Fast architectures for the η_T pairing over small-characteristic supersingular elliptic curves

Jean-Luc Beuchat, Jérémie Detrey, Nicolas Estibals, Eiji Okamoto, Francisco Rodríguez-Henríquez

Fast Architectures for the η_T Pairing over Small-Characteristic Supersingular Elliptic Curves

Jean-Luc Beuchat, Jérémie Detrey, Nicolas Estibals, Eiji Okamoto, and Francisco Rodríguez-Henríquez

Abstract— **This paper is devoted to the design of fast parallel accelerators for the cryptographic η_T pairing on supersingular elliptic curves over finite fields of characteristics two and three. We propose here a novel hardware implementation of Miller's algorithm based on a parallel pipelined Karatsuba multiplier. After a short description of the strategies we considered to design our multiplier, we point out the intrinsic parallelism of Miller's loop and outline the architecture of coprocessors for the η_T pairing over $\mathbf{F}_{2^m}$ and $\mathbf{F}_{3^m}$. Thanks to a careful choice of algorithms for the tower field arithmetic associated with the η_T pairing, we manage to keep the pipelined multiplier at the heart of each coprocessor busy. A final exponentiation is still required to obtain a unique value, which is desirable in most cryptographic protocols. We supplement our pairing accelerators with a coprocessor responsible for this task. An improved exponentiation algorithm allows us to save hardware resources.**

According to our place-and-route results on Xilinx FPGAs, our designs improve both the computation time and the area–time trade-off compared to previously published coprocessors.

Keywords: Tate pairing, η_T pairing, elliptic curve, finite field arithmetic, Karatsuba multiplier, hardware accelerator, FPGA.

I. Introduction

In 2000, Mitsunari, Sakai & Kasahara [36], Sakai, Oghishi & Kasahara [41], and Joux [25] independently showed how to use bilinear pairings defined over algebraic curves to solve cryptographic problems of long standing. This discovery ignited an intensive research that, until today, has produced an impressive number of pairing-based cryptographic protocol proposals [13]. Practice has shown that one of the most efficient options to compute bilinear pairings is to resort to the Tate pairing operating on supersingular elliptic curves of low embedding degrees.

Back in 1986, Miller [33], [34] presented an iterative algorithm that can be adapted to compute the Tate pairing with linear complexity with respect to the size of the input. Since then, significant improvements of Miller's algorithm were independently proposed in 2002 by Barreto *et al.* [4] and Galbraith *et al.* [17]. One year later, Duursma & Lee presented a radix-3 variant of Miller's algorithm especially targeted at the case of characteristic three [14]. In 2004, Barreto *et al.* [3] introduced the η_T approach, which further shortens the loop of Miller's algorithm. More recently, Hess, Smart, and Vercauteren generalized these results to ordinary curves [22], [23], [46].

We extend here the work presented in [10] and propose novel hardware architectures for computing the η_T pairing over binary and ternary fields based on parallel pipelined Karatsuba multipliers and enhanced unified arithmetic operators. We stress that the modified Tate pairing can be directly computed from the reduced η_T pairing at almost no extra cost [7]. Our hardware accelerators are able to compute the η_T pairing operating on supersingular elliptic curves defined over $\mathbf{F}_{2^{691}}$ and $\mathbf{F}_{3^{313}}$ in just 18.8 μs and 16.9 μs, respectively (Table V). We note that these field sizes enjoy an associated security equivalent to that of 105-bit and 109-bit symmetric-key cryptosystems, respectively (Table IV).

The main strategies considered to design our parallel pipelined multiplier are described in Section II. They are included in a VHDL code generator that allows us to experiment on a wide range of operators as well as a variety of design parameters. Thanks to a judicious choice of algorithms for performing tower field arithmetic and a careful analysis of the scheduling, we managed to keep our pipelined units always busy. This allows us to compute one iteration of Miller's algorithm over ternary and binary fields in only 17 and 7 clock cycles, respectively (Sections III and IV). We summarize the results obtained from our FPGA implementation and provide the reader with a thorough comparison against previously published coprocessors in Section V.

For the sake of concision, we are forced to skip the description of many important concepts of elliptic curve theory. We suggest the interested reader to review [44], [47] for an in-depth coverage of this topic.

II. Parallel Karatsuba Multipliers

Before delving into the specifics of our pairing coprocessor architectures, we first detail here the Karatsuba multipliers on which they extensively rely.

We define the p-ary extension field $\mathbf{F}_{p^m}$ as $\mathbf{F}_p[x]/(f(x))$, where f is an irreducible degree-m monic polynomial over $\mathbf{F}_p$. The product of two arbitrary elements of $\mathbf{F}_{p^m}$ represented as p-ary polynomials of degree at most $m-1$ is computed as the polynomial multiplication of the two elements modulo f. Carefully selecting an irreducible polynomial with low Hamming weight (*i.e.* trinomial, tetranomial, *etc.*) and low sub-degree allows for a simple modular reduction step.

In this work, due to its subquadratic space complexity, we opted for a variant of the classical Karatsuba multiplier to

J.-L. Beuchat and E. Okamoto are with the Graduate School of Systems and Information Engineering, University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki, 305-8573, Japan.

J. Detrey and N. Estibals are with the CARAMEL project-team, INRIA Nancy – Grand Est, Bâtiment A, 615, rue du Jardin Botanique, 54602 Villers-lès-Nancy Cédex, France.

F. Rodríguez-Henríquez is with the Computer Science Department, Centro de Investigación y de Estudios Avanzados del IPN, Av. Instituto Politécnico Nacional No. 2508, 07300 México City, México.

implement the polynomial product, while a few extra adders and subtracters over $\mathbf{F}_p$ are dedicated to performing the final reduction modulo f.

A. Variations on the Karatsuba Algorithm

The Karatsuba multiplication [27] is based on the observation that the polynomial product $c = a \cdot b$, for a and $b \in \mathbf{F}_{p^m}$, can be computed as

$$c = a^L b^L + \left[(a^H + a^L)(b^L + b^H) - (a^H b^H + a^L b^L)\right] x^n + a^H b^H x^{2n},$$

where $n = \lceil \frac{m}{2} \rceil$, $a = a^L + x^n a^H$, and $b = b^L + x^n b^H$.

Note that since we are working with polynomials, there is no carry propagation. This allows one to split the operands in a slightly different way: for instance Hanrot and Zimmermann [21] suggested to split them into odd- and even-degree parts. It was adapted to multiplication over $\mathbf{F}_{2^m}$ by Fan *et al.* [15]. Since there is no overlap between the odd and even parts at the reconstruction step, this different method of splitting saves approximately m additions over $\mathbf{F}_p$ during the reconstruction of the product.

Another natural way to generalize the Karatsuba multiplication is to split the operands into three or more parts, in a classical way (*i.e.* splitting each operand into contiguous parts from the lowest to the highest powers of x) or using a generalized odd/even split (*i.e.* according to the degree modulo the number of split parts). By applying this strategy recursively, in each iteration each polynomial multiplication is transformed into three or more subproducts of smaller degree, until all the polynomial operands are reduced to single coefficients. Nevertheless, practice has shown that it is better to prune the recursion earlier, performing the lowest-level multiplications using alternative techniques that are more compact and/or faster for low-degree operands, such as the so-called schoolbook method with quadratic complexity, which has been selected for this work.

B. A Pipelined Architecture for the Karatsuba Multiplier

We pipelined our multiplier architecture by means of optional registers inserted between the computations of the required subproducts, where the depth of the pipeline can be adjusted according to the complexity of the application at hand. This approach allows us to split the critical path of the whole multiplier structure and therefore increase its operating frequency.

In order to study a wide range of implementation strategies, we wrote a VHDL code generator, which automatically produces the description of different variants of Karatsuba multipliers according to several parameters (field extension degree, irreducible polynomial, splitting method, *etc.*). Our automatic tool was extremely useful for selecting the operator that showed the highest clock frequency, the smallest area or a good trade-off between them.

III. Reduced η_T Pairing in Characteristic Three

In the following, we consider the computation of the reduced η_T pairing in characteristic three. Table I summarizes the parameters of the algorithm and of the supersingular curves. We refer the reader to [3], [8] for more details about the computation of the η_T pairing. Recall that a final exponentiation is required to obtain a unique value, which is desirable in the context of cryptographic protocols. As pointed out by Beuchat *et al.* [9], the computations of the non-reduced pairing (*i.e.* Miller's algorithm) and of the final exponentiation do not share the same datapath, and it seems judicious to pipeline these two tasks using two distinct coprocessors in order to reduce the computation time and increase the throughput.

A. Computation of Miller's Algorithm

We rewrote in Algorithm 1 the reversed-loop algorithm in characteristic three described in [8], denoting each iteration with parenthesized indices in superscript in order to emphasize the intrinsic parallelism of the η_T pairing. At each iteration of Miller's algorithm, two tasks are performed in parallel, namely: a sparse multiplication over $\mathbf{F}_{3^{6m}}$ (lines 6 and 7), and the computation of the coefficients for the next sparse operation (lines 8 to 10). We say that an operand in $\mathbf{F}_{3^{6m}}$ is sparse when some of its coefficients are trivial (*i.e.* either zero, one, or minus one).

Algorithm 1 Computation of the reduced η_T pairing in characteristic three.[†]

```
Input: $P = (x_P, y_P)$ and $Q = (x_Q, y_Q) \in E(\mathbf{F}_{3^m})[\ell]$.
Output: $\eta_T(P, Q)^M \in \mathbf{F}^*_{3^{6m}}$.
1.  $x_P^{(0)} \leftarrow x_p - \nu b$;  $y_P^{(0)} \leftarrow -\mu b y_P$;
2.  $x_Q^{(0)} \leftarrow x_Q$;  $y_Q^{(0)} \leftarrow -\lambda y_Q$;
3.  $t^{(0)} \leftarrow x_P^{(0)} + x_Q^{(0)}$;
4.  $R^{(-1)} \leftarrow \lambda y_P^{(0)} \cdot t^{(0)} - \lambda y_Q^{(0)} \sigma - \lambda y_P^{(0)} \rho$;
5.  for $i = 0$ to $(m-1)/2$ do
6.      $S^{(i)} \leftarrow -\left(t^{(i)}\right)^2 + y_P^{(i)} y_Q^{(i)} \sigma - t^{(i)} \rho - \rho^2$;
7.      $R^{(i)} \leftarrow R^{(i-1)} \cdot S^{(i)}$;
8.      $x_P^{(i+1)} \leftarrow \sqrt[3]{x_P^{(i)}}$;  $y_P^{(i+1)} \leftarrow \sqrt[3]{y_P^{(i)}}$;
9.      $x_Q^{(i+1)} \leftarrow \left(x_Q^{(i)}\right)^3$;  $y_Q^{(i+1)} \leftarrow \left(y_Q^{(i)}\right)^3$;
10.     $t^{(i+1)} \leftarrow x_P^{(i)} + x_Q^{(i)}$;
11. end for
12. return $\left(R^{((m-1)/2)}\right)^M$;
```

[†] Intermediate variables in uppercase belong to $\mathbf{F}_{3^{6m}}$, those in lowercase to $\mathbf{F}_{3^m}$.

1) Sparse Multiplication over $\mathbf{F}_{3^{6m}}$*:* The intermediate result $R^{(i-1)}$ is multiplied by the sparse operand $S^{(i)}$ (Algorithm 1, lines 6 and 7). This operation is easier than a standard multiplication over $\mathbf{F}_{3^{6m}}$, but the choice of the sparse multiplication algorithm requires careful attention. Bertoni *et al.* [6] and Gorla *et al.* [18] took advantage of Karatsuba multiplication and Lagrange interpolation, respectively, to reduce the number of multiplications over $\mathbf{F}_{3^m}$ at the expense

TABLE I
SUPERSINGULAR CURVES OVER $\mathbf{F}_{3^m}$ AND $\mathbf{F}_{2^m}$.

	Characteristic three	**Characteristic two**
Base field	$\mathbf{F}_{3^m}$, where m is coprime to 6.	$\mathbf{F}_{2^m}$, where m is an odd integer.
Curve equation	$y^2 = x^3 - x + b$, with $b \in \{-1, 1\}$.	$y^2 + y = x^3 + x + b$, with $b \in \{0, 1\}$.
Number of rational points	$N = 3^m + 1 + \mu b 3^{(m+1)/2}$, with $\mu = \begin{cases} +1 & \text{if } m \equiv 1,\ 11 \pmod{12}, \text{ and} \\ -1 & \text{if } m \equiv 5,\ \ 7 \pmod{12}. \end{cases}$	$N = 2^m + 1 + \nu 2^{(m+1)/2}$, with $\delta = \begin{cases} b & \text{if } m \equiv 1,\ 7 \pmod{8}, \\ 1 - b & \text{if } m \equiv 3,\ 5 \pmod{8}, \end{cases}$ and $\nu = (-1)^{\delta}$.
Embedding degree	$k = 6$	$k = 4$
Distortion map	$\psi : E(\mathbf{F}_{3^m})[\ell] \longrightarrow E(\mathbf{F}_{3^{6m}})[\ell] \setminus E(\mathbf{F}_{3^m})[\ell]$ $(x, y) \longmapsto (\rho - x, y\sigma)$ with $\sigma \in \mathbf{F}_{3^2}$ satisfying $\sigma^2 = -1$, and $\rho \in \mathbf{F}_{3^3}$ satisfying $\rho^3 = \rho + b$.	$\psi : E(\mathbf{F}_{2^m})[\ell] \longrightarrow E(\mathbf{F}_{2^{4m}})[\ell] \setminus E(\mathbf{F}_{2^m})[\ell]$ $(x, y) \longmapsto (x + s^2, y + sx + t)$ with $s \in \mathbf{F}_{2^2}$ satisfying $s^2 = s + 1$, and $t \in \mathbf{F}_{2^4}$ satisfying $t^2 = t + s$.
Tower field	$\mathbf{F}_{3^{6m}} = \mathbf{F}_{3^m}[\sigma, \rho]$ $\cong \mathbf{F}_{3^m}[X, Y]/(X^2 + 1, Y^3 - Y - b)$	$\mathbf{F}_{2^{4m}} = \mathbf{F}_{2^m}[s, t]$ $\cong \mathbf{F}_{2^m}[X, Y]/(X^2 + X + 1, Y^2 + Y + X)$
Final exponentiation	$M = \left(3^{3m} - 1\right) \cdot \left(3^m + 1\right) \cdot$ $\left(3^m + 1 - \mu b 3^{(m+1)/2}\right)$	$M = \left(2^{2m} - 1\right) \cdot \left(2^m + 1 - \nu 2^{(m+1)/2}\right)$
Parameters of Algorithms 1 and 3	$\lambda = \begin{cases} +1 & \text{if } m \equiv 7,\ 11 \pmod{12}, \\ -1 & \text{if } m \equiv 1,\ \ 5 \pmod{12}, \end{cases}$ $\nu = \begin{cases} +1 & \text{if } m \equiv 5,\ 11 \pmod{12}, \text{ and} \\ -1 & \text{if } m \equiv 1,\ \ 7 \pmod{12}. \end{cases}$	$\alpha = \begin{cases} 0 & \text{if } m \equiv 3 \pmod{4}, \\ 1 & \text{if } m \equiv 1 \pmod{4}, \end{cases}$ $\beta = \begin{cases} b & \text{if } m \equiv 1,\ 3 \pmod{8}, \text{ and} \\ 1 - b & \text{if } m \equiv 5,\ 7 \pmod{8}. \end{cases}$

