A Coprocessor for the Final Exponentiation of the η_T Pairing in Characteristic Three*

Jean-Luc Beuchat[1], Nicolas Brisebarre[2,3], Masaaki Shirase[4], Tsuyoshi Takagi[4], and Eiji Okamoto[1]

[1] Laboratory of Cryptography and Information Security, University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki, 305-8573, Japan
[2] LaMUSE, Université J. Monnet, 23, rue du Dr P. Michelon, F-42023 Saint-Étienne Cedex 02, France
[3] LIP/Arénaire (CNRS-ENS Lyon-INRIA-UCBL), ENS Lyon, 46 Allée d'Italie, F-69364 Lyon Cedex 07, France
[4] Future University-Hakodate, School of Systems Information Science, 116-2 Kamedanakano-cho, Hakodate, Hokkaido, 041-8655, Japan

Abstract. Since the introduction of pairings over (hyper)elliptic curves in constructive cryptographic applications, an ever increasing number of protocols based on pairings have appeared in the literature. Software implementations being rather slow, the study of hardware architectures became an active research area. Beuchat *et al.* proposed for instance a coprocessor which computes the characteristic three η_T pairing, from which the Tate pairing can easily be derived, in 33 μs on a Cyclone II FPGA. However, a final exponentiation is required to ensure a unique output value and the authors proposed to supplement their η_T pairing accelerator with a coprocessor for exponentiation. Thus, the challenge consists in designing the smallest possible piece of hardware able to perform this task in less than 33 μs on a Cyclone II device. In this paper, we propose a novel arithmetic operator implementing addition, cubing, and multiplication over $\mathbb{F}_{3^{97}}$ and show that a coprocessor based on a single such operator meets this timing constraint.

Keywords: η_T pairing, characteristic three, final exponentiation, hardware accelerator, FPGA.

1 Introduction

The first introduction of Weil and Tate pairings in cryptography was due to Menezes *et al.* [20] and Frey and Rück [11] who used them to attack the discrete logarithm problem on some classes of elliptic curves defined over finite fields. More recently, several cryptographic schemes based on those pairings have been proposed: identity-based encryption [6], short signature [8], and efficient broadcast encryption [7] to mention but a few.

* This work was supported by the New Energy and Industrial Technology Development Organization (NEDO), Japan.

This article aims at computing the η_T pairing in characteristic three in the case of supersingular elliptic curves over $\mathbb{F}_{3^m}$. These curves are necessarily of the form $E^b : y^2 = x^3 - x + b$, with $b \in \{-1, 1\}$. According to [3], curves over fields of characteristic three often offer the best possible ratio between security level and space requirements. Note that the η_T pairing easily relates to the Tate pairing [2]. In the following, we assume that $m = 97$ and $\mathbb{F}_{3^{97}}$ is given as $\mathbb{F}_3[x]/(x^{97} + x^{12} + 2)$. This choice is currently a good trade-off between security and computation time.

After previous works by Miller [21], Barreto *et al.* [3] and Galbraith *et al.* [12], an efficient algorithm for the characteristic three was proposed by Duursma and Lee [10]. That work was then extended by Kwon [19]. The introduction of the η_T pairing by Barreto *et al.* [2] led to a reduction by a factor two of the number of iterations compared to the approach by Duursma and Lee. Algorithm 1 summarizes the scheme proposed by Barreto *et al.* and uses the following notation: let $\ell > 0$ be an integer relatively prime to 3^m (i.e. to 3). The set $E^b(\mathbb{F}_{3^m})[\ell]$ groups all the points $P \in E^b(\mathbb{F}_{3^m})$ such that $\ell P = \mathcal{O}$, where $\mathcal{O}$ is the point at infinity. Let σ and $\rho \in \mathbb{F}_{3^{6m}}$ which satisfy $\sigma^2 = -1$ and $\rho^3 = \rho + b$. They help to define the distortion map introduced in [3].

Algorithm 1 Computation of η_T pairing in characteristic three [2].

Input: $P = (x_p, y_p)$ and $Q = (x_q, y_q) \in E^b(\mathbb{F}_{3^m})[\ell]$. The algorithm requires R_0 and $R_1 \in \mathbb{F}_{3^{6m}}$, as well as $r_0 \in \mathbb{F}_{3^m}$ for intermediate computations.
Output: $\eta_T(P, Q)$
1: **if** $b = 1$ **then**
2: $y_p \leftarrow -y_p$;
3: **end if**
4: $r_0 \leftarrow x_p + x_q + b$;
5: $R_0 \leftarrow -y_p r_0 + y_q \sigma + y_p \rho$;
6: **for** $i = 0$ to $(m-1)/2$ **do**
7: $r_0 \leftarrow x_p + x_q + b$;
8: $R_1 \leftarrow -r_0^2 + y_p y_q \sigma - r_0 \rho - \rho^2$;
9: $R_0 \leftarrow R_0 R_1$;
10: $x_p \leftarrow x_p^{1/3}$; $y_p \leftarrow y_p^{1/3}$;
11: $x_q \leftarrow x_q^3$; $y_q \leftarrow y_q^3$;
12: **end for**
13: Return R_0;

