Parallel Hardware Architectures for the Cryptographic Tate Pairing

Guido M. Bertoni[1], Luca Breveglieri[2], Pasqualina Fragneto[1], and Gerardo Pelosi[2]
(Corresponding author: Gerardo Pelosi)

ST Microelectronics[1]
Via Olivetti, 20041 Agrate B.za, Milano, Italy
Politecnico di Milano[2]
Piazza L. Da Vinci 32, 20133 Milano, Italy (Email: pelosi@elet.polimi.it)

(Received June 27, 2006; revised and accepted Sept. 5, 2006)

Abstract

Identity-based cryptography uses pairing functions,which are sophisticated bilinear maps defined on elliptic curves.Computing pairings efficiently in software is presently a relevant research topic. Since such functions are very complex and slow in software, dedicated hardware *(HW)* implementations are worthy of being studied, but presently only very preliminary research is available. This work affords the problem of designing parallel dedicated *HW* architectures, i.e.,co-processors, for the Tate pairing, in the case of the Duursma-Lee algorithm in characteristic 3. Formal scheduling methodologies are applied to carry out an extensive exploration of the architectural solution space, evaluating the obtained structures by means of different figures of merit such as computation time, circuit area and combinations thereof.Comparisons with the *(few)* existing proposals are carried out, showing that a large space exists for the efficient parallel*HW* computation of pairings.

Keywords: Area-time tradeoff, parallelism, scheduling, Tate pairing

1 Introduction

Pairings playan important role in identity-based encryption (IBE). Such form of cryptography was introduced by Shamir in 1984 in [20], where a public key is derived from public identifiable information such as an e-mail address, and the corresponding private key is created by binding the user identity with a trusted authority's secret key. This idea avoids the reliance on certificates to validate the authenticity of a public key and, in some situations, simplifies the infrastructure of a public key system. In the same paper, Shamir left the task of building an effective IBE algorithm as an open problem. Solutions for implementing this idea were presented by Boneh and Franklin [5], Cocks [6] and Sakai *etal.* [19]. Apart from the Cocks scheme, both the Boneh-Franklin scheme and the Sakai *et al.* scheme make use of bilinear maps, or pairings, defined over elliptic curves. Nowadays there are many other cryptographic protocols benefiting from using pairings, including a variety of signature and key-establishment schemes. A survey of pairing-based cryptographic protocols can be found in [7].

Today the Tate pairing [9] is considered as the most convenient function in terms of computational cost. Starting from2002 many progresses have taken place for the computation of the Tate pairing in the case of super singular elliptic curves. The original algorithm was designed by Miller [17] and was modified by Barreto *at al.* [1], who obtained the so-called BKLS algorithm, while a closed formula in characteristic3 was presented for the first time by Duursma and Lee [8]. More recently the Duursma-Lee algorithm has been improved by Kwon [16], Barreto *et al.* [3] and Granger *et al.* [12]. Related works about the implementation of such algorithms by means of dedicated hardware (HW) architectures have been done by Grabher *et al.* [11] and Kerins *et al.* [14]. In [11] an evaluation of FPGA architectures for polynomial and normal bases in characteristic 3 is shown, in the case of the Duursma-Lee and Kwon algorithms; there the emphasis is on the efficiency of arithmetic in the field $\mathbb{F}_{3^m}$. In [14] the focus is instead on high-level parallelism, in the Duursma-Lee algorithm, for the operations in the extension field $\mathbb{F}_{q^k}$ (in particular with $q = 3^m$): these authors reduce computation time,but at the expense of a high number of function units and hence of the silicon area of the device.

In this paper we describe a method for designing parallel HW architectures for the Duursma-Lee algorithm in characteristic 3.Here emphasis is on executing in parallel the operations in the base field $\mathbb{F}_{3^m}$ (not in$\mathbb{F}_{(3^m)^k}$ as in [14]), and we show that a large space for such solutions exists. We balance the trade-off between the speed and the size of the dedicated HW device.Such considerations are applicable to the design of HW accelerators for

pairing-based primitives in embedded systems as well as in internet servers.

