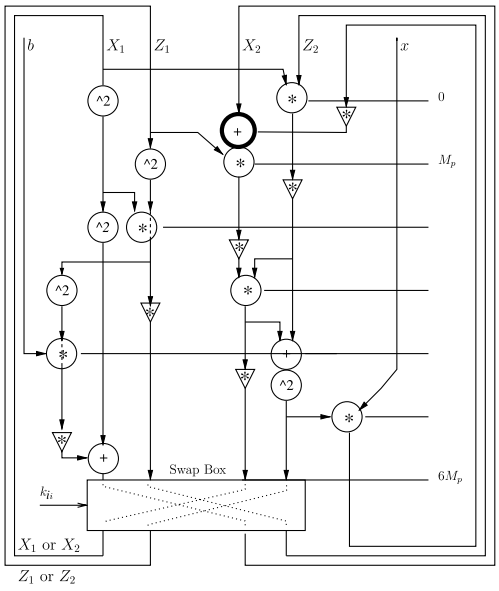

Fig. 2. LD algorithm scheduling.

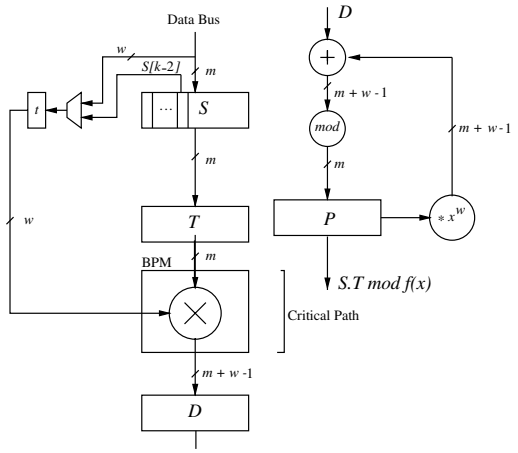

Fig. 3. MSD-first multiplier.

reconfigurable hardware, this category is full of sense. For designs fully parameterizable before synthesis, this category may seem a little artificial since all architectures are parameterizable up to a given level. Despite this, as some papers emphasize this feature, it is interesting to extract it. Generators are particularly useful for ONB based architectures as they strongly depend on the field and the irreducible polynomial.

An architecture based on ONB multiplier is presented in [44]. Bitstream manipulation is proposed to obtain flexibility. Another generator for both polynomial basis and ONB can be found in [40]. Papers [76,15] are based on ONB multipliers and focus on the hardware/software partitioning problem.

A paper about a simple generator can be found in [39]. Another generator is presented in [11] and is refined in [10,27]. Polynomial basis multipliers are used and a Karatsuba multiplier is deployed in [27]. Another architecture based on a Karatsuba-like approach, with a slightly worse complexity class, can be found in [25,35].

Generators are not described in details here as they are based on the use of tools and not really on specific hardware architectures.

4.3. Versatile scalar multiplier

Versatile processors can manage arbitrary field sizes and irreducible polynomials without reconfiguration. They can be useful in highly heterogeneous environments, where vendor preferences, individual security requirements and processor capabilities are unknown a priori.

Three main kinds of architectures can be found in the literature. The first ones can handle any kind of fields up to a given degree, but cannot take advantage of specific irreducible polynomials. The second ones use specific multiplier to save cycles when specific irreducible polynomials are used. The third ones are based on both generic and specific architecture to deal with generic and specific problems. Architectures processing only the recommended fields and irreducible polynomials have not been found. It should be noted that worst case architectures are probably not the best solution. As systems using only completely random field sizes and irreducible polynomials are unlikely, hybrid designs will be preferred.

Architectures handling worst case field sizes can be found in [63,19,41]. The ASIC design [19] is particularly unusual since affine coordinates are used. An architecture using specific polynomial basis multiplier and a new data conversion method can be found in [57]. A design, able to handle both generic and specific problems was presented in [32]. The same architecture with an improved multiplier is later described in [23].