Algorithm 1 has the drawback of using inverse Frobenius maps (i.e. cube root in characteristic three). In [5], Beuchat *et al.* proposed a modified η_T pairing algorithm in characteristic three that does not require any cube root. However, to ensure a unique output value for the η_T pairing, we have to compute R_0^W, where $W = (3^{3m} - 1)(3^m + 1)(3^m + 1 - b3^{(m+1)/2})$ here. This operation, often referred to as *final exponentiation*, requires among other things a single inversion over $\mathbb{F}_{3^m}$ and multiplications over $\mathbb{F}_{3^{6m}}$ (note that pairing calculation in characteristic

two involves an inversion over $\mathbb{F}_{2^m}$ for final exponentiation). Pairing accelerators described in the literature follow two distinct strategies:

- Several researchers designed coprocessors for arithmetic over $\mathbb{F}_{3^m}$ (or $\mathbb{F}_{2^m}$) implementing both pairing calculation and final exponentiation [17,22,23,25]. This last operation is intrinsically sequential and there is unfortunately no parallelism at all when comes the time of inversion. It is therefore crucial to embed a fast inverter to avoid impacting the overall performance of the system. Reference [18] introduces for instance an efficient architecture for Extended Euclidean Algorithm (EEA) based inversion. To our best knowledge, the fastest coprocessor designed according to this philosophy computes $\eta_T(P,Q)^W$ over the field $\mathbb{F}_{3^{97}}$ in 179 μs (114 μs for the pairing calculation and 65 μs for the final exponentiation) on a Virtex-II Pro 100 Field-Programmable Gate Array (FPGA) [22].
- Consider the computation of the η_T pairing (Algorithm 1) and note that two coefficients of R_1 are null and another one is equal to -1. This observation allowed Beuchat *et al.* to design an optimized multiplier over $\mathbb{F}_{3^{97}}$ which is at the heart of a pairing accelerator computing $\eta_T(P,Q)$ in 33 μs [5]. It is worth noticing that the computation of the pairing requires 4849 clock cycles. Since Fermat's little theorem makes it possible to carry out inversion over $\mathbb{F}_{3^m}$ by means of multiplications and cubings, $\eta_T(P,Q)^W$ could be computed on such an accelerator. It seems however more attractive to supplement it with dedicated hardware for final exponentiation.

The challenge consists in designing the smallest possible processor able to compute a final exponentiation in less than 33 μs on a Cyclone II device. Our architecture is based on an innovative algorithm introduced by Shirase, Takagi, and Okamoto in [24]. We summarize this scheme in Section 2 and describe a novel arithmetic operator performing addition, subtraction, multiplication, and cubing over $\mathbb{F}_{3^{97}}$ (Section 3). We show that a coprocessor based on a single such processing element allows us to meet our timing constraint. Section 4 provides the reader with a comparison against previously published solutions.

2 Computation of the Final Exponentiation

Algorithm 2 describes a traditional way to perform final exponentiation [5]. Ronan *et al.* took for instance advantage of such a scheme to design their η_T pairing accelerator [22]. Shirase *et al.* proposed a novel algorithm based on the following remark [24]: let $X \in \mathbb{F}_{3^{6m}}$, then $X^{3^{3m}-1}$ belong to the torus $T_2(\mathbb{F}_{3^{3m}})$, a set introduced in [14]. Then they showed that the arithmetic in T_2 is cheaper, hence a significant gain in term of number of operations compared to Algorithm 2 (see Table 1). Algorithm 6 describes this final exponentiation scheme. It uses Algorithms 3 and 5.

Algorithm 3 involves an inversion over $\mathbb{F}_{3^{3m}}$. The tower field representation allows us to substitute this operation with 12 multiplications, 11 additions, and an inversion over $\mathbb{F}_{3^m}$ (see Appendix B for details). In order to keep the circuit

Algorithm 2 Raising $\eta_T(P,Q)$ to the W-th power ($b=1$) [5].