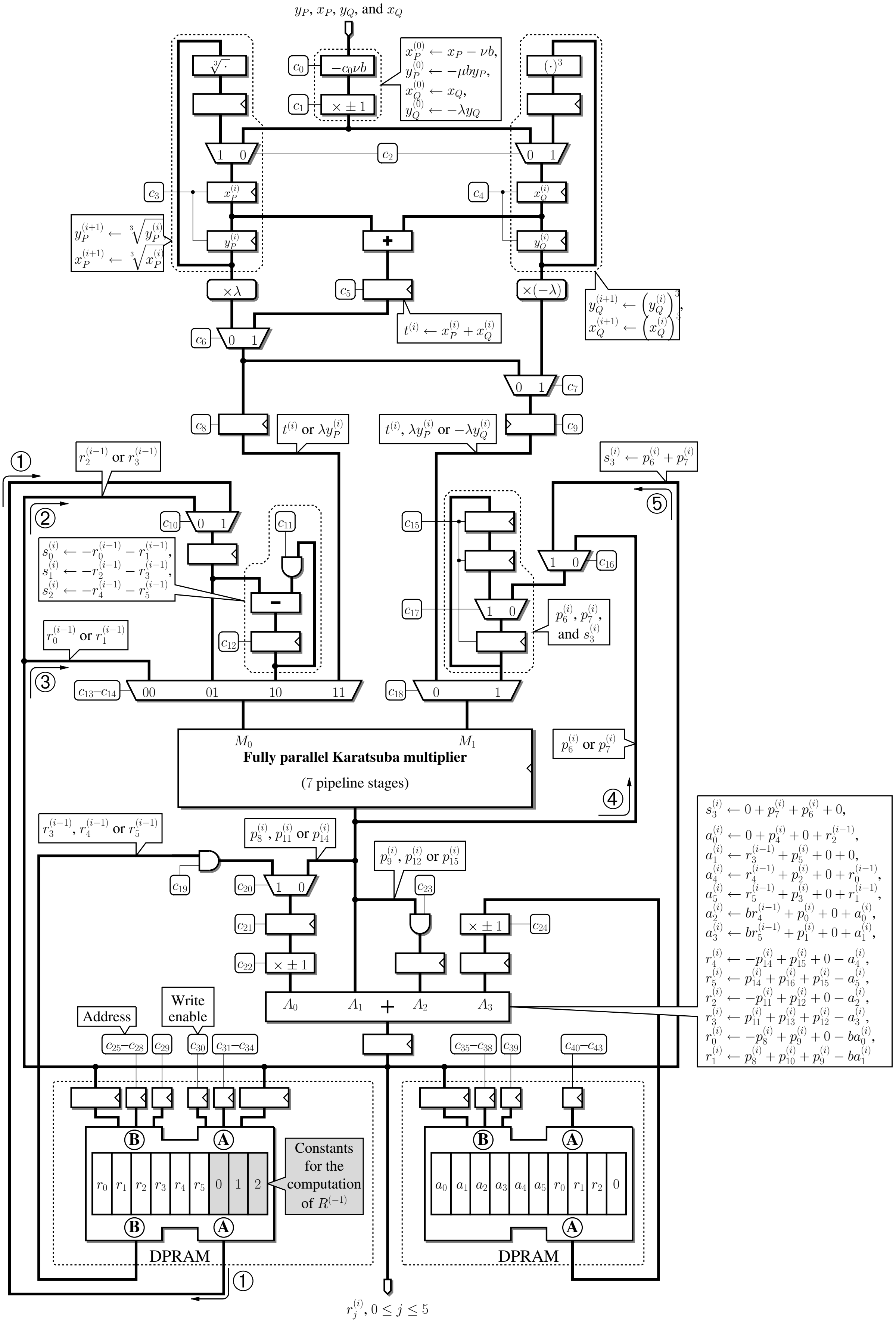
of several additions. (Note that Gorla *et al.* study standard multiplication over $\mathbf{F}_{3^{6m}}$ in [18], but extending their approach to sparse multiplication is straightforward.) In order to keep the pipeline of a Karatsuba multiplier busy, we would have to embed in our processor a large multioperand adder (up to twelve operands for the scheme proposed by Gorla *et al.*) and several multiplexers to deal with the irregular datapath. This would negatively impact the area and the clock frequency, and we prefer considering the algorithm discussed by Beuchat *et al.* in [11] which gives a better trade-off between the number of multiplications and additions over the underlying field (Algorithm 2): it involves 17 multiplications and 29 additions over $\mathbf{F}_{3^m}$ to compute $S^{(i)}$ and $R^{(i-1)} \cdot S^{(i)}$.

We suggest to take advantage of a parallel Karatsuba multiplier with seven pipeline stages to implement Miller's algorithm. Since the algorithm we selected for sparse multiplication over $\mathbf{F}_{3^{6m}}$ requires at most the addition of four elements of $\mathbf{F}_{3^m}$, it suffices to complement the multiplier with a four-operand adder to compute $s_3^{(i)}$, $a_j^{(i)}$, and $r_j^{(i)}$, $0 \le j \le 5$, as shown in Figure 1. The second loop of Algorithm 2 (lines 13 to 16) requires a small amount of additional hardware. Since the first two multiplications of the loop involve $r_{2j}^{(i-1)}$ and $r_{2j+1}^{(i-1)}$, respectively, we compute $s_j^{(i)}$ on-the-fly by means of an accumulator. Furthermore, it seems convenient to store $p_6^{(i)}$, $p_7^{(i)}$, and $s_3^{(i)}$ in a circular shift register.

We managed to find a scheduling that allows us to start a new multiplication over $\mathbf{F}_{3^m}$ at each clock cycle, thus keeping the pipeline busy and computing an iteration of Miller's algorithm in 17 clock cycles as depicted in Figure 2. It is worth noticing that the cost of additions over $\mathbf{F}_{3^m}$ is hidden and the number of clock cycles depends only on the amount of multiplications over $\mathbf{F}_{3^m}$. We easily identify five datapaths (denoted by the numerals ① to ⑤ in Figures 1 and 2) between the output of the four-operand adder and the inputs of the parallel multiplier.

Specific attention is needed to design the register file storing the coefficients of $R^{(i)}$ and the intermediate variables $a_j^{(i)}$, $0 \le j \le 6$, of the sparse multiplication algorithm. According to our scheduling scheme, we have to read simultaneously up to three variables from the register file. Thus, we decided to implement it by means of two blocks of Dual-Ported RAM (DPRAM):

- The first one is connected to input M_0 of the parallel multiplier and input A_0 of the four-operand adder, and

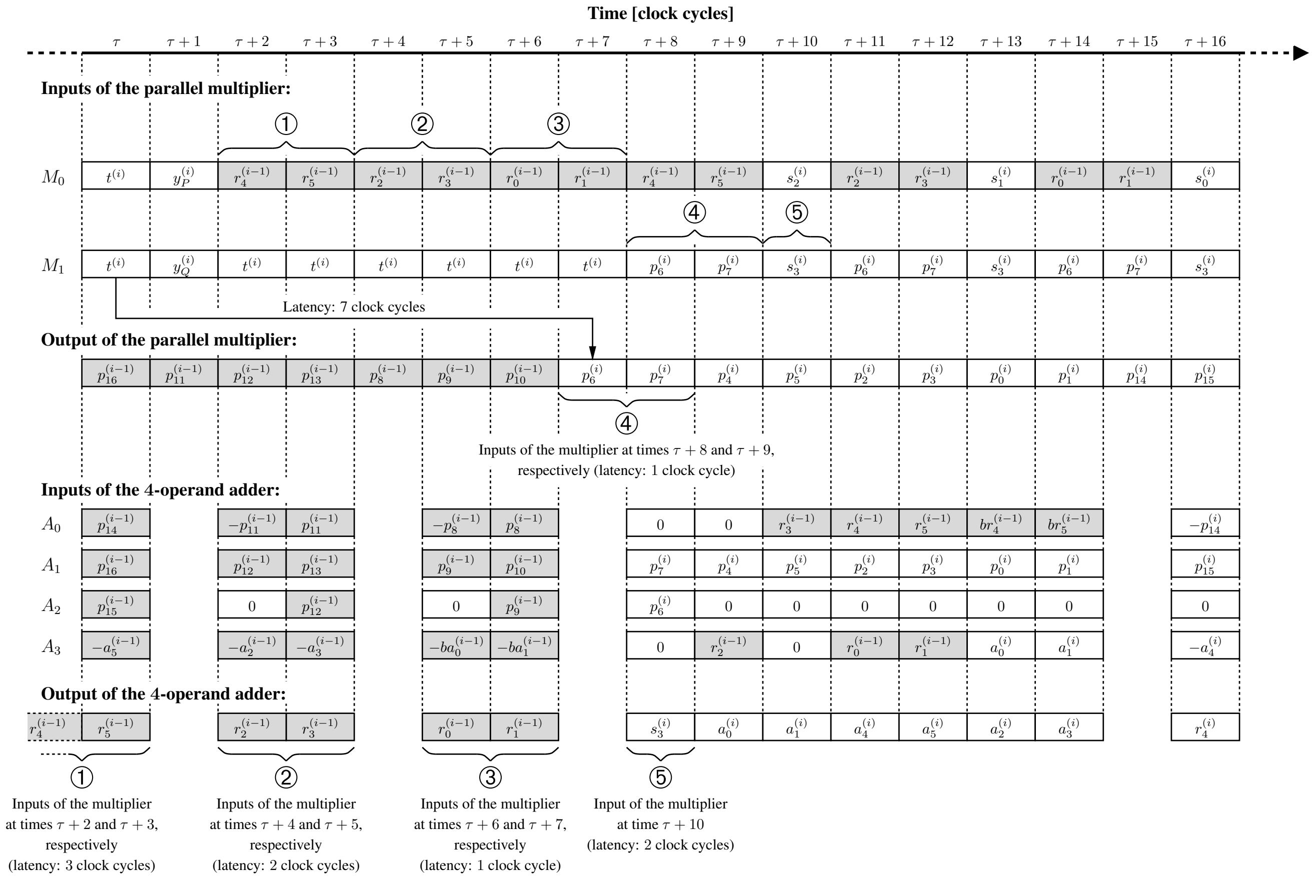

y_P, x_P, y_Q, and x_Q
c_0
$-c_0 \nu b$
c_1
$\times \pm 1$
$x_P^{(0)} \leftarrow x_P - \nu b$,
$y_P^{(0)} \leftarrow -\mu b y_P$,
$x_Q^{(0)} \leftarrow x_Q$,
$y_Q^{(0)} \leftarrow -\lambda y_Q$
$\sqrt[3]{\cdot}$
$(\cdot)^3$
1 0
c_2
0 1
c_3
$x_P^{(i)}$
c_4
$x_Q^{(i)}$
$y_P^{(i+1)} \leftarrow \sqrt[3]{y_P^{(i)}}$
$x_P^{(i+1)} \leftarrow \sqrt[3]{x_P^{(i)}}$
$y_P^{(i)}$
+
$y_Q^{(i)}$
$\times \lambda$
c_5
$\times(-\lambda)$
$y_Q^{(i+1)} \leftarrow \left(y_Q^{(i)}\right)^3$,
$x_Q^{(i+1)} \leftarrow \left(x_Q^{(i)}\right)^3$
c_6
0 1
$t^{(i)} \leftarrow x_P^{(i)} + x_Q^{(i)}$
0 1
c_7
c_8
$t^{(i)}$ or $\lambda y_P^{(i)}$
$t^{(i)}$, $\lambda y_P^{(i)}$ or $-\lambda y_Q^{(i)}$
c_9
1
$r_2^{(i-1)}$ or $r_3^{(i-1)}$
$s_3^{(i)} \leftarrow p_6^{(i)} + p_7^{(i)}$
2
c_{10}
0 1
c_{11}
c_{15}
5
$s_0^{(i)} \leftarrow -r_0^{(i-1)} - r_1^{(i-1)}$,
$s_1^{(i)} \leftarrow -r_2^{(i-1)} - r_3^{(i-1)}$,
$s_2^{(i)} \leftarrow -r_4^{(i-1)} - r_5^{(i-1)}$
−
1 0
c_{16}
c_{17}
1 0
$p_6^{(i)}$, $p_7^{(i)}$, and $s_3^{(i)}$
$r_0^{(i-1)}$ or $r_1^{(i-1)}$
c_{12}
3
c_{13}–c_{14}
00
01
10
11
c_{18}
0
1
M_0
M_1
Fully parallel Karatsuba multiplier
(7 pipeline stages)
$p_6^{(i)}$ or $p_7^{(i)}$
4
$r_3^{(i-1)}$, $r_4^{(i-1)}$ or $r_5^{(i-1)}$
$p_8^{(i)}$, $p_{11}^{(i)}$ or $p_{14}^{(i)}$
$p_9^{(i)}$, $p_{12}^{(i)}$ or $p_{15}^{(i)}$
c_{19}
c_{20}
1 0
c_{23}
c_{21}
$\times \pm 1$
c_{24}
c_{22}
$\times \pm 1$
A_0
A_1
+
A_2
A_3
Address
Write enable
c_{25}–c_{28}
c_{29}
c_{30}
c_{31}–c_{34}
c_{35}–c_{38}
c_{39}
c_{40}–c_{43}
B
A
r_0 r_1 r_2 r_3 r_4 r_5 0 1 2
Constants for the computation of $R^{(-1)}$
B
A
DPRAM
1
B
A
a_0 a_1 a_2 a_3 a_4 a_5 r_0 r_1 r_2 0
B
A
DPRAM
$r_j^{(i)}$, $0 \leq j \leq 5$

$s_3^{(i)} \leftarrow 0 + p_7^{(i)} + p_6^{(i)} + 0$,

$a_0^{(i)} \leftarrow 0 + p_4^{(i)} + 0 + r_2^{(i-1)}$,
$a_1^{(i)} \leftarrow r_3^{(i-1)} + p_5^{(i)} + 0 + 0$,
$a_4^{(i)} \leftarrow r_4^{(i-1)} + p_2^{(i)} + 0 + r_0^{(i-1)}$,
$a_5^{(i)} \leftarrow r_5^{(i-1)} + p_3^{(i)} + 0 + r_1^{(i-1)}$,
$a_2^{(i)} \leftarrow br_4^{(i-1)} + p_0^{(i)} + 0 + a_0^{(i)}$,
$a_3^{(i)} \leftarrow br_5^{(i-1)} + p_1^{(i)} + 0 + a_1^{(i)}$,

$r_4^{(i)} \leftarrow -p_{14}^{(i)} + p_{15}^{(i)} + 0 - a_4^{(i)}$,
$r_5^{(i)} \leftarrow p_{14}^{(i)} + p_{16}^{(i)} + p_{15}^{(i)} - a_5^{(i)}$,
$r_2^{(i)} \leftarrow -p_{11}^{(i)} + p_{12}^{(i)} + 0 - a_2^{(i)}$,
$r_3^{(i)} \leftarrow p_{11}^{(i)} + p_{13}^{(i)} + p_{12}^{(i)} - a_3^{(i)}$,
$r_0^{(i)} \leftarrow -p_8^{(i)} + p_9^{(i)} + 0 - ba_0^{(i)}$,
$r_1^{(i)} \leftarrow p_8^{(i)} + p_{10}^{(i)} + p_9^{(i)} - ba_1^{(i)}$

Fig. 1. Coprocessor for the η_T pairing in characteristic three. (N.B. All control bits c_i belong to $\{0, 1\}$.)