In accordance to the design principles, [23] seems to be the best example of a high performance versatile architecture for scalar multiplication. It is able to handle any field size up to 255-bits and can efficiently process the fields recommended by [56,64]: GF(2^{163}), GF(2^{193}) and GF(2^{233}). Specific hardware

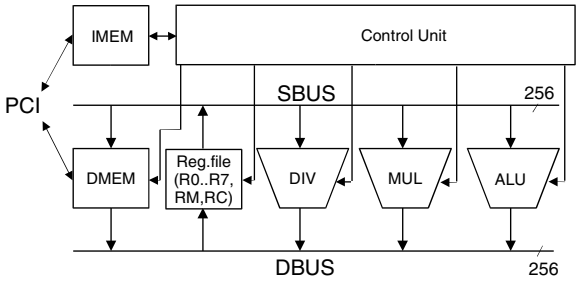

Fig. 4. Versatile global architecture.

is devoted to these named curves to speed up the computation. The LD algorithm is used on a microprogrammable load/store architecture (Fig. 4). The instruction set is composed of memory, arithmetic and control instructions. The reader is referred to the paper for more details.

The MSD-first 64×256-bit modular multiplier circuit is shown in Fig. 5. It is capable of handling named curves as well as generic curves. A hardwired modular reducer is utilized for named curves and the multiplier circuit is reused to perform modular reduction for generic curves. A partial reduction technique is also employed to avoid costly shift and mask operations required to extract field-size operands smaller than the data path width.

The squaring for named curves is performed in one clock cycle using dedicated hardware. For generic curves, squaring is translated to a generic multiplication. The transformation of the final result back to affine coordinates is executed by a dedicated divider based on extended Euclidean algorithm. The full reduction of partially reduced results is also achieved by this circuit.

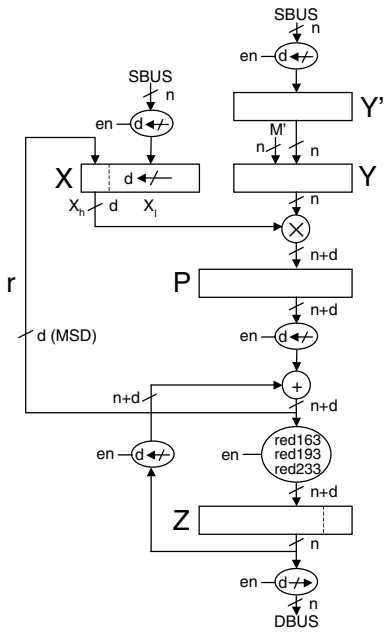

Fig. 5. Multiplier architecture.

This architecture shows the possibility of reaching flexibility without too much performances penalty. Specific hardware is devoted to named curves and existing logic is reused for generic curves. As a result, the additional resources needed to support generic curves are minimal.

4.4. General purpose

Compared with versatile processor, the use of a generic datapath provides even more flexibility at the expense of performances. This kind of processor is especially interesting while dealing with different cryptosystems, requiring a different kind of arithmetic or different field sizes. It is usually based on a 32- or 64-bit datapath.

A GF(p) processor suitable for RSA, DSA and ECDSA is presented in [37]. It is based on a Montgomery multiplier with a 16-bit digit size, implemented in a DSP TMS320C6201. Another processor using MAC2424 (24×24-bit) for GF(p^{10}) OEF is presented in [16]. It includes an interesting analysis of the precomputation needed by the signed window scalar multiplication method. More about GF(p^m) arithmetic can be found in [12,6].

Two 64-bit dual-field processors can be found in [68,24]. They also use Montgomery multipliers and are both able to compute integer and binary polynomial arithmetic (for GF(p) and GF(2^m)). In particular, RSA, DSA, ECDSA and DH are supported. The architecture proposed in [68] is an ASIC while [24] is implemented on FPGA with ASIC extrapolations.

As a typical example of an architecture based on a 64-bit datapath, the paper [24] was chosen. An overview of this microprogrammable VLIW processor is shown in Fig. 6. The core is based on a 64×64-bit multiplier able to process both integer and binary polynomial arithmetic. In particular, as it is the most frequent arithmetic function required by the considered algorithms, it implements the multiply-accumulate operation with support for multi-precision arithmetic. The processor currently implements the main public-key protocols like RSA, DSA, ECDSA and DH with arbitrary key sizes and curves.