We point out that the software (SW) computation of the Tate pairing is relatively slow [3, 12],and therefore HW solutions are desirable. The existing studies in [11, 14] are preliminary and do not consider parallelism or do so only at high level [11], thus exploring the solution space at a limited extent. Moreover the Duursma-Lee algorithm (as well as the others) is decidedly complex and hence is over the threshold of manual parallelization. In fact we adopt the methods and the techniques of the theory of automated algorithm scheduling [10], for an exhaustive automated exploration of the parallelism issue for the selected algorithm. We develop all the necessary SW tools and we generate several parallel architectures. Then we evaluate them with respect to typical figures of merit such as computation time (T), circuit area (A) and combinations thereof (AT, etc.), and we compare them with respect to literature solutions.

The paper is organized as follows: Section 2summarizes the basic notions about the Tate pairing; Section3 outlines the architecture and sketches the parallelization methodology; Section 4 sketches arithmetic details and function units; Section 5presents several parallel architectures for the Tate pairing;Section 6 evaluates and compares such architectures to literature solutions; and Section 7 contains the conclusions and some suggestions for future research directions.

2 Recall of the Tate Pairing

We give here the minimum indispensable mathematical details for introducing the Duursma-Lee algorithm in characteristic 3. Let $E(\mathbb{F}_q)$ be an elliptic curve defined over a field$\mathbb{F}_q$. Let l be a positive integer, co-prime to q, such that $E(\mathbb{F}_q)$ contains a point of order l. In cryptography l is usually a large prime such that $l \mid \#E(\mathbb{F}_q)$ and $l^2 \nmid \#E(\mathbb{F}_q)$. Let k be the smallest integer satisfying $l \mid q^k - 1$. This value k is the embedding degree of the curve with respect to l. The Tate pairing is defined in terms of rational functions over points of an elliptic curve evaluated in a divisor. Let $P \in E(\mathbb{F}_q)[l]$, then $l(P) - l(O)$ is a principal divisor. Hence there exists a rational function $f_P \in \mathbb{F}_{q^k}(E)$ with$div(f_P) = l(P) - l(O)$. Let $Q \in E(\mathbb{F}_{q^k})[l]$ be a point with coordinates in $\mathbb{F}_{q^k}$; then we construct a divisor $D \in Div^0(E)$ such that $D_Q \sim (Q) - (O)$. Such divisor should be chosen so that its support is disjoint from that of the divisor of f_P, and $f_P(D_Q) = \prod_j f_P(Q_j)^{\beta_j}$ with$D_Q = \sum_j \beta_j (Q_j)$.

Over a super singular elliptic curve there always exists a specific endomorphism ϕ (*distortion map*) [21], mapping an l-torsion point to another linearly independent l-torsion point, i.e., ϕ: $E(\mathbb{F}_q)[l] \to E(\mathbb{F}_{q^k})[l]$. In particular, there exists a distortion map linking l-torsion points in the base field to those in the extension field. Papers [1, 13] list several cryptographically interesting curves and distortion maps. Using such maps, Barreto*et al.* [1] have improved the calculation of the Tate pairing by rearranging the Miller algorithm [17]into the Modified Tate pairing:

$$\widehat{e}_l(\dots,\dots) : E(\mathbb{F}_q)[l] \times E(\mathbb{F}_{q^k})[l] \to \mu_l \qquad (1)$$

$$\widehat{e}_l(P,Q) = f_P(\phi(Q))^{\frac{q^k-1}{l}},$$

where μ_l is the subgroup of the l-th roots of unity in $\mathbb{F}^*_{q^k}$. The quotient group $\mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^l$ is isomorphic to elements of order l in $\mathbb{F}^*_{q^k}$ through the final exponentiation by $\frac{q^k-1}{l}$.

Now consider the following super singular elliptic curve, for afield of characteristic 3, i.e., for $\mathbb{F}_q$ with $q = 3^m$:

$$E(\mathbb{F}_q) : y^2 = x^3 - x + b \qquad \text{with } b = \pm 1. \qquad (2)$$

The embedding degree is $k = 6$ and, given the l-torsion point $P(x,y) \in E(\mathbb{F}_{3^m})[l]$, the distortion map of Equation (2) is $\phi(x,y) = (\rho - x, \sigma y) \in E(\mathbb{F}_{3^{6m}})[l]$, where $\sigma, \rho \in \mathbb{F}_{3^{6m}}$ are such that $\rho^3 - \rho - b = 0$ and$\sigma^2 + 1 = 0$ [21].