Time [clock cycles]
Inputs of the parallel multiplier:
①
②
③
④
⑤
Latency: 7 clock cycles
Output of the parallel multiplier:
④
Inputs of the multiplier at times $\tau+8$ and $\tau+9$, respectively (latency: 1 clock cycle)
Inputs of the 4-operand adder:
Output of the 4-operand adder:
①
Inputs of the multiplier at times $\tau+2$ and $\tau+3$, respectively (latency: 3 clock cycles)
②
Inputs of the multiplier at times $\tau+4$ and $\tau+5$, respectively (latency: 2 clock cycles)
③
Inputs of the multiplier at times $\tau+6$ and $\tau+7$, respectively (latency: 1 clock cycle)
⑤
Input of the multiplier at time $\tau+10$ (latency: 2 clock cycles)

		τ	$\tau+1$	$\tau+2$	$\tau+3$	$\tau+4$	$\tau+5$	$\tau+6$	$\tau+7$	$\tau+8$	$\tau+9$	$\tau+10$	$\tau+11$	$\tau+12$	$\tau+13$	$\tau+14$	$\tau+15$	$\tau+16$
M_0		$t^{(i)}$	$y_P^{(i)}$	$r_4^{(i-1)}$	$r_5^{(i-1)}$	$r_2^{(i-1)}$	$r_3^{(i-1)}$	$r_0^{(i-1)}$	$r_1^{(i-1)}$	$r_4^{(i-1)}$	$r_5^{(i-1)}$	$s_2^{(i)}$	$r_2^{(i-1)}$	$r_3^{(i-1)}$	$s_1^{(i)}$	$r_0^{(i-1)}$	$r_1^{(i-1)}$	$s_0^{(i)}$
M_1		$t^{(i)}$	$y_Q^{(i)}$	$t^{(i)}$	$t^{(i)}$	$t^{(i)}$	$t^{(i)}$	$t^{(i)}$	$t^{(i)}$	$p_6^{(i)}$	$p_7^{(i)}$	$s_3^{(i)}$	$p_6^{(i)}$	$p_7^{(i)}$	$s_3^{(i)}$	$p_6^{(i)}$	$p_7^{(i)}$	$s_3^{(i)}$
Output of the parallel multiplier		$p_{16}^{(i-1)}$	$p_{11}^{(i-1)}$	$p_{12}^{(i-1)}$	$p_{13}^{(i-1)}$	$p_8^{(i-1)}$	$p_9^{(i-1)}$	$p_{10}^{(i-1)}$	$p_6^{(i)}$	$p_7^{(i)}$	$p_4^{(i)}$	$p_5^{(i)}$	$p_2^{(i)}$	$p_3^{(i)}$	$p_0^{(i)}$	$p_1^{(i)}$	$p_{14}^{(i)}$	$p_{15}^{(i)}$
A_0		$p_{14}^{(i-1)}$		$-p_{11}^{(i-1)}$	$p_{11}^{(i-1)}$		$-p_8^{(i-1)}$	$p_8^{(i-1)}$		0	0	$r_3^{(i-1)}$	$r_4^{(i-1)}$	$r_5^{(i-1)}$	$br_4^{(i-1)}$	$br_5^{(i-1)}$		$-p_{14}^{(i)}$
A_1		$p_{16}^{(i-1)}$		$p_{12}^{(i-1)}$	$p_{13}^{(i-1)}$		$p_9^{(i-1)}$	$p_{10}^{(i-1)}$		$p_7^{(i)}$	$p_4^{(i)}$	$p_5^{(i)}$	$p_2^{(i)}$	$p_3^{(i)}$	$p_0^{(i)}$	$p_1^{(i)}$		$p_{15}^{(i)}$
A_2		$p_{15}^{(i-1)}$		0	$p_{12}^{(i-1)}$		0	$p_9^{(i-1)}$		$p_6^{(i)}$	0	0	0	0	0	0		0
A_3		$-a_5^{(i-1)}$		$-a_2^{(i-1)}$	$-a_3^{(i-1)}$		$-ba_0^{(i-1)}$	$-ba_1^{(i-1)}$		0	$r_2^{(i-1)}$	0	$r_0^{(i-1)}$	$r_1^{(i-1)}$	$a_0^{(i)}$	$a_1^{(i)}$		$-a_4^{(i)}$
Output of the 4-operand adder	$r_4^{(i-1)}$	$r_5^{(i-1)}$		$r_2^{(i-1)}$	$r_3^{(i-1)}$		$r_0^{(i-1)}$	$r_1^{(i-1)}$		$s_3^{(i)}$	$a_0^{(i)}$	$a_1^{(i)}$	$a_4^{(i)}$	$a_5^{(i)}$	$a_2^{(i)}$	$a_3^{(i)}$		$r_4^{(i)}$

Fig. 2. Scheduling of Miller's algorithm in characteristic three.
(N.B. The numerals ① to ⑤ refer to the datapaths between the output of the four-operand adder and the inputs of the parallel multiplier in Figure 1.)

stores the coefficients of $R^{(i)}$.

- According to our scheduling (Figure 2), the second DPRAM block provides the four-operand adder with its fourth input, namely $a_j^{(i)}$, $0 \leq j \leq 5$, and $r_j^{(i-1)}$, $0 \leq j \leq 2$.

2) Computation of the Sparse Operand: The second task consists in computing the coefficients of the sparse operand $S^{(i+1)}$ required for the next iteration of Miller's algorithm (Algorithm 1, lines 8 to 10). Two cubings and an addition over $\mathbf{F}_{3^m}$ allow us to update the coordinates of point P and to determine the coefficient $t^{(i+1)}$ of the sparse operand $S^{(i+1)}$, respectively.

Recall that the η_T pairing over $\mathbf{F}_{3^m}$ comes in two flavors: the original one involves a cubing over $\mathbf{F}_{3^{6m}}$ after each sparse multiplication. Barreto *et al.* [3] explained how to get rid of that cubing at the price of two cube roots over $\mathbf{F}_{3^m}$ to update the coordinates of point Q. It is essential to consider such an algorithm here, as an extra cubing over $\mathbf{F}_{3^{6m}}$ would put even more strain on the first task (which is already the most expensive one). According to our results, the critical path of the circuit is never located in a cube root operator when pairing-friendly irreducible trinomials or pentanomials [2], [20] are used to define $\mathbf{F}_{3^m}$. If by any chance such polynomials are not available for the considered extension of $\mathbf{F}_3$ and the critical path is in the cube root, it is always possible to pipeline this operation. Therefore, the cost of cube roots is hidden by the first task.

The hardware implementation is rather straightforward (Figure 1): four registers, a cubing operator, and a cube root operator allow us to store and update the coordinates of points P and Q. Then, a two-operand adder computes the sum of $x_P^{(i)}$ and $x_Q^{(i)}$, and the result $t^{(i)}$ is memorized in a fifth register. Multiplexers select the inputs of the parallel multiplier according to our scheduling.

3) Initialization: The initialization step of the η_T pairing (Algorithm 1, lines 1 and 2) involves a small amount of specific hardware in order to compute $x_P^{(0)}$, $y_P^{(0)}$, $x_Q^{(0)}$, and $y_Q^{(0)}$. Note that we are able to send $t^{(i)}$, $\lambda y_P^{(i)}$, and $-\lambda y_Q^{(i)}$ to input M_1 of the parallel multiplier (Figure 1). Assuming that the constants 0, 1, and 2 are stored in the DPRAM block connected to input M_0, we compute the coefficients of $R^{(-1)}$ by means of six multiplications over $\mathbf{F}_{3^m}$:

$$\begin{aligned} r_0^{(-1)} &= t^{(0)} \cdot \lambda y_P^{(0)}, & r_3^{(-1)} &= 0 \cdot t^{(0)} = 0, \\ r_1^{(-1)} &= 1 \cdot \left(-\lambda y_Q^{(0)}\right), & r_4^{(-1)} &= 0 \cdot t^{(0)} = 0, \text{ and} \\ r_2^{(-1)} &= 2 \cdot \lambda y_P^{(0)} = -\lambda y_P^{(0)}, & r_5^{(-1)} &= 0 \cdot t^{(0)} = 0. \end{aligned}$$

Loading the coordinates of points P and Q and performing the initialization step involves 17 clock cycles (*i.e.* exactly the same number of clock cycles as an iteration of Miller's algorithm). Therefore, our coprocessor returns $R^{((m-1)/2)}$ after $17 \cdot (m+3)/2$ clock cycles.

Algorithm 2 Sparse multiplication over $\mathbf{F}_{3^{6m}}$.

```
Input: $b \in \{-1, 1\}$; $t^{(i)}$, $y_P^{(i)}$, and $y_Q^{(i)} \in \mathbf{F}_{3^m}$; and
    $R^{(i-1)} \in \mathbf{F}_{3^{6m}}$.
Output: $R^{(i)} = R^{(i-1)} \cdot S^{(i)} \in \mathbf{F}_{3^{6m}}$, where
    $S^{(i)} = -\left(t^{(i)}\right)^2 + y_P^{(i)} y_Q^{(i)} \sigma - t^{(i)} \rho - \rho^2$.
for $j = 0$ to $5$ do
    $p_j^{(i)} \leftarrow r_j^{(i-1)} \cdot t^{(i)}$;
end for
$p_6^{(i)} \leftarrow t^{(i)} \cdot t^{(i)}$; $p_7^{(i)} \leftarrow -y_P^{(i)} \cdot y_Q^{(i)}$;
$s_3^{(i)} \leftarrow p_6^{(i)} + p_7^{(i)}$;
$a_0^{(i)} \leftarrow r_2^{(i-1)} + p_4^{(i)}$;
$a_1^{(i)} \leftarrow r_3^{(i-1)} + p_5^{(i)}$;
$a_2^{(i)} \leftarrow b r_4^{(i-1)} + p_0^{(i)} + a_0^{(i)}$;
$a_3^{(i)} \leftarrow b r_5^{(i-1)} + p_1^{(i)} + a_1^{(i)}$;
$a_4^{(i)} \leftarrow r_0^{(i-1)} + r_4^{(i-1)} + p_2^{(i)}$;
$a_5^{(i)} \leftarrow r_1^{(i-1)} + r_5^{(i-1)} + p_3^{(i)}$;
for $j = 0$ to $2$ do
    $p_{3j+8}^{(i)} \leftarrow r_{2j}^{(i-1)} \cdot p_6^{(i)}$;
    $p_{3j+9}^{(i)} \leftarrow r_{2j+1}^{(i-1)} \cdot p_7^{(i)}$;
    $s_j^{(i)} \leftarrow -r_{2j}^{(i-1)} - r_{2j+1}^{(i-1)}$;
    $p_{3j+10}^{(i)} \leftarrow s_j^{(i)} \cdot s_3^{(i)}$;
end for
$r_0^{(i)} \leftarrow -b a_0^{(i)} - p_8^{(i)} + p_9^{(i)}$;
$r_1^{(i)} \leftarrow -b a_1^{(i)} + p_8^{(i)} + p_9^{(i)} + p_{10}^{(i)}$;
$r_2^{(i)} \leftarrow -a_2^{(i)} - p_{11}^{(i)} + p_{12}^{(i)}$;
$r_3^{(i)} \leftarrow -a_3^{(i)} + p_{11}^{(i)} + p_{12}^{(i)} + p_{13}^{(i)}$;
$r_4^{(i)} \leftarrow -a_4^{(i)} - p_{14}^{(i)} + p_{15}^{(i)}$;
$r_5^{(i)} \leftarrow -a_5^{(i)} + p_{14}^{(i)} + p_{15}^{(i)} + p_{16}^{(i)}$;
return $r_0^{(i)} + r_1^{(i)}\sigma + r_2^{(i)}\rho + r_3^{(i)}\sigma\rho + r_4^{(i)}\rho^2 + r_5^{(i)}\sigma\rho^2$;
```

B. *Final Exponentiation*

The second and last stage in the computation of the η_T pairing is the final exponentiation, where the result of Miller's algorithm $R^{((m-1)/2)} = \eta_T(P, Q)$ is raised to the M-th power (Algorithm 1, line 12). This exponentiation is necessary since the non-reduced pairing $\eta_T(P, Q)$ is only defined up to N-th powers in $\mathbf{F}_{3^{6m}}^*$.

1) Improved Algorithm: In order to compute this final exponentiation, we use the algorithm presented by Beuchat *et al.* in [8]. This method exploits the special form of the exponent M (see Table I) to achieve better performances than with a classical square-and-multiply algorithm. Among other computations, this final exponentiation involves the raising of an element of $\mathbf{F}_{3^{6m}}^*$ to the power of $3^{(m+1)/2}$, which Beuchat *et al.* [8] perform by computing $(m+1)/2$ successive cubings over $\mathbf{F}_{3^{6m}}^*$. Each of these cubings requiring 6 cubings and 6 additions over $\mathbf{F}_{3^m}$, the total cost of this step is $3m+3$ cubings and $3m + 3$ additions.