Input: $\eta_T(P,Q) \in \mathbb{F}_{3^{6m}}$. Thirteen variables u_i, $0 \leq i \leq 6$, and v_i, $0 \leq i \leq 5$ belonging to $\mathbb{F}_{3^{6m}}$ store intermediate results.
Output: $\eta_T(P,Q)^W \in \mathbb{F}_{3^{6m}}$
1: $u_0 \leftarrow \eta_T(P,Q)$;
2: **for** $i = 1$ to 5 **do**
3: $\quad u_i \leftarrow u_{i-1}^{3^m}$;
4: **end for**
5: $u_1 \leftarrow u_1^2$;
6: $u_4 \leftarrow u_4^2$;
7: $v_0 \leftarrow \eta_T(P,Q)^{3^{(m+1)/2}}$;
8: **for** $i = 1$ to 4 **do**
9: $\quad v_i \leftarrow v_{i-1}^{3^m}$;
10: **end for**
11: $u_6 \leftarrow v_0 \cdot v_1 \cdot u_3 \cdot u_4 \cdot u_5$;
12: $v_5 \leftarrow u_0 \cdot u_1 \cdot u_2 \cdot v_3 \cdot v_4$;
13: Return $u_0 \leftarrow u_6/v_5$;

Algorithm 3 Computation of $X^{3^{3m}-1}$.

Input: $X = x_0 + x_1\sigma + x_2\rho + x_3\sigma\rho + x_4\rho^2 + x_5\sigma\rho^2 \in \mathbb{F}^*_{3^{6m}}$.
Output: $X^{3^{3m}-1} \in T_2(\mathbb{F}_{3^{3m}})$
1: $\tau_0 \leftarrow (x_0 + x_2\rho + x_4\rho^2)^2$;
2: $\tau_1 \leftarrow (x_1 + x_3\rho + x_5\rho^2)^2$;
3: $\tau_2 \leftarrow (x_0 + x_2\rho + x_4\rho^2)(x_1 + x_3\rho + x_5\rho^2)$;
4: $Y \leftarrow \dfrac{(\tau_0 - \tau_1) + \tau_2\sigma}{\tau_0 + \tau_1}$;
5: Return Y;

area as small as possible, we suggest to perform inversion according to Fermat's little theorem and Itoh and Tsujii's work [16]. Since $m = 97$, inversion requires 9 multiplications and 96 cubings over $\mathbb{F}_{3^{97}}$ (Algorithm 4, see Appendix A for a proof of correctness). Therefore, final exponentiation requires 87 multiplications, 390 cubings, and 477 additions over $\mathbb{F}_{3^{97}}$ (see Appendix B for details about the number of operations over $\mathbb{F}_{3^{97}}$ involved in the final exponentiation). Array multipliers processing D coefficients of an operand at each clock cycle are often at the heart of pairing accelerators (see Section 3.2). In [5], authors suggest to consider $D = 3$ coefficients and multiplication over $\mathbb{F}_{3^{97}}$ involves $\lceil \frac{97}{3} \rceil = 33$ clock cycles. Since addition and cubing are rather straightforward operations, they are carried out in a single clock cycle. Therefore, considering such parameters, final exponentiation requires $477 + 390 + 33 \cdot 87 = 3738$ clock cycles. Note that additional clock cycles are necessary to load and store intermediate results. However, this overhead should be smaller than 10% and a coprocessor embedding a multiplier, an adder/subtracter, as well as a cubing unit should perform this task in less than 4200 clock cycles. It is therefore possible to supplement the η_T pairing accelerator described in [5] (4849 clock cycles) with such a simple processing unit.

Algorithm 4 Inversion over $\mathbb{F}_{3^{97}}$.

Input: $a \in \mathbb{F}_{3^{97}}$
Output: $a^{-1} \in \mathbb{F}_{3^{97}}$
1: $y_0 \leftarrow a$;
2: **for** $i = 0$ to 5 **do**
3: $\quad z_i \leftarrow y_i^{3^{2^i}}$;
4: $\quad y_{i+1} \leftarrow y_i z_i$;
5: **end for**
6: $z_6 \leftarrow y_6^{3^{32}}$;
7: $y_7 \leftarrow y_5 z_6$;
8: $y_8 \leftarrow y_7^2$;
9: $y_9 \leftarrow y_8^3$;
10: Return $y_0 y_9$;

Algorithm 5 Computation of X^{3^m+1} in the torus $T_2(\mathbb{F}_{3^{3m}})$.

Input: $X \in T_2(\mathbb{F}_{3^{3m}})$
Output: $X^{3^m+1} \in T_2(\mathbb{F}_{3^{3m}})$
1: $z_0 \leftarrow x_0 x_4, z_1 \leftarrow x_1 x_5, z_2 \leftarrow x_2 x_4, z_3 \leftarrow x_3 x_5$;
2: $z_4 \leftarrow (x_0 + x_1)(x_4 - x_5)$;
3: $z_5 \leftarrow x_1 x_2, z_6 \leftarrow x_0 x_3$;
4: $z_7 \leftarrow (x_0 + x_1)(x_2 + x_3)$;
5: $z_8 \leftarrow (x_2 + x_3)(x_4 - x_5)$;
6: $y_0 \leftarrow 1 + z_0 + z_1 - bz_2 - bz_3$;
7: $y_1 \leftarrow z_1 + z_4 + bz_5 - z_0 - bz_6$;
8: $y_2 \leftarrow z_7 - z_2 - z_3 - z_5 - z_6$;
9: $y_3 \leftarrow z_3 + z_8 + bz_0 - z_2 - bz_1 - bz_4$;
10: $y_4 \leftarrow bz_2 + bz_3 + bz_7 - bz_5 - bz_6$;
11: $y_5 \leftarrow bz_3 + bz_8 - bz_2$;
12: Return $Y = (y_0 + y_2\rho + y_4\rho^2) + (y_1 + y_3\rho + y_5\rho^2)\sigma$;