Duursma and Lee introduced in [8] a faster Tate pairing algorithm using a group order $l = q^3+1 = 3^{3m}+1$, instead of l dividing $q^3 + 1$, and in this way they found a closed formula for $f_P(\phi(Q))$. The pseudo-code of the Duursma-Lee algorithm is shown in Algorithm (2.1) and is taken from [8].

Algorithm 2.1: Duursma-Lee as in [8].

Input: $P, Q \in E(\mathbb{F}_q)[l]$ $P = (x_P, y_P)$ $Q = (x_Q, y_Q)$
Output: $f = f_P(\phi(Q)) \in \mathbb{F}^*_{q^6}/\mathbb{F}^*_{q^3}$
begin
  $f \leftarrow 1$ //# of operations in the base field $\mathbb{F}_{3^m}$ //
  for $i = 1$ **to** m **do**
    $x_P \leftarrow x_P^3$ // 1 cube power //
    $y_P \leftarrow y_P^3$ // 1 cube power //
    $\mu \leftarrow x_P + x_Q + b$ // 2 additions //
    $\lambda \leftarrow -y_P \cdot y_Q \sigma - \mu^2$ // 2 multiplications //
    $g \leftarrow \lambda - \mu\rho - \rho^2$ // no operation //
    $f \leftarrow f \cdot g$ // 13 mul.s plus 50 add.s or sub.s //
    $x_Q \leftarrow \sqrt[3]{x_Q}$ // 1 cube root //
    $y_Q \leftarrow \sqrt[3]{y_Q}$ // 1 cube root //
    // tot: 52 add.s/sub.s, 15 mul.s, $2+2$ cube pow.s/roots //
  endfor
  return f
end

To obtain results compatible with Equation (1) and the BKLS algorithm [1], the output of Algorithm (2.1) must be powered to $\epsilon = 3^{3m} - 1$.Further refinements for super singular elliptic curves in characteristic 3 are outlined in [3, 12, 16], as well as for other characteristics. There the main difference from Algorithm (2.1) is the removal of the cube root operations,replaced with two extra cube powers in the base field and one cube power in the extension field.

To compute the Modified Tate pairing efficiently, a careful implementation of $\mathbb{F}_{3^m}$ arithmetic is needed.Following the approach used in [1, 11, 12, 14], were present the extension field $\mathbb{F}_{3^{6m}}$ as a tower:$\mathbb{F}_{3^{3m}} \cong \mathbb{F}_{3^m}[\rho]/\left(\rho^3-\rho-b\right)$ and $\mathbb{F}_{3^{6m}} \cong \mathbb{F}_{3^{3m}}[\sigma]/\left(\sigma^2+1\right)$. Then we expand all the operations in Algorithm (2.1) into their equivalent forms but only with operations in the base field $\mathbb{F}_{3^m}$. The comments in Algorithm (2.1) list their numbers; square is counted as one multiplication.

We developed our work only for the Duursma-Lee algorithm in characteristic 3 because, selecting the base field$\mathbb{F}_{3^m}$ accurately, cube root and power can be computed efficiently also in the polynomial basis representation. In the rest of the paper we model the algorithm entirely as a sequence of operations in the base field $\mathbb{F}_{3^m}$, in order to be able to expose the maximum possible degree of parallelism.

3 Scheduling Methodology

In the automated design of application specific circuits, the algorithmic description is commonly partitioned into a data path,i.e., a set of interconnected function units, and a control path,usually a finite state machine or a processor. The former processes data and the latter sends commands to drive the units.Data flow through the data path by steps, called control steps.Each unit is limitedly programmable and can execute one out of a set of similar operation types, and several replicated units may work in parallel. Scheduling the operations of the algorithm so that each control step achieves the best usage of units is therefore a task of great importance to optimize performances.

The algorithm is scheduled to minimize a target set of figures of merit. Here we want to put in parallel as many operations as possible, and we consider the following figures of merit:computation time, circuit area and combinations thereof. Therefore we apply the instruction scheduling methodology with are source-constrained approach [10].