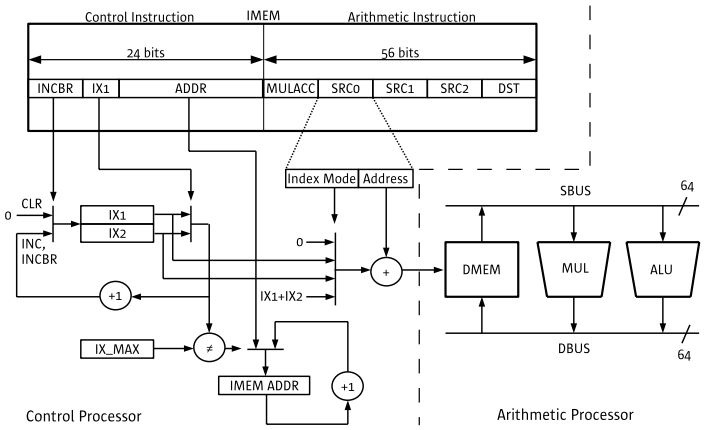

Fig. 6. General purpose global architecture.

The algorithms used for ECC scalar multiplication are the LD method for GF(2^m) and binary algorithm with mixed coordinates for GF(p). Nevertheless, as it is a microprogrammable processor, any other algorithms can be used. A simple assembler is written to develop firmware for the processor.

As expected, the Montgomery method computes the modular multiplication. The GF(2^m) squaring is performed by a general multiplication and the inversion is computed with the little Fermat's theorem. The architecture of the multiplier is presented in Fig. 7. It is composed of: a 64×64-bit array for partial products generation, a Carry Save Adder (CSA) tree for the summation of partial products and a 128-bit Carry Propagate Adder (CPA) to convert the redundant representation of the CSA tree to classical binary numbers. Moreover, the CSA tree can be modified to obtain the XOR result in addition to the integer result. This functionality is essential to perform both integer and polynomial arithmetic. A specific store, shift and add of the 64 most significant carry bits is also needed to achieve efficient multi-precision arithmetic.

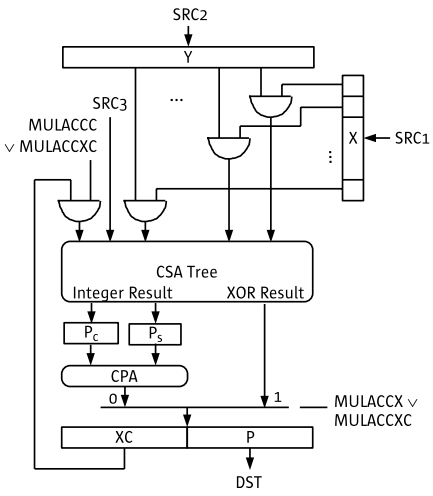

Fig. 7. Multiply-accumulator.

More on dual field arithmetic units can be found in [69,31,77,18].

For the support of binary polynomial arithmetic, the proposed multiplier only needs a 5% increase in terms of logic required. Regarding the working frequency, it should be emphasized that the ASIC results provided are highly theoretical and must be handled with care. As explained in the paper, data dependency problems may occur due to the latency of a pipelined version of this circuit. Memory access should also be of prime concern. Nevertheless, this design gives a good overview about the possibilities of a general purpose processor for public-key cryptography based on GF(p) and GF(2^m) arithmetic.

4.5. Multi-scalar multiplier

Multi-scalar multiplications are required in various protocols like the ECDSA verification. This operation can be done by addition of individual scalar multiplications. However, as each individual scalar multiplication does not need to be known, more efficient techniques can be used. To the best of our knowledge, there is currently no hardware implementation of this operation. Several algorithms have been proposed and are listed below.

A good implementation of this operation should be able to take advantage of specific multi-scalar