3 Hardware Implementation

This section describes the implementation of Algorithm 6 on a Cyclone II EP2C35F672C6 FPGA whose smallest unit of configurable logic is called Logic Element (LE). Each LE includes a 4-input Look-Up Table (LUT), carry logic, and a programmable register. A Cyclone II EP2C35F672C6 device contains for instance 33216 LEs. Readers who are not familiar with Cyclone II devices should refer to [1] for further details. After studying addition, multiplication, and cubing over $\mathbb{F}_{3^m}$, we propose a novel arithmetic operator able to perform these three operations and describe the architecture of a final exponentiation coprocessor based on such a processing element.

3.1 Addition and Subtraction over $\mathbb{F}_{3^m}$

Since they are performed component-wise, addition and subtraction over $\mathbb{F}_{3^m}$ are rather straightforward operations. Each element of $\mathbb{F}_3$ is encoded by two bit and addition modulo three on a Cyclone II FPGA requires two 4-input LUTs.

Algorithm 6 Final exponentiation of η_T pairing [24].

Input: $X = x_0 + x_1\sigma + x_2\rho + x_3\sigma\rho + x_4\rho^2 + x_5\sigma\rho^2 \in \mathbb{F}_{3^{6m}}^*$.
Output: $X^{(3^{3m}-1)(3^m+1)(3^m+1-b3^{(m+1)/2})}$
1: $Y \leftarrow X^{3^{3m}-1}$ (Algorithm 3);
2: $Y \leftarrow Y^{3^m+1}$ (Algorithm 5);
3: $Z \leftarrow Y$;
4: **for** $i = 0$ to $(m-1)/2$ **do**
5: $\quad Z \leftarrow Z^3$;
6: **end for**
7: $Y \leftarrow Y^{3^m+1}$ (Algorithm 5);
8: **if** $b = 1$ **then**
9: $\quad$ Return $Y \cdot (z_0 - z_1\sigma + z_2\rho - z_3\sigma\rho + z_4\rho^2 - z_5\sigma\rho^2)$;
10: **else**
11: $\quad$ Return YZ;
12: **end if**

Table 1. Comparison of final exponentiation algorithms (number of operations).

Algorithm	Additions over $\mathbb{F}_{3^{97}}$	Cubings over $\mathbb{F}_{3^{97}}$	Multiplications over $\mathbb{F}_{3^{97}}$
Algorithm 2	1022	390	243
Algorithm 6	477	390	87

Negation over $\mathbb{F}_3$ is performed by multiplying an operand by two. Note that the computation of the y_i's in Algorithm 5 involves the addition of up to six operands. This motivates the design of the accumulator illustrated on Figure 1a.

3.2 Multiplication over $\mathbb{F}_{3^m}$

Three families of algorithms allow one to compute $a(x)b(x) \bmod f(x)$. In parallel-serial schemes, a single coefficient of the multiplier $a(x)$ is processed at each step. This leads to small operands performing a multiplication in m steps. Parallel multipliers compute a degree-$(2m-2)$ polynomial and carry out a final modular reduction. They achieve a higher throughput at the price of a larger circuit area. By processing D coefficients of an operand at each clock cycle, array multipliers, introduced by Song and Parhi in [26], offer a good trade-off between computation time and circuit area and are at the heart of several pairing coprocessors (see for instance [5, 13, 17, 22, 23, 25]). Among the many array multipliers described in the literature (see for instance [15, 25]), the one proposed by Shu, Kwon, and Gaj [25] (Algorithm 7) is a good candidate for FPGA implementation when $f(x)$ is a trinomial [4]. Figure 1b illustrates the architecture of an operator processing $D = 3$ coefficients at each clock cycle. It mainly consists of three Partial Product Generators (PPG), three modulo $f(x)$ reduction units, a multioperand adder, and registers to store operands and intermediate results. Five bits make it possible to control this operator.