We present here a new method for computing $U^{3^{(m+1)/2}}$ for $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2 \in \mathbf{F}_{3^{6m}}^*$ by exploiting the linearity of the Frobenius map (*i.e.* cubing in characteristic three) to reduce the number of additions. Indeed, noting that $\sigma^{3^i} = (-1)^i\sigma$, $\rho^{3^i} = \rho + ib$ and $(\rho^2)^{3^i} = \rho^2 - ib\rho + i^2$, we obtain the following formula for U^{3^i}, depending on the value of i:

$$\begin{aligned} U^{3^i} = {} & (u_0 - \epsilon_1 u_2 + \epsilon_2 u_4)^{3^i} && + \epsilon_3 (u_1 - \epsilon_1 u_3 + \epsilon_2 u_5)^{3^i} \sigma \\ & + (u_2 + \epsilon_1 u_4)^{3^i} \rho && + \epsilon_3 (u_3 + \epsilon_1 u_5)^{3^i} \sigma\rho \\ & + u_4^{3^i} \rho^2 && + \epsilon_3 u_5^{3^i} \sigma\rho^2, \end{aligned}$$

with $\epsilon_1 = -ib \bmod 3$, $\epsilon_2 = i^2 \bmod 3$, and $\epsilon_3 = (-1)^i$. Thus, according to the value of $(m+1)/2$ modulo 6, the computation of $U^{3^{(m+1)/2}}$ will still require $3m+3$ cubings but at most only 6 additions or subtractions over $\mathbf{F}_{3^m}$.

2) Hardware Implementation: Our first attempt at computing the final exponentiation was to use the unified arithmetic operator introduced by Beuchat *et al.* [8]. Unfortunately, due to the sequential scheduling inherent to this operator, it turned out that the final exponentiation algorithm required more clock cycles than the computation of Miller's algorithm by our coprocessor. We therefore had to consider a slightly more parallel architecture.

Noticing that the critical operations in the final exponentiation algorithm were multiplication and long sequences of cubings over $\mathbf{F}_{3^m}$, we designed the coprocessor for arithmetic over $\mathbf{F}_{3^m}$ depicted in Figure 3. Besides a register file implemented by means of DPRAM, our coprocessor embeds a parallel–serial multiplier [45] processing D coefficients of an operand at each clock cycle (typically $D = 13$ or 14), along with a novel unified operator supporting addition, subtraction, accumulation, Frobenius map (*i.e.* cubing), and double Frobenius map (*i.e.* raising to the ninth power). This architecture allowed us to efficiently implement the final exponentiation algorithm described for instance in [8], while taking advantage of the improvement proposed above.

3) Using Inverse Frobenius Maps: We adapt here the idea behind the square-root-and-multiply algorithm for exponentiation over binary finite fields given by Rodríguez-Henríquez *et al.* in [37].

From the final exponentiation algorithm given in [8], it can be noticed that Frobenius maps over $\mathbf{F}_{3^m}$ (*i.e.* cubings) are needed only to perform an inversion over $\mathbf{F}_{3^m}^*$ [8, Algorithms 10 and 11] and for raising an element of $\mathbf{F}_{3^{6m}}^*$ to the power of $3^{(m+1)/2}$, as already discussed in Section III-B.1.

As far as the inversion is concerned, note that the first two lines of [8, Algorithm 10] are dedicated to computing $u = d^{(3^m-3)/2}$ thanks to successive cubings and multiplications, where $d \in \mathbf{F}_{3^m}^*$ is the element to be inverted. It is actually also possible to compute that same element by means of only cube roots and multiplications by making use of the identity $\sqrt[3^i]{z} = z^{3^{m-i}}$ for all $z \in \mathbf{F}_{3^m}$, derived from Fermat's little theorem. Indeed, considering the family of elements $(w_i)_{0 \le i < m}$ such that $w_i = d^{(3^m - 3^{m-i})/2}$, we note that $w_1 = d^{3^{m-1}} = \sqrt[3]{d}$ and $u = d^{(3^m-3)/2} = w_{m-1}$. Since for any two integers n and n' we also have

$$w_{n+n'} = d^{(3^m - 3^{m-n})/2} \cdot d^{(3^{m-n} - 3^{m-n-n'})/2} = w_n \cdot \sqrt[3^n]{w_{n'}},$$

it follows that, given an addition chain of length l for $m - 1$, we can compute $u = w_{m-1}$ in l multiplications and $m - 1$

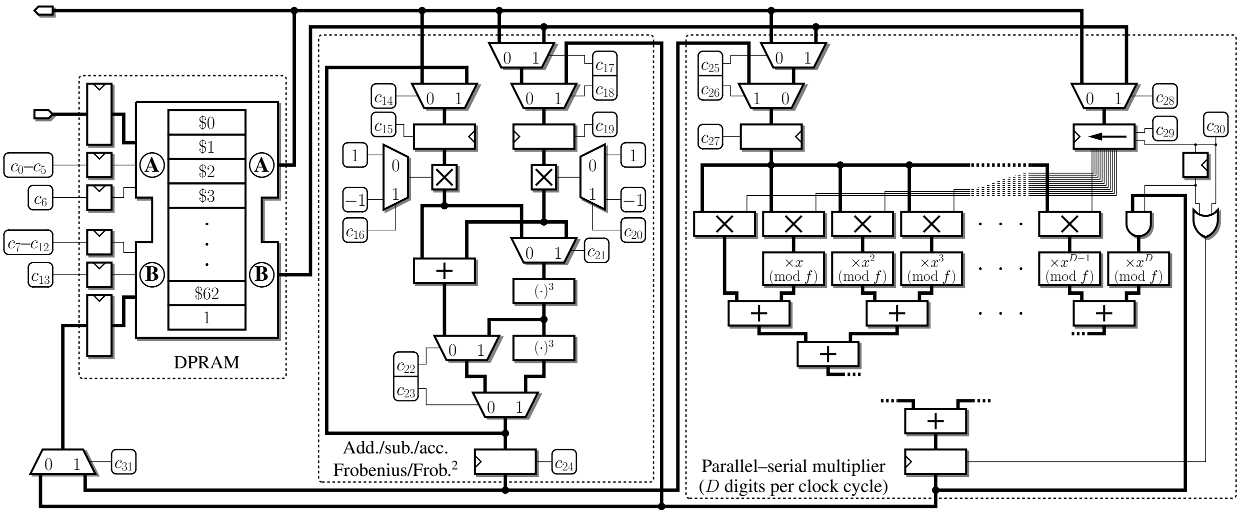


Fig. 3. Coprocessor for the final exponentiation of the η_T pairing in characteristic three. (N.B. All control bits c_i belong to $\{0, 1\}$.)

cube roots over $\mathbf{F}_{3^m}$. This has to be compared to the l multiplications and $m-1$ cubings required in [8, Algorithm 10] to obtain the same u.

As for the raising of an element of $\mathbf{F}^*_{3^{6m}}$ to the power of $3^{(m+1)/2}$, also part of the final exponentiation algorithm, we simply apply Fermat's little theorem once more to see that $z^{3^{(m+1)/2}} = \sqrt[3^{(m-1)/2}]{z}$ for all $z \in \mathbf{F}_{3^m}$. Thus, we can directly trade the $3m+3$ required cubings (as explained in the analysis given in Section III-B.1) for $3m-3$ cube roots over $\mathbf{F}_{3^m}$.

Hence, from the previous considerations, it is possible to replace all Frobenius maps (cubings) by inverse Frobenius maps (cube roots) in the final exponentiation. This is particularly interesting since the irreducible polynomial used to represent $\mathbf{F}_{3^m}$ was carefully chosen to allow for low-complexity cubings and cube roots, as both are required for the computation of the non-reduced η_T pairing. Furthermore, it appears that for the considered irreducible trinomials, the complexity of the cube root is always lower than that of the cubing. This is shown in Table II, where the third column reports the total number of required additions/subtractions over $\mathbf{F}_3$, and the fourth column indicates the largest number of elements of $\mathbf{F}_3$ that need to be added/subtracted to one another to compute a coefficient of the result (for instance, cubing over $\mathbf{F}_{3^{97}}$ requires summing at most four elements of $\mathbf{F}_3$ at a time).

In order to assess the impact of replacing the Frobenius and double-Frobenius operators by inverse-Frobenius (cube root) and double-inverse-Frobenius (ninth root) operators in the architecture presented in Figure 3, we implemented the different variants on Xilinx Virtex-4 LX FPGAs (xc4vlx40-11). The place-and-route results, reported in the last two columns of Table II, show that the use of cube roots usually shortens the critical path, even though the circuits are then slightly larger, as the cubing formulae generally involve more common subexpressions which can then share the same logic and decrease the total resource usage. All in all, it appears that relying on inverse Frobenius maps to compute the final exponentiation is by and large an effective optimization.

TABLE II
FROBENIUS VS. INVERSE FROBENIUS MAPS IN CHARACTERISTIC THREE, AND THEIR INFLUENCE ON THE COPROCESSOR OF FIGURE 3.

Field representation	**Op.**	**Total # of add.**	**Max. # of elements**	**Area [slices]**	**Freq. [MHz]**
$\mathbf{F}_3[x]/(x^{97}+x^{16}-1)$	$(\cdot)^3$	106	4	**4704**	185
	$\sqrt[3]{\cdot}$	96	3	4722	**192**
$\mathbf{F}_3[x]/(x^{167}-x^{71}+1)$	$(\cdot)^3$	229	5	**7607**	160
	$\sqrt[3]{\cdot}$	166	3	7682	**175**
$\mathbf{F}_3[x]/(x^{193}+x^{64}-1)$	$(\cdot)^3$	234	4	9265	**179**
	$\sqrt[3]{\cdot}$	192	3	**9092**	**179**
$\mathbf{F}_3[x]/(x^{239}-x^{5}+1)$	$(\cdot)^3$	242	4	**11589**	**179**
	$\sqrt[3]{\cdot}$	238	3	11848	177
$\mathbf{F}_3[x]/(x^{313}-x^{187}-1)$	$(\cdot)^3$	558	6	**15073**	141
	$\sqrt[3]{\cdot}$	312	3	15117	**172**

IV. Reduced η_T Pairing in Characteristic Two

An approach similar to that of characteristic three allowed us to design a parallel coprocessor for the reduced η_T pairing in characteristic two. The supersingular curves and the parameters of the algorithm are summarized in Table I.

A. Computation of Miller's Algorithm

Applying the strategy we used for characteristic three to the case of characteristic two, we adopted the reversed-loop algorithm described in [7], which we recall here in Algorithm 3. However, the scheduling turns out to be slightly more difficult than in characteristic three since we have to perform three tasks in parallel at each iteration of Miller's algorithm.

1) Sparse Multiplication over $\mathbf{F}_{2^{4m}}$: The intermediate result $F^{(i-1)}$ computed during the previous iteration is multiplied by the sparse operand $G^{(i)} = g_0^{(i)} + g_1^{(i)}s + t$ by means of a parallel Karatsuba multiplier (Algorithm 3, lines 13 and 14). This operation is easier than the standard multiplication over $\mathbf{F}_{2^{4m}}$ and requires only 6 multiplications and 14 additions over the base field $\mathbf{F}_{2^m}$, as explained in Algorithm 4. Thanks to the careful scheduling given in Figure 5, the intermediate variables $a_0^{(i)}$, $a_1^{(i)}$, and $a_2^{(i)}$ (Algorithm 4, lines 1 and 2) are generated