Table 1
Performance summary of state of the art

Name, Ref./Computation, IrrPol	Device/Size	Frequency/Time	Multiplier/Remarks
Ansari et al. [4]	XC2V2000	100 MHz	MSB pipelined $D = 41$
2^m, 163-bit, NIST	8300 luts	41 μs	7 bRAMs, No final inv.
Ansari et al. [4]	0.18 μm CMOS	167 MHz	MSB pipelined $D = 41$
2^m, 163-bit, NIST	36000 gates	21 μs	No final inv.
Sozzani et al. [75]	0.13 μm CMOS	417 MHz	2 Mults $D = 8$
2^m, 163-bit, NIST	0.59 mm^2	30 μs	
Saqib et al. [67]	XCV3200E	9.99 MHz	Parallel Karatsuba
2^m, 191-bit, trino.	18314 slices	56 μs	24 bRAMs, No final inv.
Shu et al. [70]	XCV2000E-7	68.9 MHz	MSD 6 mult $D = 32,8$
2^m, 163-bit, NIST	25763 luts	48 μs	
Shu et al. [70]	XCV2000E-7	67.9 MHz	MSD 6 mult $D = 32,8$
2^m, 233-bit, NIST	35800 luts	89 μs	
Orlando et al. [61]	XCV1000E-8	40 MHz	MMM, Booth, $D = 4$
p, 192-bit, NIST	11416 luts	3 ms	35 bRAMs
McIvor et al. [48]	XC2VP125-7	39.5 MHz	MMM
p, 256-bit, Any	15755 slices	3.84 ms	256 Mults 18×18
Jarvinen et al. [39]	XC2V8000-5	90.2 MHz	
2^m, 163-bit, NIST	18079 slices	106 μs	Generator
Grabbe et al. [27]	XC2V6000	100 MHz	Hybride KOA
2^m, 233-bit, NIST	19440 luts	130 μs	Generator
Daneshbeh et al. [19]	0.18 μm CMOS	700 MHz	Div/Mult
2^m, <256-bit, Any	35000 gates	734 μs	Systo. array, Aff. Coord.
Eberle et al. [23]	XCV2000E-7	66.4 MHz	MSD $D = 64$
2^m, <256-bit, NIST	20068 luts	144 μs	Result for 163-bit
2^m, <256-bit, Any		302 μs	Result for 163-bit
Itoh et al. [37]	TMS320C6201	200 MHz	
p, 160-bit, Any		3.09 ms	DSP, $a \neq 0$
Satoh et al. [68]	0.13 μs CMOS		MMM, $D = 64$
2^m, 160-bit, Any	117500 gates	190 μs	510.2 MHz
p, 160-bit, Any		1.21 ms	137.7 MHz

multiplication algorithms and also efficiently compute single scalar multiplications.

Papers [73,52] explain the classical algorithms proposed for the multi-scalar multiplication. An interesting analysis of this operation with the use of precomputation in affine coordinates can be found in [59]. Transformation of the LD algorithm for the case of multi-scalar multiplication is proposed in [2,5].

4.6. Performances summary

In order to give a global overview about the performances of each approach, the authors selected the most relevant implementations in their opinion. The performances are summarized in Table 1. XCV devices are Xilinx Virtex FPGAs. Several abbreviations are used in this table. MSB/LSB stands for Most/Least Significant Bit first polynomial basis multiplier. D is the digit size of a serial by parallel multiplier. NIST means that a specific trinomial or pentanomial is used. MMM stands for the Montgomery Modular Multiplication algorithm and Booth for the Booth signed digit recoding technique.

5. Conclusion

Concerning the design of hardware accelerators for elliptic curve cryptography, many context can occur. First, it could be used for very high performances while sacrificing flexibility. Moreover, automatic generation can be a required feature.

Conversely, flexibility about key sizes and curves can be needed. Even more, support for various cryptosystem can be mandatory. Finally, support for efficient multi-scalar multiplication can be necessary.

For all contexts, it is important to take care of several principles. Efficient coordinates for representing points together with a fast scalar multiplication algorithm must be chosen. Then, according to the capabilities of the selected multiplier architecture, an accurate scheduling of the scalar multiplication algorithm has to be determined. Furthermore, the designer must be sure that integration of large multipliers results in a significant speedup. If possible, the pipelining technique should be applied. Finally, an appropriate algorithm for the conversion of the final result back to affine coordinates has to be implemented.

While a flexible feature is achieved by reconfiguration, it is currently only a complete reconfiguration. Future implementations allowing on-the-fly reconfiguration of a dedicated modular reduction circuit could be an efficient solution and should help to recover the loss of flexibility. For versatile processors, more work could be done to take the best advantage of full size hardware while computing with smaller operands. The open literature also lacks of multi-scalar multiplication hardware architecture. A good implementation of this operation should be able to take advantage of specific multi-scalar multiplication algorithms and also efficiently compute single scalar multiplications.

Acknowledgement

The authors wish to thank Lejla Batina for her interesting comments.

References

[1] G.B. Agnew, T. Beth, R.C. Mullin, S.A. Vanstone, Arithmetic operations in GF(2^m), Journal of Cryptology 6 (1) (1993) 3–13.