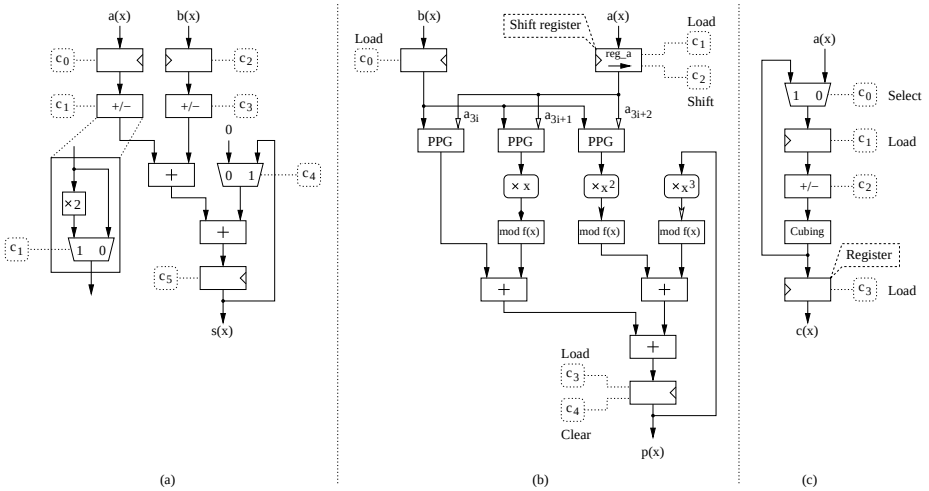

Fig. 1. Arithmetic operators over $\mathbb{F}_{3^m}$. (a) Addition/subtraction of two operands and accumulation. (b) Multiplication [25] ($D = 3$ coefficients of $a(x)$ processed at each clock cycle). (c) Cubing.

In the following, we will focus on multiplication over $\mathbb{F}_{3^{97}}$ and assume that $D = 3$ (i.e. multiplication requires 33 clock cycles). With such parameters, the first iteration of Algorithm 7 is defined as follows: $t(x) \leftarrow a_{96}b(x) + (a_{97}xb(x)) \bmod f(x) + (a_{98}x^2b(x)) \bmod f(x)$. To ensure a correct result, we have to guarantee that $a_{97} = a_{98} = 0$. Therefore, the shift register stores a degree-98 polynomial whose two most significant coefficients are set to zero.

Algorithm 7 Multiplication over $\mathbb{F}_{3^m}$ [25].

Input: A degree-m monic polynomial $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1x + f_0$ and two degree-$(m-1)$ polynomials $a(x)$ and $b(x)$. A parameter D which defines the number of coefficients of $a(x)$ processed at each clock cycle. The algorithm requires a degree-$(m-1)$ polynomial $t(x)$ for intermediate computations.

Output: $p(x) = a(x)b(x) \bmod f(x)$

1: $p(x) \leftarrow 0$;
2: **for** i from $\lceil m/D \rceil - 1$ downto 0 **do**
3: $\quad t(x) \leftarrow \sum_{j=0}^{D-1} \left(a_{Di+j}x^j b(x) \right) \bmod f(x)$;
4: $\quad p(x) \leftarrow t(x) + (x^D p(x) \bmod f(x))$;
5: **end for**

3.3 Cubing over $\mathbb{F}_{3^m}$

Since we set $f(x) = x^{97} + x^{12} + 2$, cubing over $\mathbb{F}_{3^m}$ is a pretty simple arithmetic operation: a GP/PARI program provides us with a closed formula:

$$\begin{array}{lll} b_0 = a_{93} + a_{89} + a_0, & b_1 = a_{65} + 2a_{61}, & b_2 = a_{33}, \\ b_3 = a_{94} + a_{90} + a_1, & \ldots = \ldots, & b_{94} = a_{96} + a_{92} + a_{88}, \\ b_{95} = a_{64} + 2a_{60}, & b_{96} = a_{32}. & \end{array} \quad (1)$$

The most complex operation involved in cubing is therefore the addition of three elements belonging to $\mathbb{F}_3$. Recall that inversion over $\mathbb{F}_{3^{97}}$ involves successive cubing operations. Since storing intermediate results in memory would be too time consuming, our cubing unit should include a feedback mechanism to efficiently implement Algorithm 4. Furthermore, cubing over $\mathbb{F}_{3^{6m}}$ requires the computation of $-y_i^3$, where $y_i \in \mathbb{F}_{3^m}$ (see Appendix B for details). These considerations suggest the design of the operator depicted by Figure 1c.

Place-and-Route Results. These three arithmetic operators were captured in the VHDL language and prototyped on an Altera Cyclone II EP2C35F672C6 device. Both synthesis and place-and-route steps were performed with Quartus II 6.0 Web Edition (Table 2). A naive solution would then consist in connecting the outputs of these operators to the memory blocks by means of a three-input multiplexer controlled by two bits. Such an arithmetic and logic unit (ALU) requires 3308 Logic Elements (LEs) and final exponentiation can be carried out within 4082 clock cycles, thus meeting our timing constraint. Cubings only occur in inversion (Algorithm 4) and in the computation of Z (step 5 of Algorithm 6). Due to the sequential nature of these algorithms, both multiplier and adder remain idle at that time. The same observation can be made for additions and multiplications: most of the time, only a single arithmetic operator is processing data. Is it therefore possible to save hardware resources by designing an operator able to perform addition, multiplication, and cubing over $\mathbb{F}_{3^{97}}$?