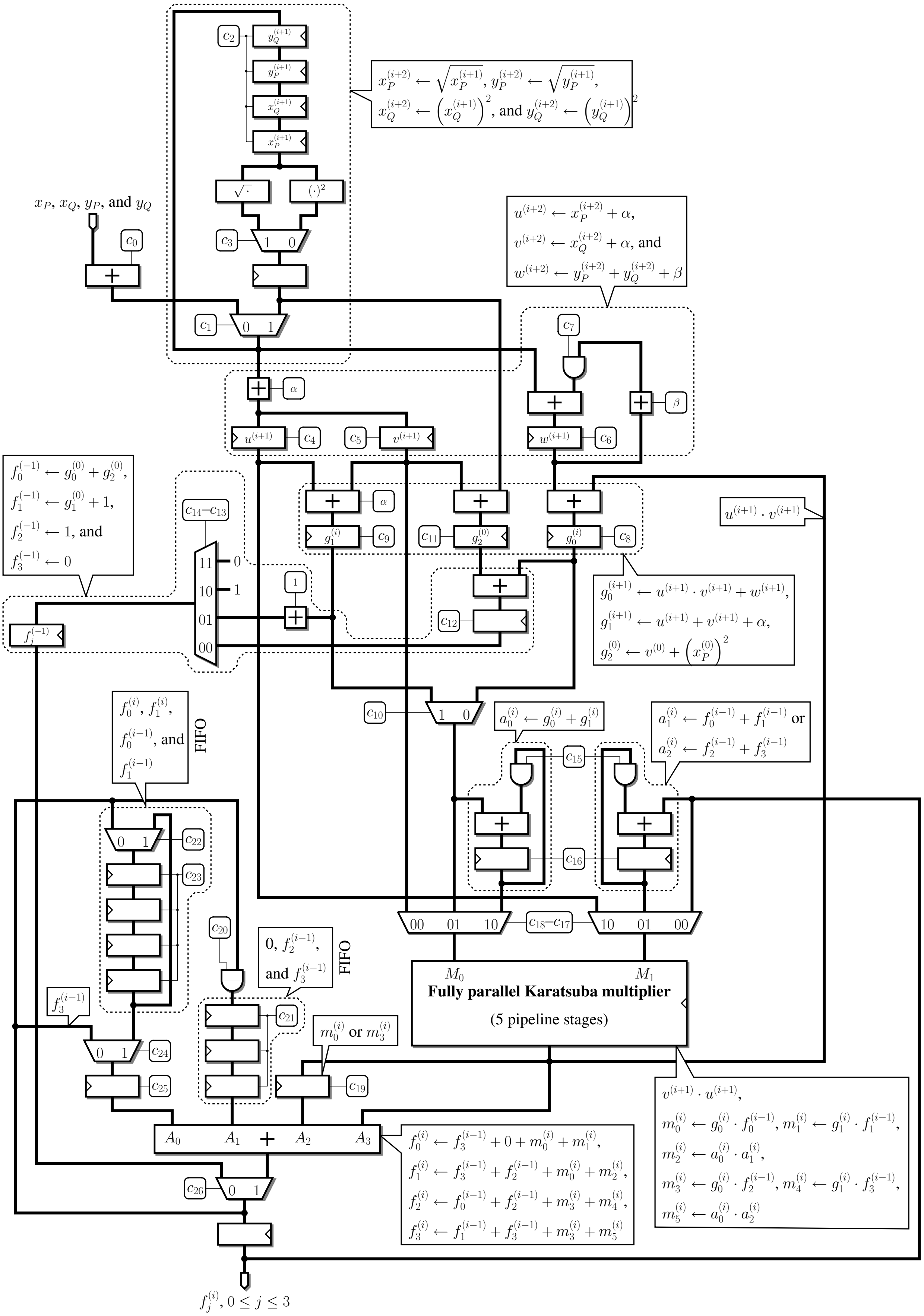

c_2
$y_Q^{(i+1)}$
$y_P^{(i+1)}$
$x_Q^{(i+1)}$
$x_P^{(i+1)}$
$x_P^{(i+2)} \leftarrow \sqrt{x_P^{(i+1)}}$, $y_P^{(i+2)} \leftarrow \sqrt{y_P^{(i+1)}}$,
$x_Q^{(i+2)} \leftarrow \left(x_Q^{(i+1)}\right)^2$, and $y_Q^{(i+2)} \leftarrow \left(y_Q^{(i+1)}\right)^2$
$\sqrt{\cdot}$
$(\cdot)^2$
x_P, x_Q, y_P, and y_Q
c_0
c_3
1 0
$u^{(i+2)} \leftarrow x_P^{(i+2)} + \alpha$,
$v^{(i+2)} \leftarrow x_Q^{(i+2)} + \alpha$, and
$w^{(i+2)} \leftarrow y_P^{(i+2)} + y_Q^{(i+2)} + \beta$
c_1
0 1
c_7
α
β
$u^{(i+1)}$
c_4
c_5
$v^{(i+1)}$
$w^{(i+1)}$
c_6
$f_0^{(-1)} \leftarrow g_0^{(0)} + g_2^{(0)}$,
$f_1^{(-1)} \leftarrow g_1^{(0)} + 1$,
$f_2^{(-1)} \leftarrow 1$, and
$f_3^{(-1)} \leftarrow 0$
c_{14}–c_{13}
α
$u^{(i+1)} \cdot v^{(i+1)}$
$g_1^{(i)}$
c_9
c_{11}
$g_2^{(0)}$
$g_0^{(i)}$
c_8
11
10
01
00
0
1
1
$f_j^{(-1)}$
c_{12}
$g_0^{(i+1)} \leftarrow u^{(i+1)} \cdot v^{(i+1)} + w^{(i+1)}$,
$g_1^{(i+1)} \leftarrow u^{(i+1)} + v^{(i+1)} + \alpha$,
$g_2^{(0)} \leftarrow v^{(0)} + \left(x_P^{(0)}\right)^2$
$f_0^{(i)}$, $f_1^{(i)}$,
$f_0^{(i-1)}$, and
$f_1^{(i-1)}$
FIFO
c_{10}
1 0
$a_0^{(i)} \leftarrow g_0^{(i)} + g_1^{(i)}$
$a_1^{(i)} \leftarrow f_0^{(i-1)} + f_1^{(i-1)}$ or
$a_2^{(i)} \leftarrow f_2^{(i-1)} + f_3^{(i-1)}$
c_{15}
0 1
c_{22}
c_{23}
c_{16}
c_{20}
00 01 10
c_{18}–c_{17}
10 01 00
0, $f_2^{(i-1)}$,
and $f_3^{(i-1)}$
FIFO
M_0
M_1
Fully parallel Karatsuba multiplier
(5 pipeline stages)
$f_3^{(i-1)}$
c_{21}
$m_0^{(i)}$ or $m_3^{(i)}$
0 1
c_{24}
c_{25}
c_{19}
$v^{(i+1)} \cdot u^{(i+1)}$,
$m_0^{(i)} \leftarrow g_0^{(i)} \cdot f_0^{(i-1)}$, $m_1^{(i)} \leftarrow g_1^{(i)} \cdot f_1^{(i-1)}$,
$m_2^{(i)} \leftarrow a_0^{(i)} \cdot a_1^{(i)}$,
$m_3^{(i)} \leftarrow g_0^{(i)} \cdot f_2^{(i-1)}$, $m_4^{(i)} \leftarrow g_1^{(i)} \cdot f_3^{(i-1)}$,
$m_5^{(i)} \leftarrow a_0^{(i)} \cdot a_2^{(i)}$
A_0 A_1 + A_2 A_3
$f_0^{(i)} \leftarrow f_3^{(i-1)} + 0 + m_0^{(i)} + m_1^{(i)}$,
$f_1^{(i)} \leftarrow f_3^{(i-1)} + f_2^{(i-1)} + m_0^{(i)} + m_2^{(i)}$,
$f_2^{(i)} \leftarrow f_0^{(i-1)} + f_2^{(i-1)} + m_3^{(i)} + m_4^{(i)}$,
$f_3^{(i)} \leftarrow f_1^{(i-1)} + f_3^{(i-1)} + m_3^{(i)} + m_5^{(i)}$
c_{26}
0 1
$f_j^{(i)}, 0 \leq j \leq 3$

Fig. 4. Coprocessor for the η_T pairing in characteristic two. (N.B. All control bits c_i belong to $\{0, 1\}$.)

Algorithm 3 Computation of the reduced η_T pairing in characteristic two.†

```
Input: $P = (x_P, y_P)$, $Q = (x_Q, y_Q) \in E(\mathbf{F}_{2^m})[\ell]$.
Output: $\eta_T(P,Q)^M \in \mathbf{F}^*_{2^{4m}}$.
1. $x_P^{(0)} \leftarrow x_P$;   $y_P^{(0)} \leftarrow y_P + \bar{\delta}$;
2. $x_Q^{(0)} \leftarrow x_Q$;   $y_Q^{(0)} \leftarrow y_Q$;
3. $u^{(0)} \leftarrow x_P^{(0)} + \alpha$;   $v^{(0)} \leftarrow x_Q^{(0)} + \alpha$;
4. $w^{(0)} \leftarrow y_P^{(0)} + y_Q^{(0)} + \beta$;
5. $g_0^{(0)} \leftarrow u^{(0)} \cdot v^{(0)} + w^{(0)}$;
6. $g_1^{(0)} \leftarrow u^{(0)} + v^{(0)} + \alpha$; $g_2^{(0)} \leftarrow v^{(0)} + \left(x_P^{(0)}\right)^2$;
7. $x_P^{(1)} \leftarrow \sqrt{x_P^{(0)}}$;   $y_P^{(1)} \leftarrow \sqrt{y_P^{(0)}}$;
8. $x_Q^{(1)} \leftarrow \left(x_Q^{(0)}\right)^2$;   $y_Q^{(1)} \leftarrow \left(y_Q^{(0)}\right)^2$;
9. $u^{(1)} \leftarrow x_P^{(1)} + \alpha$;   $v^{(1)} \leftarrow x_Q^{(1)} + \alpha$;
10. $w^{(1)} \leftarrow y_P^{(1)} + y_Q^{(1)} + \beta$;
11. $F^{(-1)} \leftarrow \left(g_0^{(0)} + g_2^{(0)}\right) + \left(g_1^{(0)} + 1\right)s + t$;
12. for $i = 0$ to $\frac{m-1}{2}$ do
13.     $G^{(i)} \leftarrow g_0^{(i)} + g_1^{(i)}s + t$;
14.     $F^{(i)} \leftarrow F^{(i-1)} \cdot G^{(i)}$;
15.     $g_0^{(i+1)} \leftarrow u^{(i+1)} \cdot v^{(i+1)} + w^{(i+1)}$;
16.     $g_1^{(i+1)} \leftarrow u^{(i+1)} + v^{(i+1)} + \alpha$;
17.     $x_P^{(i+2)} \leftarrow \sqrt{x_P^{(i+1)}}$;   $y_P^{(i+2)} \leftarrow \sqrt{y_P^{(i+1)}}$;
18.     $x_Q^{(i+2)} \leftarrow \left(x_Q^{(i+1)}\right)^2$;   $y_Q^{(i+2)} \leftarrow \left(y_Q^{(i+1)}\right)^2$;
19.     $u^{(i+2)} \leftarrow x_P^{(i+2)} + \alpha$;   $v^{(i+2)} \leftarrow x_Q^{(i+2)} + \alpha$;
20.     $w^{(i+2)} \leftarrow y_P^{(i+2)} + y_Q^{(i+2)} + \beta$;
21. end for
22. return $\left(F^{((m-1)/2)}\right)^M$;
```

†Intermediate variables in uppercase belong to $\mathbf{F}_{2^{4m}}$, those in lowercase to $\mathbf{F}_{2^m}$.

Algorithm 4 Sparse multiplication over $\mathbf{F}_{2^{4m}}$ [7].

```
Input: $G^{(i)} = g_0^{(i)} + g_1^{(i)}s + t \in \mathbf{F}_{2^{4m}}$ and
    $F^{(i-1)} = f_0^{(i-1)} + f_1^{(i-1)}s + f_2^{(i-1)}t + f_3^{(i-1)}st \in \mathbf{F}_{2^{4m}}$.
Output: $F^{(i)} = G^{(i)} \cdot F^{(i-1)}$.
1. $a_0^{(i)} \leftarrow g_0^{(i)} + g_1^{(i)}$;   $a_1^{(i)} \leftarrow f_0^{(i-1)} + f_1^{(i-1)}$;
2. $a_2^{(i)} \leftarrow f_2^{(i-1)} + f_3^{(i-1)}$;
3. $m_0^{(i)} \leftarrow g_0^{(i)} \cdot f_0^{(i-1)}$;   $m_1^{(i)} \leftarrow g_1^{(i)} \cdot f_1^{(i-1)}$;
4. $m_2^{(i)} \leftarrow a_0^{(i)} \cdot a_1^{(i)}$;   $m_3^{(i)} \leftarrow g_0^{(i)} \cdot f_2^{(i-1)}$;
5. $m_4^{(i)} \leftarrow g_1^{(i)} \cdot f_3^{(i-1)}$;   $m_5^{(i)} \leftarrow a_0^{(i)} \cdot a_2^{(i)}$;
6. $f_0^{(i)} \leftarrow m_0^{(i)} + m_1^{(i)} + f_3^{(i-1)}$;
7. $f_1^{(i)} \leftarrow m_0^{(i)} + m_2^{(i)} + f_2^{(i-1)} + f_3^{(i-1)}$;
8. $f_2^{(i)} \leftarrow m_3^{(i)} + m_4^{(i)} + f_0^{(i-1)} + f_2^{(i-1)}$;
9. $f_3^{(i)} \leftarrow m_3^{(i)} + m_5^{(i)} + f_1^{(i-1)} + f_3^{(i-1)}$;
10. return $f_0^{(i)} + f_1^{(i)}s + f_2^{(i)}t + f_3^{(i)}st$;
```

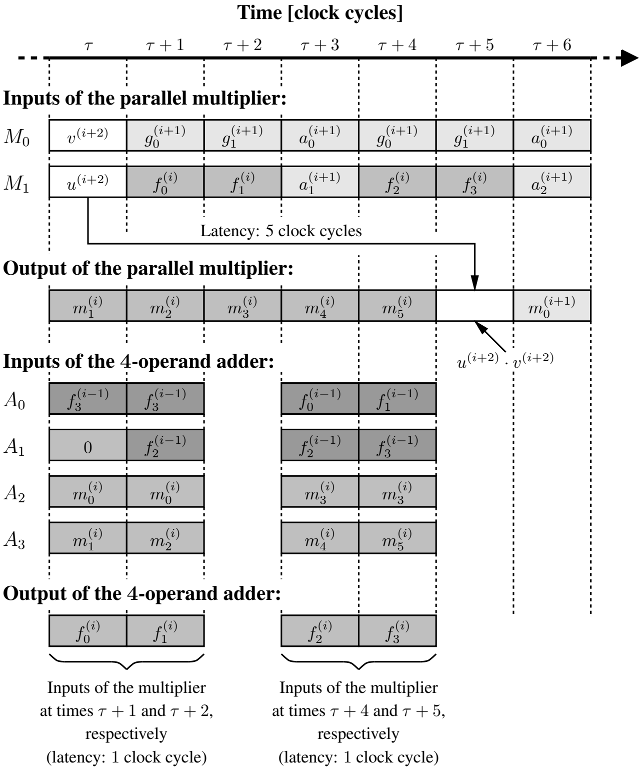


Fig. 5. Scheduling of Miller's algorithm in characteristic two.

on-the-fly by means of two accumulators. (Note that in order to share the control bits c_{15} and c_{16} between both accumulators, we compute $a_0^{(i)}$ twice at each iteration of Miller's algorithm.) The addition of at most four operands belonging to $\mathbf{F}_{2^m}$ allows for computing $f_j^{(i)}$, $0 \leq j \leq 3$ (Algorithm 4, lines 6 to 9).

It is worth mentioning that the datapath between the output of the four-operand adder and the parallel multiplier is much simpler than in characteristic three: it suffices to delay $f_j^{(i)}$, $0 \leq j \leq 3$, by one clock cycle and there is therefore no need for a memory block to store the operands of the multiplier. Dealing with inputs A_0 and A_1 of the four-operand adder is unfortunately more difficult because of data dependencies between the coefficients of $F^{(i-1)}$ and $F^{(i)}$ in Algorithm 4. According to our scheduling, we update the coefficients of $F^{(i)}$ in the following order: $f_0^{(i)}$, $f_1^{(i)}$, $f_2^{(i)}$, and eventually $f_3^{(i)}$. Since $f_2^{(i)}$ and $f_3^{(i)}$ depend on $f_0^{(i-1)}$ and $f_1^{(i-1)}$, respectively, we have to keep a copy of those values until the end of the i-th iteration of Miller's algorithm. Instead of including a DPRAM block in our design, we propose a solution based on two small FIFOs (see Figure 6 for details). An advantage of characteristic two over characteristic three is that the register file is smaller in terms of circuit area and requires fewer control bits.

2) Computation of g_0 and g_1: Both coefficients $g_0^{(i+1)}$ and $g_1^{(i+1)}$ (Algorithm 3, lines 15 and 16) are involved in the next sparse multiplication and thus have to be computed in beforehand. The product $u^{(i+1)} \cdot v^{(i+1)}$ is evaluated by means of our parallel Karatsuba multiplier. Then, two-operand adders allow for working out $g_0^{(i+1)}$ and $g_1^{(i+1)}$.