[2] T. Akishita, Fast simultaneous scalar multiplication on elliptic curve with montgomery form, in: Selected Areas in Cryptography (SAC), LNCS 2259, 2001, pp. 255–267.

[3] ANSI, ANSI X9.62 The elliptic curve digital signature algorithm (ECDSA). Available from: <http://www.ansi.org>.

[4] B. Ansari, M. Anwar Hasan, High performance architecture of elliptic curve scalar multiplication, Tech. Report CACR 2006-01, 2006. Available from: <http://www.cacr.math.uwaterloo.ca/techreports/2006/cacr2006-01.pdf>.

[5] G. Bai, G. Chen, H. Chen, Fast scalar multiplications of elliptic curve cryptosystems over binary fields, in: SKLOIS Information Security and Cryptology (CISC), 2005, pp. 315–323.

[6] J.-C. Bajard, L. Imbert, C. Negre, T. Plantard, Efficient multiplication in GF(p^k) for elliptic curve cryptography, in: IEEE Symposium on Computer Arithmetic (ARITH-16), 2003, pp. 181–187.

[7] S. Bajracharya, C. Shu, K. Gaj, T. El-Ghazawi, Implementation of elliptic curve cryptosystems over GF(2^n) in optimal normal basis on a reconfigurable computer, in: Field-Programmable Logic and Applications (FPL), LNCS 3203, 2004, pp. 1001–1005.

[8] L. Batina, G. Bruin-Muurling, S.B. Örs, Flexible hardware design for RSA and elliptic curve cryptosystems, in: The Cryptographer's Track at RSA Conference (CT-RSA), LNCS 2964, 2004, pp. 250–263.

[9] L. Batina, S.B. Örs, B. Preneel, J. Vandewalle, Hardware architectures for public key cryptography, Elsevier Integration, the VLSI Journal, special issue on Embedded Cryptographic Hardware, 34 (2003) 1–2, pp. 1–64.

[10] M. Bednara, M. Daldrup, J. Teich, J. von zur Gathen, J. Shokrollahi, Tradeoff analysis of FPGA based elliptic curve cryptography, in: IEEE Symposium on Circuits and Systems (ISCAS), 5, 2002, pp. 797–800.

[11] M. Bednara, M. Daldrup, J. von zur Gathen, J. Shokrollahi, J. Teich, Reconfigurable implementation of elliptic curve crypto algorithms, in: Reconfigurable Architectures Workshop (RAW), 2002.

[12] G. Bertoni, J. Guajardo, S. Kumar, G. Orlando, C. Paar, T. Wollinger, Efficient GF(p^m) Arithmetic architectures for cryptographic applications, in: The Cryptographer's Track at RSA Conference (CT-RSA), LNCS 2612, 2003, pp. 158–175.

[13] I. Blake, G. Seroussi, N.P. Smart, Elliptic curves in cryptography, London Mathematical Society Lecture Note Series, 1999.

[14] B. Canvel, A. Hiltgen, S. Vaudenay, M. Vuagnoux, Password interception in a SSL/TLS channel, in: Cryptology Conference (CRYPTO), LNCS 2729, 2003, pp. 583–599.

[15] R.C.C. Cheung, W. Luk, P.Y.K. Cheung, Reconfigurable elliptic curve cryptosystems on a chip, in: Design, Automation and Test in Europe (DATE), 1, 2005, pp. 24–29.

[16] J.W. Chung, S.G. Sim, P.J. Lee, Fast implementation of elliptic curve defined over GF(p^m) on CalmRISC with MAC2424 coprocessor, in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 1965, 2000, pp. 57–70.

[17] A. Cilardo, L. Coppolino, N. Mazzocca, L. Romano, Elliptic curve cryptography engineering, Proceedings of the IEEE 94 (2) (2006) 395–406.

[18] F. Crowe, A. Daly, W. Marnane, A scalable dual mode arithmetic unit for public key cryptosystems, in: IEEE Symposium on Information Technology: Coding and Computing (ITCC), 1, 2005, pp. 568–573.

[19] A. K. Daneshbeh, M.A. Hasan, Area efficient high speed elliptic curve cryptoprocessor for random curves, in: IEEE Symposium on Information Technology: Coding and Computing (ITCC), 2, 2004, pp. 588–592.