This methodology is quite general, flexible and explores extensively the architectural solution space. It works as follows: the algorithm is modelled as a data dependence graph (DDG), the nodes and arcs of which represent operations and data dependencies among operations,respectively; where necessary, temporary variables store the intermediate results; each node is labelled by a time latency and circuit area cost; scheduling the algorithm means sorting the nodes of the DDG in topological order compatibly with the resource and latency constraints. The following analyses are performed:

1) Assume the number of units of each type is unlimited and the latencies of the units are all identical, then:
   - Compute the As-Soon-As-Possible (ASAP) schedule of the DDG;
   - Compute the As-Late-As-Possible (ALAP) schedule of the DDG.

   From such schedules, determine the maximum number of each unit type,called "variability range", such that by allowing more units of that type further speed-up of the algorithm would not take place any longer.

2) Chosen a parameter to optimize (total computation time, total circuit area or a function thereof), and given a fixed set off unction units (possibly replicated), each labelled with its effective latency, do what follows:
   - Using the List-based Scheduling (LBS) discipline, find the schedule of the algorithm compatible with the resource and latency constraints;
   - For such schedule compute the figure of merit under observation (cost/performance estimate);
   - And additionally compute the minimum number of temporary variables (register allocation).

3) Repeat Step (2) by changing the number of each function unit type, but remaining within the bounds determined in Step (1), valid for the unlimited case.

The entire procedure yields all the LB schedules possible by varying the resources. The LBS discipline uses the latency and number of every function unit type, to construct the schedule by assigning all the operations to each control step at a time. It maintains a ready-set of operations, i.e., those the predecessors of which are already scheduled, assigned to the current control step until the available resources are run out. To determine which of the ready-set operations of the same type to schedule first,that having minimum mobility is selected. Mobility is the control step interval delimited by the ASAP and ALAP schedules constrained by the actual resources and latencies; both schedules are recomputed whenever the resource availability changes. The LBS heuristic discipline is well established and is known for its efficiency in the general case [10].

We have developed (in C language) all the SW tools needed for the above described scheduling methodology. The Duursma-Lee Algorithm (2.1) contains a few initialization operations and a loop repeated for a constant (and relatively high) number of times;most operations are in the loop body, the rest is almost costless.Therefore we schedule only the loop body, and the resulting architecture iterates the computation as many times as the loop does. The loop body is expressed as a list of operations in the base field $\mathbb{F}_{3^m}$, with temporary variables, latency and area specifications; the list is given to the tools, which extract the labelled DDG of the algorithm and process it as explained above.

4 Datapath and Function Units

The HW coprocessor we considered for the Duursma-Lee algorithm is a dedicated parallel architecture, as outlined in Section 3. The data path contains function units for each operation type, and the units may be replicated. The units work at the level of the base field $\mathbb{F}_{3^m}$; hence all data are elements of $\mathbb{F}_{3^m}$. Each function unit has one or two inputs (for unary or binary operations) and one output, and contains buffer registers to store operand(s) and result (hence2 or 3 registers); such buffers decouple the units of one another. A generic unit consumes one clock cycle for loading the operand(s) into the input buffers, one or more cycles for computing (depending on the unit type) and one final cycle for storing the result into the output buffer. A set of registers is also included to store temporary variables. All the registers (buffer and temporary) have the same size as an element of$\mathbb{F}_{3^m}$ (i.e., m digits as in [11, 14], see below).

Function units and temporary registers are connected through dedicated busses and switches, each of size $\mathbb{F}_{3^m}$. The control path drives all the units and registers, and dispatches data via the busses. Such an architecture is consistent with those adopted in [11, 14]. In all the cases we considered, almost all the units and registers happen to have fan-in and fan-out of one or two, and the rest is only slightly over. This limits the number of busses and switches; hence we assume that their time and area costs are negligible.

The Tate pairing function defines a highly arithmetic-intensive primitive. There are several works dedicated to the hardware implementation of efficient arithmetic in Characteristic 3, see [4, 11, 15, 18].In these papers the implementation is done on a FPGA device and the size is evaluated by counting the number of the needed FPGA elements or equivalent gates. To estimate the area and latency costs of each function unit, here we follow a similar approach.