Table 2. Arithmetic operators over $\mathbb{F}_{3^{97}}$ on a Cyclone II FPGA.

	Addition/ subtraction	**Multiplication (D = 3)**	**Cubing**	**ALU**
Area [LEs]	970	1375	668	3308
Control [bits]	6	5	4	17

3.4 An Operator for Multiplication, Addition, and Cubing over $\mathbb{F}_{3^{97}}$

Consider again the closed formula for cubing over $\mathbb{F}_3[x]/(x^{97} + x^{12} + 2)$ (Equation (1)). We can for instance write $b_1 = a_{65} + a_{61} + a_{61}$ and $b_2 = a_{33} + 0 + 0$.

Let us define $c_0(x)$, $c_1(x)$, and $c_2(x) \in \mathbb{F}_{3^{97}}$ such that:

$$\begin{aligned} c_0(x) &= a_{93} + a_{65}x + a_{33}x^2 + \ldots + a_{88}x^{94} + a_{64}x^{95} + a_{32}x^{96}, \\ c_1(x) &= a_{89} + a_{61}x + 0 \cdot x^2 + \ldots + a_{92}x^{94} + a_{60}x^{95} + 0 \cdot x^{96}, \\ c_2(x) &= a_0 + a_{61}x + 0 \cdot x^2 + \ldots + a_{96}x^{94} + a_{60}x^{95} + 0 \cdot x^{96}. \end{aligned} \tag{2}$$

Then, $a(x)^3 = c_0(x) + c_1(x) + c_2(x)$ and cubing requires the addition of three operands as well as some wiring to compute the $c_i(x)$'s. Remember now that our array multiplier (Figure 1b) embeds a three-operand adder and an accumulator, which also makes possible the implementation of addition and cubing. Furthermore, since negation over $\mathbb{F}_{3^m}$ consists in multiplying the operand by two, PPGs can perform this task.

These considerations suggest the design of a three-input arithmetic operator for addition, accumulation, cubing, and multiplication over $\mathbb{F}_{3^{97}}$ (Figure 2). In order to compute the product $a(x)b(x) \bmod f(x)$, it suffices to load $a(x)$ in register R0, and $b(x)$ in registers R1 and R2. Addition and cubing are slightly more complex and we will consider a toy example to illustrate how our operator works. Let us assume we have to compute $-a(x) + b(x)$ and $a(x)^3$, where $a(x)$, $b(x) \in \mathbb{F}_{3^{97}}$. We respectively load $a(x)$ and $b(x)$ in registers R2 and R1 and define a control word stored in R0 so that $\mathrm{d0}_{3i} = 2$, $\mathrm{d0}_{3i+1} = 1$, and $\mathrm{d0}_{3i+2} = 0$. We will thus compute $(2a(x) + b(x) + 0 \cdot a(x)) \bmod f(x) = (-a(x) + b(x)) \bmod f(x)$. For cubing, we load $a(x)$ in both registers R1 and R2. If $\mathrm{d0}_{3i} = \mathrm{d0}_{3i+1} = \mathrm{d0}_{3i+2} = 1$, then our operator implements Equation (2) and returns $a(x)^3$. Thus, register R0 stores either an operand of a multiplication or a control word for up to 33 successive additions and cubings (recall that this shift register stores a degree-98 polynomial and that three coefficients are processed at each clock cycle). Place-and-route results indicate that this processing element requires 2676 LEs instead of 3308 LEs with the naive approach. Furthermore, this architecture allows one to reduce the number of control bits from 17 (see Table 2) to 11.

3.5 Architecture of the Coprocessor

Figure 3 describes the architecture of our coprocessor which embeds a single arithmetic unit performing addition, accumulation, cubing, or multiplication over $\mathbb{F}_{3^{97}}$. Intermediate results (194 bits) and control words for additions and cubings (198 bits) are stored in 64 registers implemented by a dual-port RAM (13 Cyclone II M4K memory blocks). An element of $\mathbb{F}_{3^{6m}}$ returned by the η_T pairing accelerator is sequentially loaded in the RAM. Then, a simple Finite State Machine and a ROM generate all control signals required to perform the final exponentiation according to Algorithm 6. Each instruction stored in the ROM consists of four fields: a control word which specifies the functionality of the processing element, addresses and write enable signals for both ports of the RAM, and a counter which indicates how many times the instruction must be repeated. Inversion over $\mathbb{F}_{3^{97}}$ involves for instance consecutive cubings (Algorithm 4). This approach allows one to execute them with a single instruction.