[20] G. Meurice de Dormale, P. Bulens, J.-J. Quisquater, Efficient modular division implementation: ECC over GF(p) affine coordinates application, in: Field-Programmable Logic and Applications (FPL), LNCS 3203, 2004, pp. 231–240.

[21] G. Meurice de Dormale, J.-J. Quisquater, Iterative modular division over GF(2^m): novel algorithm and implementations

on FPGA, in: Reconfigurable Computing: Architectures and Applications (ARC), LNCS 3985, 2006, pp. 370–382.
[22] Z. Dyka, P. Langendoerfer, Area efficient hardware implementation of elliptic curve cryptography by iteratively applying Karatsuba's method, in: Design, Automation and Test in Europe (DATE), 3, 2005, pp. 70–75.
[23] H. Eberle, N. Gura, S. Chang-Shantz, A cryptographic processor for arbitrary elliptic curves over GF(2^m), in: Application-Specific Systems, Architectures, and Processors (ASAP), 2003, pp. 444–454.
[24] H. Eberle, N. Gura, S.C. Shantz, V. Gupta, L. Rarick, S. Sundaram, A public-key cryptographic processor for RSA and ECC, in: Application-Specific Systems, Architectures, and Processors (ASAP), 2004, pp. 98–110.
[25] M. Ernst, M. Jung, F. Madlener, S. Huss, R. Blümel, A reconfigurable system on chip implementation for elliptic curve cryptography over GF(2^m), in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 2523, 2002, pp. 381–399.
[26] M. Ernst, S. Klupsch, O. Hauck, S.A. Huss, Rapid prototyping for hardware accelerated elliptic curve public-key cryptosystems, in: IEEE Rapid System Prototyping (RSP), 2001, pp. 24–31.
[27] C. Grabbe, M. Bednara, J. von zur Gathen, J. Shokrollahi, J. Teich, A high performance vliw processor for finite field arithmetic, in: Reconfigurable Architectures Workshop (RAW), 2003.
[28] J. von zur Gathen, J. Shokrollahi, Efficient FPGA-based karatsuba multipliers for polynomials over $\mathbb{F}_2$, in: Selected Areas in Cryptography (SAC), LNCS 3897, 2005, pp. 359–369.
[29] P. Gaudry, F. Hess, N.P. Smart, Constructive and destructive facets of Weil descent on elliptic curves, Journal of Cryptology 15 (2002) 19–46.
[30] D.M. Gordon, A survey of fast exponentiation methods, Journal of Algorithms 27 (1998) 129–146.
[31] J. Großschädl, A Bit-Serial unified multiplier architecture for finite fields GF(p) and GF(2^m), in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 2162, 2001, pp. 202–219.
[32] N. Gura, S.C. Shantz, H. Eberle, D. Finchelstein, S. Gupta, V. Gupta, D. Stebila, An end-to-end systems approach to elliptic curve cryptography, in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 2523, 2002, pp. 349–365.
[33] D. Hankerson, A. Menezes, S. Vanstone, Guide to elliptic curve cryptography, Springer-Verlag, 2004.
[34] O. Hauck, A. Katoch, S.A. Huss, VLSI system design using asynchronous wave pipelines: A 0.35 μm CMOS 1.5 GHz elliptic curve public key cryptosystem chip, in: Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC), 2000, pp. 188–197.
[35] S.A. Huss, M. Jung, F. Madlener, High speed elliptic curve crypto processors: design space exploration by means of reconfigurable hardware, International Scientific and Applied Conference – Information Security, 2004.
[36] IEEE P1363, Standard specifications for public key cryptography, 1999.