3) Computation of u, v, and w: The three values $u^{(i+2)}$, $v^{(i+2)}$, and $w^{(i+2)}$ (Algorithm 3, lines 17 to 20) will be

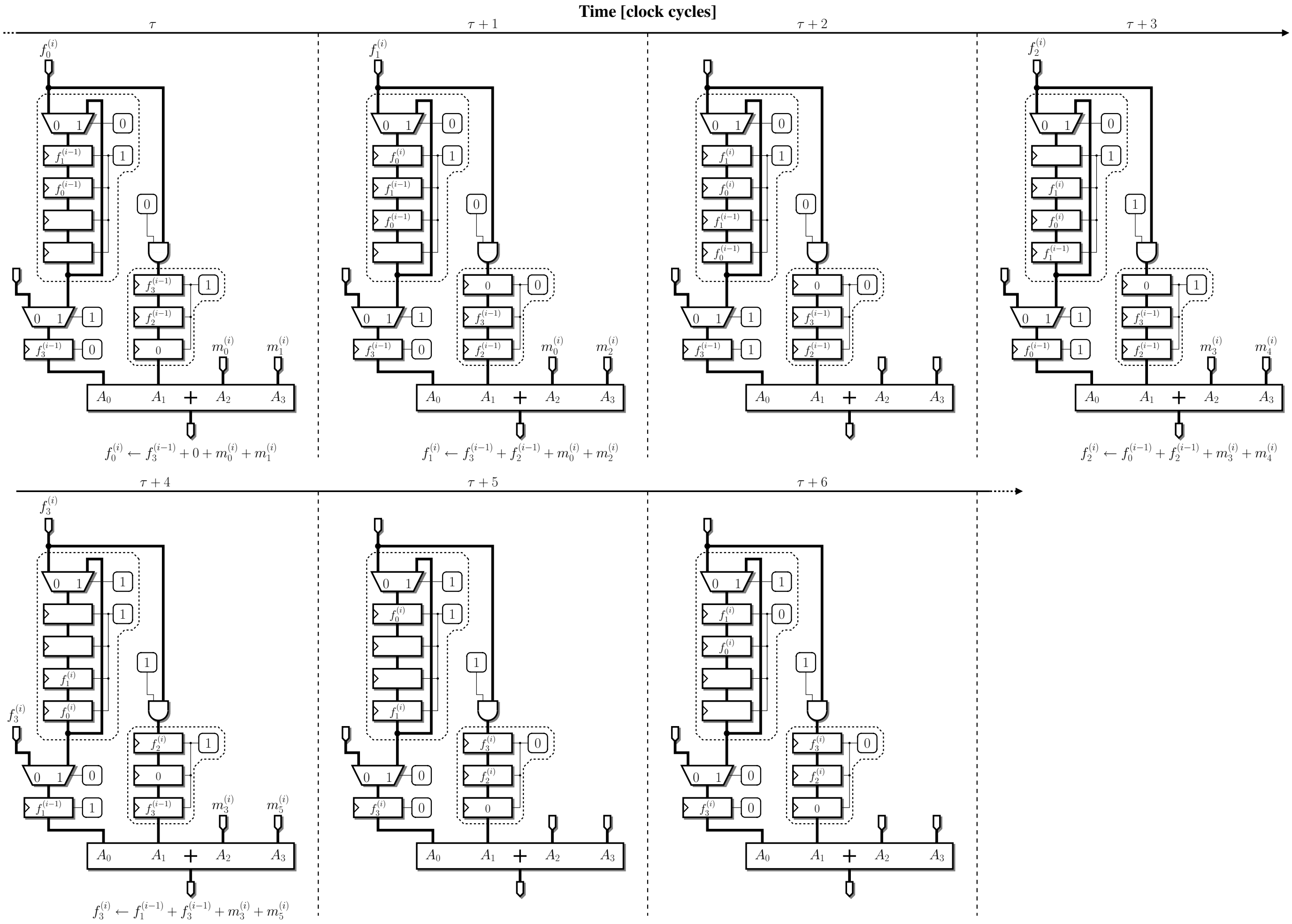

Time [clock cycles]
τ
τ + 1
τ + 2
τ + 3
τ + 4
τ + 5
τ + 6
A0 A1 + A2 A3
$f_0^{(i)} \leftarrow f_3^{(i-1)} + 0 + m_0^{(i)} + m_1^{(i)}$
$f_1^{(i)} \leftarrow f_3^{(i-1)} + f_2^{(i-1)} + m_0^{(i)} + m_2^{(i)}$
$f_2^{(i)} \leftarrow f_0^{(i-1)} + f_2^{(i-1)} + m_3^{(i)} + m_4^{(i)}$
$f_3^{(i)} \leftarrow f_1^{(i-1)} + f_3^{(i-1)} + m_3^{(i)} + m_5^{(i)}$

Fig. 6. Contents of the register file of our coprocessor for the η_T pairing in characteristic two.

required to calculate $g_0^{(i+2)}$ and $g_1^{(i+2)}$ during the next iteration of Miller's algorithm. Here we have to work with the coordinates of points P and Q that are stored in a FIFO and updated by means of squaring and square-root operators, respectively (see Section IV-A.6). Then, an accumulator allows for computing $w^{(i+2)}$ in two clock cycles. Depending on the values of α and β, 1-bit adders (*i.e.* XOR gates) are necessary to calculate the least significant bit of $u^{(i+2)}$, $v^{(i+2)}$, and $w^{(i+2)}$.

4) Choosing the Adequate Karatsuba Multiplier: In these settings, a Karatsuba multiplier with 5 pipeline stages can be kept busy during the computation of the main loop, as shown in Figure 5. Since we have to carry out 7 multiplications over $\mathbf{F}_{2^m}$ at each iteration, the calculation time for the full loop is equal to $7{\cdot}(m{+}1)/2$ clock cycles. It is again crucial to consider an algorithm with inverse Frobenius maps (*i.e.* square roots) in order to avoid squaring $F^{(i)}$ at each iteration of Miller's algorithm (see for instance [7] for a survey of algorithms for the Tate pairing over supersingular curves in characteristic two). Such an operation would lengthen the computation time and pipeline bubbles would be inserted in the multiplier.

5) Initialization: The initialization step requires specific attention. In order to start multiplying $u^{(0)}$ by $v^{(0)}$ as soon as possible (Algorithm 3, line 5), we load the coordinates of points P and Q in the following order: x_P, x_Q, y_P, and y_Q. Thus, $u^{(0)}$ and $v^{(0)}$ are available after two clock cycles. Thanks to this scheduling, we complete the initialization step in 15 clock cycles.

6) Irreducible Pentanomials Suitable for Low-Complexity Square-Root Computation: Although irreducible trinomials allowing for simple computations of squarings and square roots exist in some binary finite fields, as detailed in [37], this was not the case for several fields considered in this work (see Table III). To tackle this issue, we present here a novel family of irreducible square-root-friendly pentanomials that, to the best of our knowledge, has not been proposed before in the literature.

Let m and d be two odd positive integers with $d < m/2$ and such that the degree-m monic polynomial

$$f(x) = x^m + x^{m-d} + x^{m-2d} + x^d + 1$$

is irreducible over $\mathbf{F}_2$. We then represent the binary extension field $\mathbf{F}_{2^m}$ as $\mathbf{F}_2[x]/\left(f(x)\right)$.

Note that, reducing modulo f, we have $x^{m+2d+1} + x^{m+1} + x^{m-2d+1} + x^{3d+1} = x$, where all the exponents on the left-hand side are even. It then follows that

$$\sqrt{x} = x^{\frac{m+2d+1}{2}} + x^{\frac{m+1}{2}} + x^{\frac{m-2d+1}{2}} + x^{\frac{3d+1}{2}}.$$

Therefore, using this expression for $\sqrt{x}$, we can compute the square root of an element $a \in \mathbf{F}_{2^m}$ as [16]

$$\sqrt{a} = \sum_{i=0}^{\frac{m-1}{2}} a_{2i}x^i + \sqrt{x}\sum_{i=0}^{\frac{m-3}{2}} a_{2i+1}x^i \bmod f(x).$$

Furthermore, one can show that if $(2m-1)/7 \le d \le (2m+1)/5$ then the complete computation of the square root (*i.e.* including the reduction modulo f) will require the addition of at most three elements of $\mathbf{F}_2$ at a time for each coefficient of the result. And finally, choosing $d \ge (m-1)/6$ ensures that a squaring will involve only additions of at most four operands.

As reported in Table III, three pentanomials of this family have been selected to represent the finite fields $\mathbf{F}_{2^{557}}$, $\mathbf{F}_{2^{613}}$, and $\mathbf{F}_{2^{691}}$, with $d =$ 197, 185, and 243, respectively.

TABLE III

FROBENIUS VS. INVERSE FROBENIUS MAPS IN CHARACTERISTIC TWO.

Field representation	Op.	Total # of add.	Max. # of elements
$\mathbf{F}_2[x]/(x^{239}+x^{81}+1)$	$(\cdot)^2$	159	3
	$\sqrt{\cdot}$	119	2
$\mathbf{F}_2[x]/(x^{313}+x^{79}+1)$	$(\cdot)^2$	195	3
	$\sqrt{\cdot}$	156	2
$\mathbf{F}_2[x]/(x^{457}+x^{61}+1)$	$(\cdot)^2$	258	3
	$\sqrt{\cdot}$	228	2
$\mathbf{F}_2[x]/(x^{557}+x^{360}+x^{197}+x^{163}+1)$	$(\cdot)^2$	752	4
	$\sqrt{\cdot}$	688	3
$\mathbf{F}_2[x]/(x^{613}+x^{428}+x^{243}+x^{185}+1)$	$(\cdot)^2$	883	4
	$\sqrt{\cdot}$	733	3
$\mathbf{F}_2[x]/(x^{691}+x^{448}+x^{243}+x^{205}+1)$	$(\cdot)^2$	932	4
	$\sqrt{\cdot}$	849	3

B. *Final Exponentiation*

The final exponentiation (Algorithm 3, line 22) is carried out according to the algorithms proposed in [7] and [42], [43] for $\nu = 1$ and $\nu = -1$, respectively. We took advantage of the algorithm introduced in [12] when raising to the (2^m+1)-st power over $\mathbf{F}_{2^{4m}}$. Here again, the linearity of the Frobenius map allows us to reduce the number of additions when computing $U^{2^{(m+1)/2}}$ for $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbf{F}^*_{2^{4m}}$. Noting that $s^{2^i} = s + \gamma_1$ and $t^{2^i} = t + \gamma_1 s + \gamma_2$, where $\gamma_1 = i \bmod 2$ and $\gamma_2 = \lfloor \frac{i}{2} \rfloor \bmod 2$, we obtain the following formula for U^{2^i}, depending on the value of i modulo 4:

$$\begin{aligned} U^{2^i} \quad = \quad & (u_0 + \gamma_1 u_1 + \gamma_2 u_2 + \gamma_3 u_3)^{2^i} \\ & + (u_1 + \gamma_1 u_2 + \gamma_2 u_3)^{2^i} s \\ & + (u_2 + \gamma_1 u_3)^{2^i} t + u_3^{2^i} st, \end{aligned}$$

where $\gamma_3 = 1$ when $i \bmod 4 = 1$, and $\gamma_3 = 0$ otherwise. According to the value of $(m+1)/2 \bmod 4$, the computation of $U^{2^{\frac{m+1}{2}}}$ requires $2m{+}2$ squarings and at most four additions over $\mathbf{F}_{2^m}$.

Here again, a hybrid arithmetic operator, similar to the one used in the case of characteristic three (see Figure 3), allows us to perform the final exponentiation in slightly less clock cycles than Miller's algorithm without impacting too much on the resource usage. The architecture is very similar to that of characteristic three, except that we removed the multiplications by 1 and -1, which are useless in characteristic two, and replaced the double-cubing by a triple-squaring operator, to accommodate for the longer chains of successive Frobenius maps in the final exponentiation algorithm. The parallel–serial multiplier processes here between $D = 15$ and $D = 17$ coefficients of its second operand per clock cycle.

It is worth noting that the trick of trading Frobenius for inverse Frobenius maps used for the final exponentiation in characteristic three can also be applied to the case of characteristic two. Indeed, as reported in Table III, the complexity of the square root is always lower than that of the squaring over the considered finite fields.

However, putting this optimization into practice happens to be more complex than in characteristic three. Apart from the Frobenius maps required for the inversion over $\mathbf{F}^*_{2^m}$ and for the raising to the power of $2^{(m+1)/2}$ over $\mathbf{F}^*_{2^{4m}}$, further squarings are actually necessary to raise elements of $\mathbf{F}^*_{2^{4m}}$ to the $(2^{2m}{-}1)$-st and $(2^m{+}1)$-st powers, as per [7, Algorithm 3] and [12, Algorithm 3] respectively.

These few squarings could be replaced by actual multiplications, which would then slightly increase the number of clock cycles required to compute the final exponentiation. Alternatively, we could take the fourth root of the result, which by linearity of the square root would then cancel all the extra Frobenius maps, but we would then end up computing a fixed power of the η_T pairing and not the η_T pairing itself.

However, observing that in characteristic two the critical path of the whole pairing accelerator lies in the non-reduced-pairing coprocessor and not in the final-exponentiation one, this is actually a moot point as there is no use trying to shorten further the critical path in the final exponentiation. We therefore decided against using this optimization altogether in the case of characteristic two.

V. Results and Comparisons

A. *Comparison with Previous Works*

Thanks to our automatic VHDL code generator, we designed several versions of the proposed architectures and prototyped our coprocessors on Xilinx Virtex-II Pro and Virtex-4 LX FPGAs with average speedgrade. Table IV details the specifics of the considered supersingular curves, while Table V provides the reader with a comparison between our work and accelerators for the Tate and η_T pairings over supersingular (hyper)elliptic curves published in the open literature. (Note that our comparison remains fair since the Tate pairing can be computed from the η_T pairing at no extra cost [7].) Finally, these results are summarized in Figure 7, where post-place-and-route computation time and area–time product estimations are plotted against the achieved level of security.

In the presented benchmarks, the logic resource usage is given in terms of *slices*, which is the usual metric on Xilinx FPGAs. Each slice comprises two 4-input look-up tables and two 1-bit flip-flops. Furthermore, it is worth noting that even though our coprocessors also make use of some embedded memory blocks as register files, they are by far not a critical resource and are therefore not reported in the benchmarks.