We use the base field $\mathbb{F}_{3^m}$, as in [11, 18], and we represent its elements in polynomial basis, the usual choice. An element of $\mathbb{F}_3$ isa digit of $\{-1, 0, 1\}$, encoded with two bits in such a way that by swapping the bits the sign is inverted. The elementary operations in $\mathbb{F}_3$ (addition, subtraction and multiplication) can be performed in one clock cycle by using two FPGA look-up tables of Type 4: 1 (LUT).

The base field $\mathbb{F}_{3^m}$ is isomorphic to $\mathbb{F}_3[x]/f(x)$, where $f(x)$ is am-degree irreducible reduction polynomial with coefficients in$\mathbb{F}_3$. We use the trinomial type $f(x) = x^m + x^h - 1 \in \mathbb{F}_3[x]$. A generic element of$\mathbb{F}_{3^m}$ is a sequence of m digits in $\mathbb{F}_3$. We need binary and unary function units with inputs $A, B \in \mathbb{F}_3$, limitedly programmable to invert the signs of A or$B$. To do so, we use for each input an array of $2m$ two-way one-bit multiplexers (MX), to swap when requested the operand bit sand thus to invert its sign.

The adder unit must perform the four algebraic additions $\pm A \pm B$. Such operations can be performed digit-wise in one clock cycle,using an array of m adders in $\mathbb{F}_3$ (F3A) [4, 14] plus an array of $4m$ MXs for the signs of the two summands. The multiplier unit must perform the four operations $(\pm A) \times (\pm B)$. The multiplication of two elements in $\mathbb{F}_{3^m}$ is performed in digit-serial way: the former factor is processed in parallel, the latter by groups of D digits in parallel in one clock cycle; hence the total number of cycles is $\lceil m/D \rceil$. Assuming $D \leq m - h$ holds, to simplify reduction, the area cost of the multiplier is the sum of the following contributions: $(m+1)\,D$ F3As; $(m+2)\,D-2$ multipliers in$\mathbb{F}_3$ (F3M); some control logic, namely $4m$ MX; some inner registers for partial products, equivalent to $(6m + 2D - 2)$ flip-flops (FF) [4]; and finally an array of $4m$ MXs for the signs of the two factors, as done in the adder.

Table 1: Costs of arithmetic units in $\mathbb{F}_{3^m}$

Function Unit	**Area**	**Latency**
Adder [4]	m F3A + $4m$ MX	1
Multiplier [4]	$(m+1)\,D$ F3A + $((m+2)D-2)$ F3M + $8m$ MX + FF $(6m+2D-2)$	$\lceil m/D \rceil$
Cube Power [4]	$2m$ F3A + $2m$ MX	1
Cube Root [2]	$(m-1)$ F3A + $2m$ MX	1

The cube power unit must perform the two operations $(\pm A)^3$. In fields of characteristic 3 represented in polynomial basis, this is a linear operation implementable by thinning the coefficients of the base A [18]. The unit takes one clock cycle [4] with an area cost of$2m$ F3As plus $2m$ MXs for the base sign. The cube root unit must perform the two operations $\sqrt[3]{\pm A}$. In the same field type as before, cube root can be computed as in [2], provided the reduction trinomial $f(x)$ satisfies $m, h \equiv 1, 1$ or $m, h \equiv 2, 2 (\mathrm{mod}\ 3)$. A unit specific for $f(x)$ has a latency of one clock cycle and an area equal to $(m-1)$ F3As plus$2m$ MXs for the radicand sign. Finally, all the registers (temporary and buffer) store m digits of $\mathbb{F}_3$, hence they consist of arrays of $2m$ flip-flops;read/write takes one clock cycle. The area and latency formulas of the function units are summarized in Table 1.

Table 2: Costs of arithmetic units in$\mathbb{F}_{3^m}$ including I/O buffers ($m = 97$, $h = 16$ and $D = 4$)

Func. Unit	**Area** (LUT)	**Latency** (clk)	**Var. Range**
Adder	1358	3	18
Multiplier	3908	27	11
Cube Power	1100	3	2
Cube Root	904	3	2

As we want to compare our results with those in the literature, e.g. in [11, 14], we have chosen $m = 97$, $h = 16$and $D = 4$; such values are mathematically and

technologically sound, and fit all the previous constraints. Moreover, since every function unit contains input/output buffers, we have added two more clock cycles to the unit latency, to account for the delay of the read/write operations, and we have increased the unit area to account for the I/O buffers. Since we measure all the areas in terms of FPGA LUTs, we assume the equivalences 1 MX = 1 LUTand 1 FF = $\frac{4}{3}$ LUT, which are somewhat conservative for the FPGAs used for comparison [4, 22]. Westress that our results are rather stable with respect to these hypotheses. In Table 2 the resulting area and latency costs of the various units are shown.