[37] K. Itoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara, Fast implementation of public-key cryptography on a DSP TMS320C6201, in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 1717, 1999, pp. 61–72.
[38] T. Itoh, S. Tsujii, A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases, Information and Computation 78 (3) (1988) 171–177.
[39] K. Järvinen, M. Tommiska, J. Skyttä, A scalable architecture for elliptic curve point multiplication, in: IEEE Field-Programmable Technology (FPT), 2004, pp. 303–306.
[40] T. Kerins, W. Marnane, E. Popovici, Design for reuse of elliptic curve cryptosystem processors for FPGAs, in: Irish Signals and Systems Conference (ISSC), 2004, pp. 577–582.
[41] T. Kerins, W.P. Marnane, E.M. Popovici, An FPGA implementation of a flexible secure elliptic curve cryptography processor, in: Reconfigurable Computing: Architectures and Applications (ARC), 2005.
[42] B. King, An improved implementation of elliptic curves over GF(2^n) when using projective point arithmetic, in: Selected Areas in Cryptography (SAC), LNCS 2259, 2001, pp. 134–150.
[43] N. Koblitz, Elliptic curve cryptosystem, Mathematics of Computation 48 (1987) 203–209.
[44] P. Leong, I. Leung, A microcoded elliptic curve processor using FPGA technology, IEEE Transactions on VLSI Systems 10 (5) (2002) 550–559.
[45] J. López, R. Dahab, Fast multiplication on elliptic curves over GF(2^m), in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 1717, 1999, pp. 316–327.
[46] J. Lutz, M.A. Hasan, High-performance finite field multiplier for cryptographic applications, Advanced Signal Processing Algorithms, Architectures, and Implementations XIII – SPIE 5205 (2003) 541–551.
[47] J. Lutz, M.A. Hasan, High performance FPGA based elliptic curve cryptographic co-processor, in: IEEE Symposium on Information Technology: Coding and Computing (ITCC), 2, 2004, pp. 486–492.
[48] C. McIvor, M. McLoone, J. McCanny, An FPGA elliptic curve cryptographic accelerator over GF(p), in: Irish Signals and Systems Conference (ISSC), 2004, pp. 589–594.
[49] A. Menezes, E. Teske, A. Weng, Weak fields for ECC, in: The Cryptographer's Track at RSA Conference (CT-RSA), LNCS 2964, 2004, pp. 366–386.
[50] N. Mentens, S.B. Örs, B. Preneel, An FPGA implementation of an elliptic curve processor GF(2^m), ACM Great Lakes Symposium on VLSI (2004) 454–457.
[51] V. Miller, Uses of elliptic curves in cryptography, in: Cryptology Conference (CRYPTO), LNCS 218, 1985, pp. 417–426.
[52] B. Möller, Algorithms for multi-exponentiation, in: Selected Areas in Cryptography (SAC), LNCS 2259, 2001, pp. 165–180.
[53] P. Montgomery, Speeding the pollard and elliptic curve methods of factorization, Mathematics of Computation 48 (177) (1987) 243–264.
[54] D. Naccache, N.P. Smart, J. Stern, Projective coordinates leak, in: Theory and Applications of Cryptographic Techniques (EUROCRYPT), LNCS 3027, 2004, pp. 257–267.
[55] N. Nguyen, K. Gaj, D. Caliga, T. El-Ghazawi, Implementation of elliptic curve cryptosystems on a reconfigurable computer, in: IEEE Field-Programmable Technology (FPT), 2003, pp. 60–67.
[56] U.S. Department of Commerce/National Institute of Standards and Technology (NIST), Digital Signature Standard (DSS), FIPS PUB 182-2change1, 2000.