TABLE IV
SUPERSINGULAR ELLIPTIC CURVES CONSIDERED IN THIS PAPER.

Field	**Curve equation**	**Co-factor(s)**	**ECC security** ($\log_2 \ell$)	**MOV security** ($\log_2 \#\mathbf{F}_{p^{km}}$)	**Security**†
$\mathbf{F}_{2^{239}}$	$y^2 + y = x^3 + x + 1$	1	119 bits	956 bits	67 bits
$\mathbf{F}_{2^{313}}$	$y^2 + y = x^3 + x$	$5 \cdot 1933526201 \cdot 307168226569$ $\cdot 338431049916629$	97 bits	1252 bits	75 bits
$\mathbf{F}_{2^{457}}$	$y^2 + y = x^3 + x + 1$	1	228 bits	1828 bits	88 bits
$\mathbf{F}_{2^{557}}$	$y^2 + y = x^3 + x$	5	277 bits	2228 bits	96 bits
$\mathbf{F}_{2^{613}}$	$y^2 + y = x^3 + x$	5	305 bits	2452 bits	100 bits
$\mathbf{F}_{2^{691}}$	$y^2 + y = x^3 + x$	5	344 bits	2764 bits	105 bits
$\mathbf{F}_{3^{97}}$	$y^2 = x^3 - x + 1$	7	75 bits	922 bits	66 bits
$\mathbf{F}_{3^{167}}$	$y^2 = x^3 - x + 1$	7	131 bits	1588 bits	83 bits
$\mathbf{F}_{3^{193}}$	$y^2 = x^3 - x - 1$	1	153 bits	1835 bits	89 bits
$\mathbf{F}_{3^{239}}$	$y^2 = x^3 - x - 1$	1	189 bits	2273 bits	97 bits
$\mathbf{F}_{3^{313}}$	$y^2 = x^3 - x + 1$	$7 \cdot 37561 \cdot 477013$	230 bits	2977 bits	109 bits

†Security is given here as the required key length for a symmetric-key cryptosystem of equivalent security.

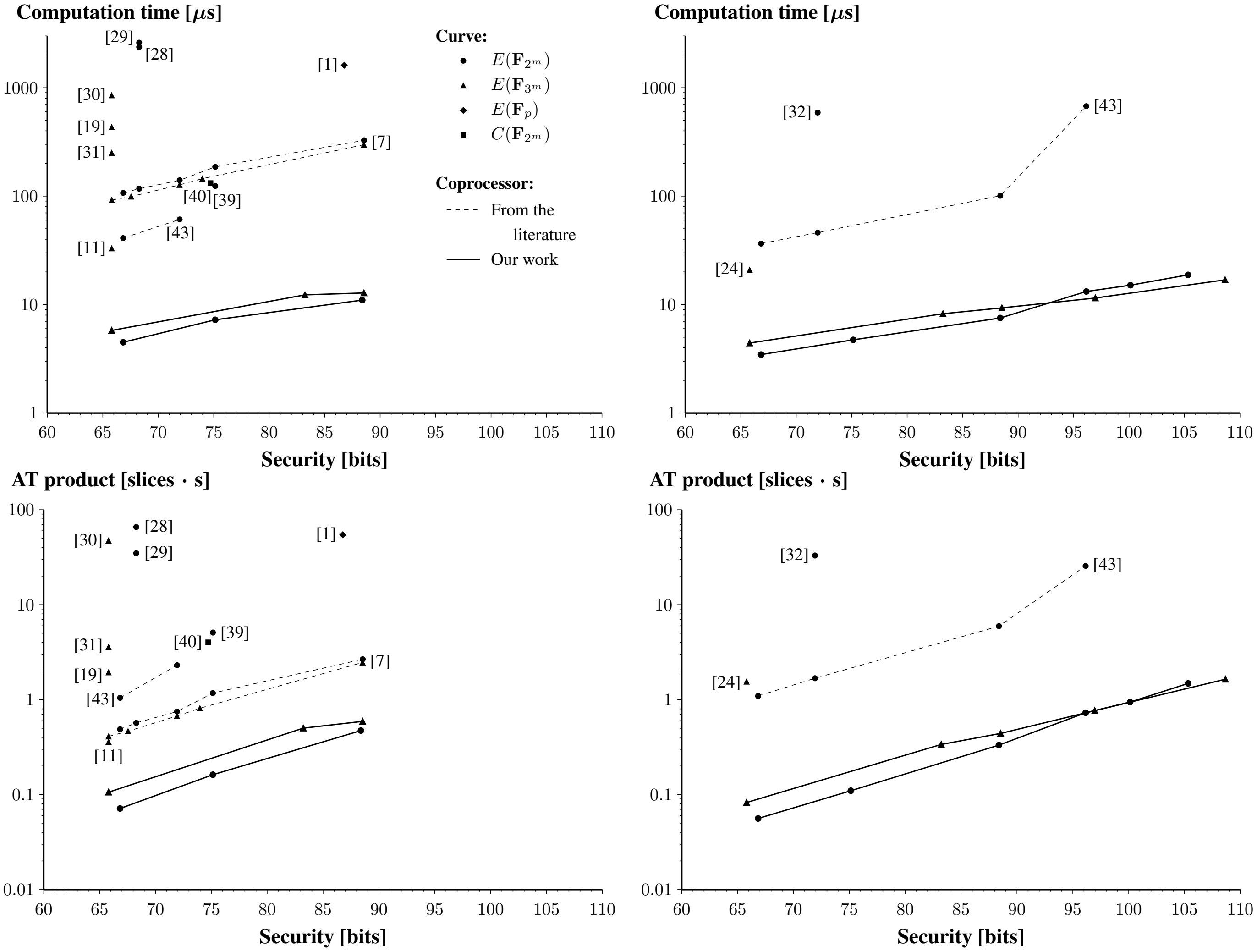


Fig. 7. Performance comparison in terms of pairing computation time (top) and area–time (AT) product (bottom) between the proposed architectures and the coprocessors published in the literature, on Virtex-II Pro (left) and Virtex-4 (right) FPGAs.

TABLE V

HARDWARE ACCELERATORS FOR THE TATE AND η_T PAIRINGS.

	Curve	Security [bits]	FPGA	Area [slices]	Frequency [MHz]	Calculation time [μs]	Area–time product
Kerins *et al.* [30]	$E(\mathbf{F}_{3^{97}})$	66	xc2vp125	55616	15	850	47.3
Kömürcü & Savas[*] [31]	$E(\mathbf{F}_{3^{97}})$	66	xc2vp100	14267	77	250.7	3.6
Ronan *et al.* [38]	$E(\mathbf{F}_{3^{97}})$	66	xc2vp100-6	15401	85	183	2.8
Grabher & Page [19]	$E(\mathbf{F}_{3^{97}})$	66	xc2vp4-6	4481	150	432.3	1.9
Jiang [24]	$E(\mathbf{F}_{3^{97}})$	66	xc4vlx200-11	74105	78	20.9	1.55
Beuchat *et al.* [7]	$E(\mathbf{F}_{3^{97}})$	66	xc2vp20-6	4455	105	92	0.41
Beuchat *et al.* [11]	$E(\mathbf{F}_{3^{97}})$	66	xc2vp30-6	10897	147	33	0.36
This work[†]	$E(\mathbf{F}_{3^{97}})$	66	xc2vp50-6	18367	147	5.78	0.106
	$E(\mathbf{F}_{3^{97}})$	66	xc4vlx60-11	18701	192	4.42	0.083
Shu *et al.* [43]	$E(\mathbf{F}_{2^{239}})$	67	xc2vp100-6	25487	84	41	1.04
Shu *et al.* [43]	$E(\mathbf{F}_{2^{239}})$	67	xc4vlx200-10	29920	100	36.5	1.09
Beuchat *et al.* [7]	$E(\mathbf{F}_{2^{239}})$	67	xc2vp20-6	4557	123	107	0.49
This work	$E(\mathbf{F}_{2^{239}})$	67	xc2vp50-6	15919	190	4.49	0.071
	$E(\mathbf{F}_{2^{239}})$	67	xc4vlx60-11	16203	247	3.46	0.056
Keller *et al.* [28]	$E(\mathbf{F}_{2^{251}})$	68	xc2v6000-4	27725	40	2370	65.7
Keller *et al.* [29]	$E(\mathbf{F}_{2^{251}})$	68	xc2v6000-4	13387	40	2600	34.8
Li *et al.* [32]	$E(\mathbf{F}_{2^{283}})$	72	xc4vfx140-11	55844	160	590	32.9
Shu *et al.* [43]	$E(\mathbf{F}_{2^{283}})$	72	xc2vp100-6	37803	72	61	2.3
Shu *et al.* [43]	$E(\mathbf{F}_{2^{283}})$	72	xc4vlx200-10	36481	100	46.1	1.68
Ronan *et al.* [39]	$E(\mathbf{F}_{2^{313}})$	75	xc2vp100-6	41078	50	124	5.1
Ronan *et al.* [40]	$C(\mathbf{F}_{2^{103}})$	75	xc2vp100-6	30464	41	132	4.02
This work	$E(\mathbf{F}_{2^{313}})$	75	xc2vp70-6	22395	154	7.24	0.162
	$E(\mathbf{F}_{2^{313}})$	75	xc4vlx80-11	23254	235	4.73	0.110
This work[†]	$E(\mathbf{F}_{3^{167}})$	83	xc2vp100-6	40765	118	12.3	0.50
	$E(\mathbf{F}_{3^{167}})$	83	xc4vlx100-11	40974	175	8.24	0.34
Barenghi *et al.* [1]	$E(\mathbf{F}_{p_{512}})$	87	xc2v8000-5	33857	135	1610	54.5
Shu *et al.* [43]	$E(\mathbf{F}_{2^{457}})$	88	xc4vlx200-10	58956	100	100.8	5.94
This work	$E(\mathbf{F}_{2^{457}})$	88	xc2vp100-6	42965	147	11.0	0.47
	$E(\mathbf{F}_{2^{457}})$	88	xc4vlx100-11	44223	215	7.52	0.33
Beuchat *et al.* [7]	$E(\mathbf{F}_{2^{459}})$	89	xc2vp20-6	8153	115	327	2.66
Beuchat *et al.* [7]	$E(\mathbf{F}_{3^{193}})$	89	xc2vp20-6	8266	90	298	2.46
This work[†]	$E(\mathbf{F}_{3^{193}})$	89	xc2vp100-6	46135[‡]	130	12.8	0.59
	$E(\mathbf{F}_{3^{193}})$	89	xc4vlx200-11	47260	179	9.33	0.44
Shu *et al.* [43]	$E(\mathbf{F}_{2^{557}})$	96	xc4vlx200-10	37931	66	675.5	25.62
This work	$E(\mathbf{F}_{2^{557}})$	96	xc4vlx200-11	55156	149	13.2	0.73
This work	$E(\mathbf{F}_{3^{239}})$	97	xc4vlx200-11	66631	179	11.5	0.77
This work	$E(\mathbf{F}_{2^{613}})$	100	xc4vlx200-11	62418	143	15.1	0.95
This work	$E(\mathbf{F}_{2^{691}})$	105	xc4vlx200-11	78874	130	18.8	1.48
This work[†]	$E(\mathbf{F}_{3^{313}})$	109	xc4vlx200-11	97105[‡]	159	16.9	1.64

[*] No final exponentiation, non-reduced pairing only.

[†] The inverse Frobenius map was preferred over the Frobenius map to compute the final exponentiation, as per Section III-B.3.

[‡] The design exceeds the FPGA's capacity: the η_T pairing and final-exponentiation coprocessors were placed-and-routed separately.

Our architectures are also much faster than software implementations. Mitsunari wrote a very careful multithreaded implementation of the η_T pairing over $\mathbf{F}_{3^{97}}$ and $\mathbf{F}_{3^{193}}$ [35]. He reported a computation time of 92 μs and 553 μs, respectively, on an Intel Core 2 Duo processor (2.66 GHz). Interestingly enough, his software library outperforms several hardware architectures proposed by other researchers for low levels of security. When we compare his results with our work, we note that the gap between software and hardware increases when considering larger values of m. The computation of the η_T pairing over $\mathbf{F}_{3^{193}}$ on a Virtex-4 LX FPGA with a medium speedgrade is for instance roughly fifty times faster than software. This speedup justifies the use of large FPGAs which are now available in servers and supercomputers such as the SGI Altix 4700 platform.

Kammler *et al.* [26] reported the first hardware implementation of the Optimal Ate pairing [46] over a Barreto–Naehrig (BN) curve [5], that is an ordinary curve defined over a prime field $\mathbf{F}_p$ with embedding degree $k = 12$. The proposed design is implemented with a 130 nm standard cell library and computes a pairing in 15.8 ms over a 256-bit BN curve. It is however difficult to make a fair comparison between our respective works since the level of security and the target technology are not the same.

B. Characteristic Two vs. Characteristic Three

It is worth noting that, in order to achieve the same level of security for the η_T pairing over supersingular curves in characteristics two and three, the extension degree m of $\mathbf{F}_{2^m}$ has to be larger than that of $\mathbf{F}_{3^{m'}}$. More precisely, we have the ratio

$$\frac{m}{m'} = \frac{3\log 3}{2\log 2} \approx 2.4,$$

since the embedding degree is 6 in characteristic three, against 4 in characteristic 2. This ratio also applies asymptotically to the number of iterations in Miller's algorithm, which is $(m+1)/2$ and $(m'+1)/2$, respectively.

However, the arithmetic over $\mathbf{F}_{2^{4m}}$ required for the computation of the pairing in characteristic two is much simpler that the arithmetic over $\mathbf{F}_{3^{6m'}}$: one iteration of Miller's algorithm requires only 7 multiplications over $\mathbf{F}_{2^m}$, against 17 multiplications over $\mathbf{F}_{3^{m'}}$ in the case of characteristic three. Coincidentally, the ratio between the two is also $17/7 \approx 2.4$.

Thus, although necessitating 2.4 times as many iterations as in characteristic three, the η_T pairing over $\mathbf{F}_{2^m}$ requires almost exactly as many products over the base field as the η_T pairing over $\mathbf{F}_{3^{m'}}$. Furthermore, a smaller extension degree m' compensates for the arithmetic over $\mathbf{F}_3$ being more expensive than that over $\mathbf{F}_2$.

That close similarity in terms of performances between characteristics two and three at a constant level of security, as hinted at by this short analysis, can actually be observed in the place-and-route results of our coprocessors (Figure 7), even though characteristic two appears to have a slight advantage for low security.