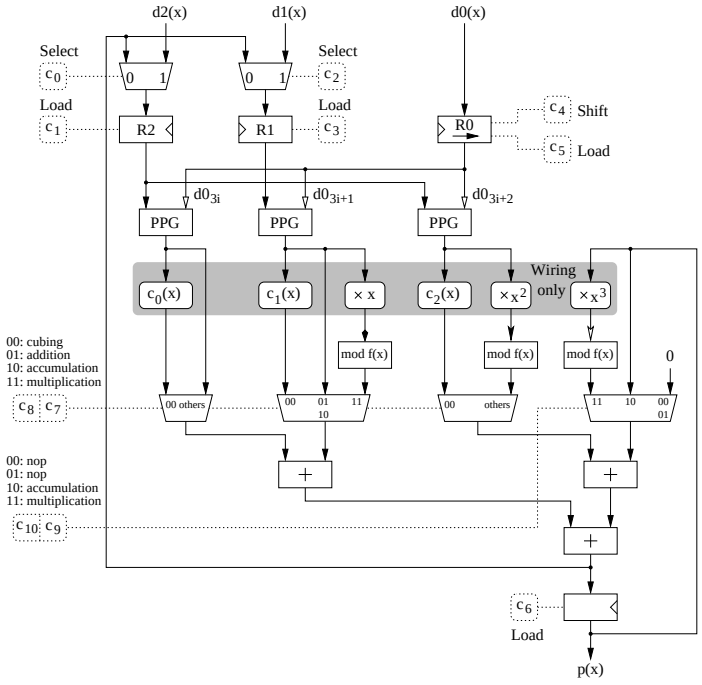

Fig. 2. Addition, accumulation, cubing, and multiplication over $\mathbb{F}_{3^{97}}$.

The implementation of Algorithm 6 on this coprocessor requires 658 instructions which are executed within 4082 clock cycles. Ten control words, stored in the dual-port RAM, manage all additions and cubings involved in the computation of the final exponentiation.

4 Results and Comparisons

Our final exponentiation coprocessor was implemented on an Altera Cyclone II EP2C35F672C6 FPGA. According to place-and-route tools, this architecture requires 2787 LEs and 21 M4K memory blocks. Since the maximum frequency is 159 MHz, an exponentiation is computed within 26 μs and our timing constraint is fully met. It is worth noticing that the inversion over $\mathbb{F}_{3^{97}}$ based on the EEA described in [18] occupies 3422 LEs [27] and needs $2m = 194$ clock cycles. Our approach based on Fermat's little theorem (Algorithm 4) performs the same operation in 394 clock cycles. Therefore, introducing specific hardware for inversion would double the circuit area while reducing the calculation time by only 5%.

To our best knowledge, the only η_T pairing accelerator in characteristic three implementing final exponentiation was proposed by Ronan *et al.* in [22]. In order to easily study the trade-off between calculation time and circuit area, they wrote a C program which automatically generates a VHDL description of a processor and its control according to the number of multipliers to be included and D. The

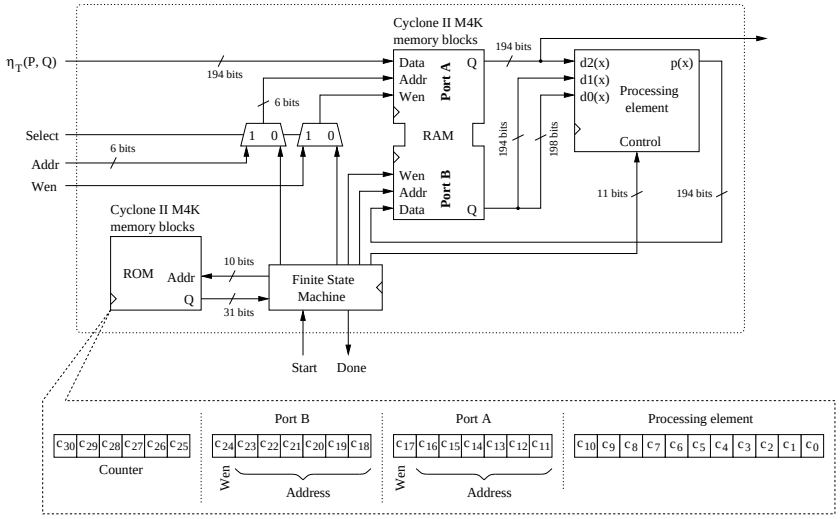

Fig. 3. Architecture of the coprocessor for final exponentiation.

ALU also embeds an adder, a subtracter, a cubing unit, and an inversion unit. The most attractive architecture contains three multipliers processing $D = 8$ coefficients at each clock cycle. It computes $\eta_T(P, Q)$ in 114 μs and requires 65 μs to perform final exponentiation according to Algorithm 2 on a Xilinx Virtex-II Pro 100 FPGA (clock frequency: 70.4 MHz). This architecture requires 10000 slices of a Virtex-II Pro FPGA. Each slice of this FPGA family features two 4-input LUTs, carry logic, wide function multiplexers, and two storage elements. Let us assume that Xilinx design tools try to utilize both LUTs of a slice as often as possible (i.e. area optimization). Under this hypothesis, we consider that a slice is roughly equivalent to two LEs and our coprocessor is seven times smaller than the one described in [22].