[57] S. Okada, N. Torii, K. Itoh, M. Takenaka, Implementation of elliptic curve cryptographic coprocessor over GF(2^m) on an FPGA, in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 1965, 2000, pp. 25–40.
[58] K. Okeya, K. Sakurai, Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the y-coordinate on a montgomery-form elliptic curve, in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 2162, 2001, pp. 126–141.
[59] K. Okeya, K. Sakurai, Fast multi-scalar multiplication methods on elliptic curves with precomputation strategy using montgomery trick, in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 2523, 2002, pp. 564–578.
[60] G. Orlando, C. Paar, A high-performance reconfigurable elliptic curve processor for GF(2^m), in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 1965, 2000, pp. 41–56.
[61] G. Orlando, C. Paar, A scalable GF(p) elliptic curve processor architecture for programmable hardware, in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 2162, 2001, pp. 356–371.
[62] S.B. Örs, L. Batina, B. Preneel, J. Vandewalle, Hardware implementation of an elliptic curve processor over GF(p), in: Application-Specific Systems, Architectures, and Processors (ASAP), 2003, pp. 433–443.
[63] M.J. Potgieter, B.J. van Dyk, Two hardware implementations of the group operations necessary for implementing an elliptic curve cryptosystem over a characteristic two finite field, in: IEEE Africon Conference in Africa (AFRICON), 2002.
[64] Certicom Research, SEC 2: Recommended elliptic curve domain parameters, v1.0, 2000. Available from: <http://www.secg.org/>.
[65] F. Rodríguez-Henríquez, Ç.K. Koç, On fully parallel karatsuba multipliers for GF(2^m), in: Computer Science and Technology (CST), 2003, pp. 405–410.
[66] N.A. Saqib, F. Rodríguez-Henríquez, A. Díaz-Pérez, A parallel architecture for computing scalar multiplication on hessian elliptic curves, in: Symposium on Information Technology: Coding and Computing (ITCC), 2, 2004, pp. 493–497.
[67] N.A. Saqib, F. Rodríguez-Henríquez, A. Díaz-Pérez, A parallel architecture for fast computation of elliptic curve scalar multiplication over GF(2^m), in: Parallel & Distributed Processing Symposium (IPDPS), 2004.
[68] A. Satoh, K. Takano, A scalable dual-field elliptic curve cryptographic processor, IEEE Transactions Computers 52 (4) (2003) 449–460.
[69] E. Savas, A.F. Tenca, Ç.K. Koç, A scalable and unified multiplier architecture for finite fields GF(p) and GF(2^m), in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 1965, 2000, pp. 277–292.
[70] C. Shu, K. Gaj, T. El-Ghazawi, Low latency elliptic curve cryptography accelerators for NIST curves on binary fields, in: IEEE Field-Programmable Technology (FPT), 2005, pp. 309–310.
[71] Ecrypt, Vampire lab, The side channel cryptanalysis lounge, 2006. Available from: <http://www.crypto.ruhr-uni-bochum.de/en_sclounge.html>.
[72] N.P. Smart, The hessian form of an elliptic curve, in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 2162, 2001, pp. 118–125.
[73] J.A. Solinas, Low-weight binary representations for pairs of integers, Tech. Report CORR 01-41, Department of Combinatorics & Optimization. Available from: <http://www.cacr.math.uwaterloo.ca/techreports/2001/corr2001-41.ps>, 2001.
[74] L. Song, K.K. Parhi, Low energy digit-serial/parallel finite field multipliers, Journal of VLSI Signal Processing 19 (2) (1998) 149–166.
[75] F. Sozzani, G. Bertoni, S. Turcato, L. Breveglieri, A parallelized design for an elliptic curve cryptosystem coprocessor, in: Symposium on Information Technology: Coding and Computing (ITCC), 1, 2005, pp. 626–630.
[76] N. Telle, W. Luk, R.C.C. Cheung, Customising hardware designs for elliptic curve cryptography, in: Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2004, pp. 274–283.
[77] J. Wolkerstorfer, Dual-field arithmetic unit for GF(p) and GF(2^m), in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 2523, 2002, pp. 500–514.
[78] H. Wu, Low complexity bit-parallel finite field arithmetic using polynomial basis, in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 1717, 1999, pp. 280–291.
[79] H. Wu, On complexity of polynomial basis squaring in F_{2^m}, in: Selected Areas in Cryptography (SAC), LNCS 2012, 2000, pp. 118–129.
[80] H. Wu, Montgomery multiplier and squarer for a class of finite fields, IEEE Transactions on Computers 51 (5) (2002) 521–529.

Guerric Meurice de Dormale was born in Brussels, Belgium, in 1979. He received the Electrical Engineering degree from the Université catholique de Louvain (UCL), Louvain-la-Neuve, Belgium, in 2003. He is currently working toward Ph.D. degree in the UCL Crypto Group, under the supervision of Prof. J.-J. Quisquater. His research interests include digital design and field programmable gate arrays, design and hardware implementation of asymmetric ciphers, and computer arithmetic.

Jean-Jacques Quisquater is professor of cryptography and multimedia security at the Department of Electrical Engineering, University of Louvain, Louvain-la-Neuve, Belgium, where he is responsible, at least at the scientific level, of many projects related to smart cards (protocols, implementations, side-channels), to secure protocols for communications, digital signatures, payTV, protection of copyrights and security tools for electronic commerce. He was the main designer for several coprocessors for powerful smart cards: CORSAIR (Philips) and FAME (Philips). He holds 17 patents in the field of smart cards. He is co-inventor of an identification cryptographic scheme, the so-called GQ scheme.