5 Synthesis of Architectures

Here we list the results obtained by scheduling, as described in Section 3, the loop body of the Duursma-Lee Algorithm (2.1). First we consider the ideal case,with unboundedly many function units having identical latency,i.e., one clock cycle; the total numbers of operations are listed in Algorithm (2.1). In such conditions execution takes 18control steps, which is the absolute minimum. Table 2 lists the variability range of each unit type (see Section 3, point Step 1). The adder and multiplier units exhibit a somewhat high variability of 18 and 11, respectively, meaning there is space for parallelization.

Table 3: Optimal T for the loop body

Combinations of Function Units				**T**	**T incr.**
Mul	**Power**	**Add**	**Root**	[clk]	[%]
1	2	1	1	423	386.2
2	2	4	1	234	169.0
3	2	4	1	159	82.7
4	2	4	1	132	51.7
5	1	4	1	108	24.1
6	1	3	1	105	20.7
7, 8	2	3	1	99	13.8
≥ 9	1	4	1	87	min.

Second, we conduct a scheduling campaign with the effective latency and area costs for each unit as in Table 2, optimizing the loop time T, measured in clock cycles, and the loop area-time product AT, measured in equivalent LUTs $\times$ clock cycles (see Section 4), by allowing different combinations of function units. In particular, since the multiplier has the highest time and area costs (Table 2), it is advisable to find the optimal schedules with respect to the selected figure of merit, by allowing from 1up to 11 multipliers (i.e., the variability range, Table 2). This is carried out as follows:

1) Keeping fixed the number of multipliers, compute all the schedules changing the numbers of adders, cube power and root units, with in their variability ranges.

2) For all such schedules, compute the figure of merit under observation (either T or AT) and select the schedule with the minimum value of the figure of interest.

3) Restart from point Item (1), increasing the number of multipliers; terminate when the variability range of the multiplier has been completely explored.

Table 4: Optimal AT for the loop body

Combinations of Function Units				**AT incr.**	**T**
Mul	**Power**	**Add**	**Root**	[%]	[clk]
1	1	1	1	74	426
2	1	1	1	26	246
3	1	2	1	9	168
4	1	2	1	6	141
5	1	4	1	min.	108
6	1	3	1	4	105
7	1	2	1	11	105
8	2	3	1	21	99
9	1	4	1	16	87

In principle there may exist multiple optimal schedules with the same number of multipliers; in practice such an event is unlikely and never occurs in our scheduling campaign.

Table 3 shows the optimal schedules with respect to the computation time T. Each row lists: the numbers of function unit types, the absolute value of T and the increment percentage with respect to the minimum time of 87 clock cycles, reached with 9multipliers; no further speed-up is achieved with more such units. It is evident that with 7 multipliers, the residual time performance gain, achievable with more units, is about 10%.Therefore, in general the time optimal architecture ought not to contain over 7 multipliers.

The register allocation procedure shows that all the schedules in Table 3 require from 28 up to 32 temporary registers. Since register allocation does not depend on the area costs of the function units, we assume to have exactly 32registers, which will suffice for any possible schedule, and weal ways add their area cost to the total area A of the device.

Table 4 shows the optimal schedules with respect to the area-time product AT (where area is estimated as in Table 2 plus the 32 temporary registers). Each row lists: the numbers of function unit types and the percentage increment of AT with respect to the minimum, reached with 5multipliers, and the absolute time T. This is the trade-off for balancing area and time costs, and is in good agreement with the behavior of the AT^2 figure of merit as well (computed but not shown here) [10]. Solutions common to both Tables 3 and 4 are due to chance. Clearly with over 5 multipliers AT must grow again, while T tends to stabilize asymptotically.