VI. Conclusion

We proposed novel architectures based on a parallel pipelined Karatsuba multiplier for the η_T pairing in characteristics two and three. The main design challenge we faced was to keep the pipeline continuously busy. Accordingly, we modified the scheduling of Miller's algorithm in order to introduce more parallelism in the pairing computation. We also presented a faster way to perform the final exponentiation by exploiting the linearity of the Frobenius map and/or taking advantage of a simpler inverse Frobenius map in certain cases. Both software and hardware implementations can benefit from these techniques.

To our knowledge, the implementation of our designs on several Xilinx FPGA devices improved both the computation time and the area–time trade-off of all the hardware pairing coprocessors previously published in the open literature [1], [7], [11], [19], [24], [28]–[31], [38]–[40], [42], [43].

However, as of today, the design of pairing accelerators providing a level of security equivalent to that of AES-128 remains a problem of major interest. Although Kammler *et al.* [26] proposed a first solution over a Barreto–Naehrig curve, several questions remain open. For instance, is it possible to achieve such a level of security in hardware with supersingular (hyper)elliptic curves at a reasonable cost in terms of computation time and circuit area? Since several protocols rely on such curves, it seems crucial to us to address this topic in a near future.

Another interesting direction for further work is to investigate the use of the hybrid operator (Figure 3) to compute the complete Tate pairing and not only the final exponentiation. From our experiments, this operator should offer a competitive balance between the area-efficient unified operators of [7] and the latency-oriented architectures presented here.

Acknowledgments

The authors would like to thank Nidia Cortez-Duarte along with the anonymous referees for their valuable comments. This work was supported by the Japan–France Integrated Action Program (Ayame Junior/Sakura).

The authors would also like to express their warmest thanks to Toñito for his tasty contribution to this paper. ¡Muchas gracias por tus riquísimos tacos!

References

[1] A. Barenghi, G. Bertoni, L. Breveglieri, and G. Pelosi. A FPGA coprocessor for the cryptographic Tate pairing over $\mathbf{F}_p$. In *Proceedings of the Fourth International Conference on Information Technology: New Generations (ITNG'08)*. IEEE Computer Society, 2008.

[2] P.S.L.M. Barreto. A note on efficient computation of cube roots in characteristic 3. Cryptology ePrint Archive, Report 2004/305, 2004.

[3] P.S.L.M. Barreto, S.D. Galbraith, C. Ó hÉigeartaigh, and M. Scott. Efficient pairing computation on supersingular Abelian varieties. *Designs, Codes and Cryptography*, 42:239–271, 2007.

[4] P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, number 2442 in Lecture Notes in Computer Science, pages 354–368. Springer, 2002.

[5] P.S.L.M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. Tavares, editors, *Selected Areas in Cryptography – SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2006.

[6] G. Bertoni, L. Breveglieri, P. Fragneto, and G. Pelosi. Parallel hardware architectures for the cryptographic Tate pairing. In *Proceedings of the Third International Conference on Information Technology: New Generations (ITNG'06)*. IEEE Computer Society, 2006.

[7] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, and F. Rodríguez-Henríquez. A comparison between hardware accelerators for the modified tate pairing over $\mathbf{F}_{2^m}$ and $\mathbf{F}_{3^m}$. In S.D. Galbraith and K.G. Paterson, editors, *Pairing-Based Cryptography – Pairing 2008*, number 5209 in Lecture Notes in Computer Science, pages 297–315. Springer, 2008.

[8] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, M. Shirase, and T. Takagi. Algorithms and arithmetic operators for computing the η_T pairing in characteristic three. *IEEE Transactions on Computers*, 57(11):1454–1468, November 2008.

[9] J.-L. Beuchat, N. Brisebarre, M. Shirase, T. Takagi, and E. Okamoto. A coprocessor for the final exponentiation of the η_T pairing in characteristic three. In C. Carlet and B. Sunar, editors, *Proceedings of Waifi 2007*, number 4547 in Lecture Notes in Computer Science, pages 25–39. Springer, 2007.

[10] J.-L. Beuchat, J. Detrey, N. Estibals, E. Okamoto, and F. Rodríguez-Henríquez. Hardware accelerator for the Tate pairing in characteristic three based on Karatsuba–Ofman multipliers. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, number 5747 in Lecture Notes in Computer Science, pages 225–239. Springer, 2009.

[11] J.-L. Beuchat, H. Doi, K. Fujita, A. Inomata, P. Ith, A. Kanaoka, M. Katouno, M. Mambo, E. Okamoto, T. Okamoto, T. Shiga, M. Shirase, R. Soga, T. Takagi, A. Vithanage, and H. Yamamoto. FPGA and ASIC implementations of the η_T pairing in characteristic three. *Computers and Electrical Engineering*, 36(1):73–87, January 2010.

[12] J.-L. Beuchat, E. López-Trejo, L. Martínez-Ramos, S. Mitsunari, and F. Rodríguez-Henríquez. Multi-core implementation of the Tate pairing over supersingular elliptic curves. In J.A. Garay, A. Miyaji, and A. Otsuka, editors, *Cryptology and Network Security – CANS 2009*, number 5888 in Lecture Notes in Computer Science, pages 413–432. Springer, 2009.

[13] R. Dutta, R. Barua, and P. Sarkar. Pairing-based cryptographic protocols: A survey. Cryptology ePrint Archive, Report 2004/64, 2004.

[14] I. Duursma and H.S. Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In C.S. Laih, editor, *Advances in Cryptology – ASIACRYPT 2003*, number 2894 in Lecture Notes in Computer Science, pages 111–123. Springer, 2003.

[15] H. Fan, J. Sun, M. Gu, and K.-Y. Lam. Overlap-free Karatsuba–Ofman polynomial multiplication algorithm. Cryptology ePrint Archive, Report 2007/393, 2007.

[16] K. Fong, D. Hankerson, J. López, and A. Menezes. Field inversion and point halving revisited. *IEEE Transactions on Computers*, 53(8):1047–1059, August 2004.

[17] S.D. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In C. Fieker and D.R. Kohel, editors, *Algorithmic Number Theory – ANTS V*, number 2369 in Lecture Notes in Computer Science, pages 324–337. Springer, 2002.

[18] E. Gorla, C. Puttmann, and J. Shokrollahi. Explicit formulas for efficient multiplication in $\mathbf{F}_{3^{6m}}$. In C. Adams, A. Miri, and M. Wiener, editors, *Selected Areas in Cryptography – SAC 2007*, number 4876 in Lecture Notes in Computer Science, pages 173–183. Springer, 2007.

[19] P. Grabher and D. Page. Hardware acceleration of the Tate pairing in characteristic three. In J.R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, number 3659 in Lecture Notes in Computer Science, pages 398–411. Springer, 2005.

[20] D. Hankerson, A. Menezes, and M. Scott. *Software Implementation of Pairings*, chapter 12, pages 188–206. Cryptology and Information Security Series. IOS Press, 2009.

[21] G. Hanrot and P. Zimmermann. A long note on Mulders' short product. *Journal of Symbolic Computation*, 37(3):391–401, 2004.

[22] F. Hess. Pairing lattices. In S.D. Galbraith and K.G. Paterson, editors, *Pairing-Based Cryptography – Pairing 2008*, number 5209 in Lecture Notes in Computer Science, pages 18–38. Springer, 2008.

[23] F. Hess, N. Smart, and F. Vercauteren. The Eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10):4595–4602, October 2006.

[24] J. Jiang. Bilinear pairing (Eta_T Pairing) IP core. Technical report, City University of Hong Kong – Department of Computer Science, May 2007.

[25] A. Joux. A one round protocol for tripartite Diffie–Hellman. In W. Bosma, editor, *Algorithmic Number Theory – ANTS IV*, number 1838 in Lecture Notes in Computer Science, pages 385–394. Springer, 2000.

[26] D. Kammler, D. Zhang, P. Schwabe, H. Scharwaechter, M. Langenberg, D. Auras, G. Ascheid, R. Leupers, R. Mathar, and H. Meyr. Designing an ASIP for cryptographic pairings over Barreto–Naehrig curves. Cryptology ePrint Archive, Report 2009/056, 2009.

[27] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Phys. Doklady (English Translation)*, 7(7):595–596, January 1963.

[28] M. Keller, T. Kerins, F. Crowe, and W.P. Marnane. FPGA implementation of a GF(2^m) Tate pairing architecture. In K. Bertels, J.M.P. Cardoso, and S. Vassiliadis, editors, *International Workshop on Applied Reconfigurable Computing (ARC 2006)*, number 3985 in Lecture Notes in Computer Science, pages 358–369. Springer, 2006.

[29] M. Keller, R. Ronan, W.P. Marnane, and C. Murphy. Hardware architectures for the Tate pairing over GF(2^m). *Computers and Electrical Engineering*, 33(5–6):392–406, 2007.

[30] T. Kerins, W.P. Marnane, E.M. Popovici, and P.S.L.M. Barreto. Efficient hardware for the Tate pairing calculation in characteristic three. In J.R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, number 3659 in Lecture Notes in Computer Science, pages 412–426. Springer, 2005.

[31] G. Kömürcü and E. Savaş. An efficient hardware implementation of the Tate pairing in characteristic three. In E. Prasolova-Førland and M. Popescu, editors, *Proceedings of the Third International Conference on Systems – ICONS 2008*, pages 23–28. IEEE Computer Society, 2008.

[32] H. Li, J. Huang, P. Sweany, and D. Huang. FPGA implementations of elliptic curve cryptography and Tate pairing over a binary field. *Journal of Systems Architecture*, 54:1077–1088, 2008.

[33] V.S. Miller. Short programs for functions on curves. Available at `http://crypto.stanford.edu/miller`, 1986.

[34] V.S. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004.

[35] S. Mitsunari. A fast implementation of η_T pairing in characteristic three on Intel Core 2 Duo processor. Cryptology ePrint Archive, Report 2009/032, 2009.

[36] S. Mitsunari, R. Sakai, and M. Kasahara. A new traitor tracing. *IEICE Trans. Fundamentals*, E85–A(2):481–484, Feb 2002.

[37] F. Rodríguez-Henríquez, G. Morales-Luna, and J. López. Low-complexity bit-parallel square root computation over $GF(2^m)$ for all trinomials. *IEEE Transactions on Computers*, 57(4):472–480, April 2008.

[38] R. Ronan, C. Murphy, T. Kerins, C. Ó hÉigeartaigh, and P.S.L.M. Barreto. A flexible processor for the characteristic 3 η_T pairing. *Int. J. High Performance Systems Architecture*, 1(2):79–88, 2007.

[39] R. Ronan, C. Ó hÉigeartaigh, C. Murphy, M. Scott, and T. Kerins. FPGA acceleration of the Tate pairing in characteristic 2. In *Proceedings of the IEEE International Conference on Field Programmable Technology – FPT 2006*, pages 213–220. IEEE, 2006.

[40] R. Ronan, C. Ó hÉigeartaigh, C. Murphy, M. Scott, and T. Kerins. Hardware acceleration of the Tate pairing on a genus 2 hyperelliptic curve. *Journal of Systems Architecture*, 53:85–98, 2007.

[41] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *2000 Symposium on Cryptography and Information Security (SCIS2000), Okinawa, Japan*, pages 26–28, January 2000.

[42] C. Shu, S. Kwon, and K. Gaj. FPGA accelerated Tate pairing based cryptosystem over binary fields. In *Proceedings of the IEEE International Conference on Field Programmable Technology – FPT 2006*, pages 173–180. IEEE, 2006.

[43] C. Shu, S. Kwon, and K. Gaj. Reconfigurable computing approach for Tate pairing cryptosystems over binary fields. *IEEE Transactions on Computers*, 58(9):1221–1237, September 2009.

[44] J.H. Silverman. *The Arithmetic of Elliptic Curves*. Number 106 in Graduate Texts in Mathematics. Springer-Verlag, 1986.

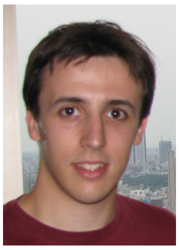
[45] L. Song and K.K. Parhi. Low energy digit-serial/parallel finite field multipliers. *Journal of VLSI Signal Processing*, 19(2):149–166, July 1998.

[46] F. Vercauteren. Optimal pairings. Cryptology ePrint Archive, Report 2008/096, 2008.

[47] L.C. Washington. *Elliptic Curves – Number Theory and Cryptography*. CRC Press, 2nd edition, 2008.

Jean-Luc Beuchat received his M.Sc. and his Ph.D. in computer science from the Swiss Federal Institute of Technology, Lausanne, in 1997 and 2001, respectively. He is an associate professor in the Graduate School of Systems and Information Engineering at the University of Tsukuba. His current research interests include computer arithmetic and cryptography.

Jérémie Detrey received his M.Sc. and his Ph.D. in computer science at the LIP of ÉNS Lyon (France) in 2003 and 2007 respectively, under the supervision of Florent de Dinechin and Jean-Michel Muller. He is now a junior researcher in the CARAMEL project-team at INRIA in Nancy (France). His research interests cover the various hardware aspects of computer arithmetic, with a special focus on finite fields, elliptic curves, and their applications in cryptography.



Nicolas Estibals received his M.Sc. in computer science from the ÉNS Lyon (France) in 2009. He is now a Ph.D. student in the CARAMEL project-team in Nancy (France), under the supervision of Jérémie Detrey and Pierrick Gaudry. His research interests are cryptography, computer arithmetic, and hardware development.

Eiji Okamoto received his B.S., M.S. and Ph.D degrees in electronics engineering from the Tokyo Institute of Technology in 1973, 1975 and 1978, respectively. He worked and studied communication theory and cryptography for NEC central research laboratories since 1978. From 1991 he was a professor at Japan Advanced Institute of Science and Technology, then at Toho University. Now he is a professor at Graduate School of Systems and Information Engineering, University of Tsukuba. His research interests are cryptography and information security. He is a member of IEEE and a coeditor-in-chief of Internatinal Journal of Information Security.

Francisco Rodríguez-Henríquez received the Ph.D. (2000) degree in electrical and computer engineering from Oregon State University, the M.Sc. (1992) degree in electrical and computer engineering from the National Institute of Astrophysics, Optics and Electronics (INAOE), Mexico and the B.Sc. (1989) degree in electrical engineering from the University of Puebla, Mexico. Currently, he is a professor (CINVESTAV-3B Researcher) at the Computer Science Department of CINVESTAV-IPN, in Mexico City, Mexico, which he joined in 2002. His major research interests are in cryptography, finite field arithmetic, and hardware implementation of cryptographic algorithms.