6 Evaluation and Comparison

Works dealing with HW architectures for the Tate pairing are [11, 14]. Such authors analyze the modified Duursma-Lee algorithm (D-L) and use function units similar to those explained in Section4, with identical values of m, h and D; hence the loop body iterates exactly $m = 97$ times. In [14] Kerins *et al.* give a parallel FPGA architecture, with units working in the extension field$\mathbb{F}_{3^{6m}}$ (and presumably with temporary registers,though nothing is said). To synthesize such high-level units, they use 18 multipliers and 6 cube power units in$\mathbb{F}_{3^m}$, plus over 100 adders; their architecture computes the entire D-L algorithm in $8,924$ clock cycles.Instead, in [11] Grabher *et al.* present a FPGA co-processor for arithmetic in $\mathbb{F}_{3^m}$, which contains one unit per type, has 32 temporary registers (as we do) but works serially. Higher-level operations are compiled on a general purpose processor. The reported running time of the entire D-L algorithm is of $59,946$ clock cycles.

For the complete D-L algorithm, we obtain a minimum computation time of $87m = 87 \times 97 = 8,439$ clock cycles with 9 multiplier sand 4 adders (see Table 3); such a time is comparable with that of [14], but our architecture is by far smaller. In the case of minimum area-time product AT, a number of5 multipliers and 4 adders yields a total D-L time of $108m = 108 \times 97 = 10,476$ clock cycles (see Table 4),still comparable to the minimum time of [14] but with an even smaller area. In both cases, our total D-L time is about one order of magnitude shorter than that of [11]. In summary, scheduling seems to be convenient and the performance leap is stable against the assumptions we made.

Moreover, we determined also the optimal time schedule with a single function unit per type, which yields a time cost of 426 clock cycles for the loop body (see Table 4). The total time to compute the whole D-L algorithm is then of $426m = 426 \times 97 = 41,322$ clock cycles, which represents a speed-up of 31% with respect to the time of $59,946$ clock cycles reported in [11]. Such a speed-up is due to the adoption of an automated and exhaustive scheduling methodology.

7 Conclusions

An analysis of the computation of the Tate pairing with the Duursma-Lee algorithm, by means of a parallel dedicated HW co-processor in$\mathbb{F}_{3^m}$, has been presented. Results show that the algorithm is well suited for parallelism and that the cost-performance tradeoff can be considerably improved with respect to literature solutions. Future research directions include the analysis of other Tate pairing algorithms as well as of different architecture models, e.g. changing m and D, or for instance an architecture with a limited number of busses (two or three) or a pipelined structure.

References

[1] P. Barreto, H. Kim, B. Lynn, and M. Scott, "Efficient algorithms for pairing-based cryptosystems," in *Advances in Cryptology (Crypto'02)*, LNCS 2442, pp. 354-368, 2002.

[2] P. Barreto, "A note on efficient computation of cube roots in characteristic 3," *Cryptology ePrint Archive*, 2004. (http://eprint.iacr.org/2004/305.pdf)

[3] P. Barreto, S. Galbraith, C. Eigeartaigh, and M. Scott, "Efficient pairing computation on supersingular abelian varieties," *Cryptology ePrint Archive*, 2004. (http://eprint.iacr.org/2004/375.pdf)

[4] G. Bertoni, J. Guajardo, S. Kumar, G. Orlando, C. Paar, and T. Wollinger,"Efficient GF(p^m) arithmetic architectures for cryptographic applications," in *Topics in Cryptology (CT RSA'03)*, LNCS 2612, pp. 158-175, 2003.

[5] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Advances in Cryptology (Crypto'01)*, LNCS 2139, pp. 213-229, 2001.

[6] C. Cocks, "An identity-based encryption scheme based on quadratic residues," *Cryptography and Coding*, LNCS 2260, pp. 360-363, 2001.

[7] R. Dutta, R. Barua, and P. Sarkar, "Pairing-based cryptography: A survey," *Cryptology ePrint Archive*, 2004. (http://eprint.iacr.org/2004/064.pdf)

[8] I. Duursma and H. Lee, "Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$," in *Advances in Cryptology (Asiacrypt'03)*, LNCS 2894, pp. 111-123, 2003.

[9] G. Frey, M. Muller, and H. Ruck, "The tate pairing and the discrete logarithm applied to elliptic curve cryptosystems," *IEEE Transactions on Information Theory*, vol. 45, no. 5, pp. 1717-718, 1999.

[10] D. Gajski, N. Dutt, and A. Wu, *High-level Synthesis - Introduction to Chip and System Design*, 2nd Print, Kluwer Academic Publisher, 1993.

[11] P. Grabher and D. Page, "Hardware acceleration of the tate pairing in characteristic 3," in *Cryptographic Hardware and Embedded Systems (CHES'05)*, LNCS 3659, pp. 398-411, 2005.

[12] R. Granger, D. Page, and M. Stam, "On small characteristic algebraic tori in pairing-based cryptography," *Cryptology ePrint Archive*, 2004. (http://eprint.iacr.org/2004/132)

[13] A. Joux, "The weil and tate Pairings as building blocks for public key cryptosystems," in *Proceedings of the 5th International Symposium on Algorithmic Number Theory (ANTS-V)*, LNCS 2369, pp. 20-32, 2002.

[14] T. Kerins, W. Marnane, E. Popovici, and P. Barreto, "Efficient hardware for the tate pairing calculation in characteristic 3," in *Cryptographic Hardware and Embedded Systems (CHES'05)*, LNCS 3659, pp. 412-426, 2005.

[15] T. Kerins, E. Popovici, and W. Marnane, "Algorithms and architectures for use in FPGA implementations of identity based encryption schemes,"

in *Field Programmable Logic and Application (FPL;04)*, LNCS 3203, pp. 74-83, 2004.
[16] S. Kwon, "Efficient tate pairing computation for supersingular elliptic curves over binary fields," *Cryptology ePrint Archive*, 2004. (http://eprint.iacr.org/2004/303)
[17] V. Miller, *Short Programs for Functions on Curves*, unpublished manuscript, 1986.
[18] D. Page and N. Smart, "Hardware implementation of finite fields of characteristic 4," in *Cryptographic Hardware and Embedded Systems (CHES'02)*, LNCS 2523, pp. 529-539, 2002.
[19] R. Sakai, K. Ohgishi, and M. Kasahara, "Cryptosystems based on pairing," in *Symposium on Cryptography and Information Security (SCIS2000)*, pp. 26-28, 2000.
[20] A. Shamir, "Identity-based cryptosystems and signature schemes," *Advances in Cryptology (Crypto'85)*, LNCS 196, pp. 47-53, 1985.
[21] E. Verheul, "Evidence that XTR is more secure than supersingular elliptic curve cryptosystems," in *Advances in Cryptology (Eurocrypt'01)*, LNCS 2045, pp. 195-210, 2001.
[22] XILINX, *Gate Count Capacity Metrics for FPGAs*, Application Note, XAPP 059 (Ver. 1.1), Feb. 1, 1997. (http://www.origin.xilinx.com/bvdocs/appnotes/xapp059.pdf)

Guido Bertoni received the Dr. Eng degree in computer engineering and the Ph.D degree from Politecnico di Milano in 1999 and 2004 respectively. He joins ST in fall 2003 as researcher in the field of cryptography. His research interests include the cryptographic algorithms, hardware and software implementations, and problems related to side channels attack. He teaches cryptography at Politecnico di Milano as contract professor.

Luca Breveglieri received both the M.Sc. degree in electronic engineering and the D.Sc. degree in electronic engineering of information technology and systems from the Politecnico di Milano, Italy, in 1986 and 1992, respectively. From 1991 to 1998, he was a computer technician and a part-time researcher. Since 1998 he has been an associate professor of computer science at the Politecnico di Milano. He has over 60 publications in refereed journals and conferences. His current research interests include architectures of computing systems, application specific VLSI synthesis, computer arithmetic and applied cryptography, and automata and formal languages theory.

Pasqualina Fragneto received the Laurea in mathematics from the University Federico II, Napoli, Italy in 1998. Since October 1999, she has been with the AST group of the STMicroelectronics. Her research interests includes algorithmic number theory and computer arithmetic, efficient implementation of cryptographic systems.

Gerardo Pelosi received his M.S. degree in Telecommunication engineering from the Politecnico di Milano, Italy in February 2003. He is currently towards his PhD degree in Information Technology at Politecnico di Milano. His primary research areas include cryptographic algorithms and efficient hardware and software implementations.