Recall that Algorithm 6 allows one to divide by 2.8 the number of multiplications over $\mathbb{F}_{3^{97}}$ (Table 1). Therefore, our coprocessor would compute final exponentiation according to Algorithm 2 in around $26 \times 2.8 = 72.8\,\mu$s.

The η_T pairing accelerator described in [5] returns $\eta_T(P, Q)$ in 33 μs using 14895 LEs and 13 memory blocks. We can therefore estimate the total area of a coprocessor computing $\eta_T(P, Q)^W$ to 18000 LEs and 34 M4K memory blocks. Thus, with roughly the same amount of configurable logic, we should achieve five times faster η_T pairing calculation than Ronan *et. al.*

5 Concluding Remarks

We proposed a novel arithmetic operator performing addition, accumulation, cubing, and addition over $\mathbb{F}_{3^{97}}$ and designed a coprocessor able to compute the final exponentiation of the η_T pairing in 26 μs on a Cyclone II FPGA. Since the calculation time of the η_T pairing accelerator described in [5] is 33 μs, we

can pipeline both architectures without impacting the overall performance of the system and our approach allows one to divide by five the calculation time of $\eta_T(P,Q)^W$ compared to the best implementation reported in the open literature [22]. Since different FPGA families are involved, it is unfortunately difficult to provide the reader with a fair area comparison. A rough estimate indicates that our coprocessor requires the same hardware resources.

Another important result is that hardware for inversion is not necessary for the calculation of the η_T pairing on a characteristic three elliptic curve over $\mathbb{F}_{3^{97}}$: our final exponentiation coprocessor meets our timing constraint with an algorithm based on Fermat's little theorem. Furthermore, the architecture proposed in [22] computes $\eta_T(P,Q)^W$ in 15113 clock cycles. Since an inverter based on the EEA saves only 200 clock cycles and that no other operation can be performed in parallel, we believe it is not interesting to include dedicated hardware for this operation.

The approach introduced in this paper to design our arithmetic operator offers several further research topics we plan to study in the future. It would for instance be interesting to implement the computation of both pairing and final exponentiation with the coprocessor described in this paper. Such an architecture could for instance be attractive for ASIC implementations. Another open question is if our operator is able to carry out other functions (e.g. cube root) or if this design methodology works for other irreducible polynomials and finite fields. Finally, note that our processor always performs the same operation: at each clock cycle, the content of the shift register is updated (load or shift operation), and a sum of three partial products is computed. Pairing operations could therefore be split into atomic blocks (side-channel atomicity [9]) and such architectures could prevent simple side-channel attacks.

References

1. Altera. *Cyclone II Device Handbook*, 2006. Available from Altera's web site (`http://altera.com`).
2. 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(3):239–271, 2007.
3. 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.
4. J.-L. Beuchat, T. Miyoshi, Y. Oyama, and E. Okamoto. Multiplication over $\mathbb{F}_{p^m}$ on FPGA: A survey. In P. C. Diniz, E. Marques, K. Bertels, M. M. Fernandes, and J. M. P. Cardoso, editors, *Reconfigurable Computing: Architectures, Tools and Applications – Proceedings of ARC 2007*, number 4419 in Lecture Notes in Computer Science, pages 214–225. Springer, 2007.
5. J.-L. Beuchat, M. Shirase, T. Takagi, and E. Okamoto. An algorithm for the η_T pairing calculation in characteristic three and its hardware implementation. In *Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, 2007. To appear.

6. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, number 2139 in Lecture Notes in Computer Science, pages 213–229. Springer, 2001.
7. D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, number 3621 in Lecture Notes in Computer Science, pages 258–275. Springer, 2005.
8. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, number 2248 in Lecture Notes in Computer Science, pages 514–532. Springer, 2001.
9. B. Chevallier-Mames, M. Ciet, and M. Joye. Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. *IEEE Transactions on Computers*, 53(6):760–768, June 2004.
10. 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.
11. G. Frey and H.-G. Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Math. Comp.*, 62(206):865–874, April 1994.
12. 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.
13. 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.
14. R. Granger, D. Page, and M. Stam. On small characteristic algebraic tori in pairing-based cryptography. *LMS Journal of Computation and Mathematics*, 9:64–85, March 2006. Available from `http://www.lms.ac.uk/jcm/9/lms2004-025/`.
15. J. Guajardo, T. Güneysu, S. Kumar, C. Paar, and J. Pelzl. Efficient hardware implementation of finite fields with applications to cryptography. *Acta Applicandae Mathematicae*, 93(1–3):75–118, September 2006.
16. T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. *Information and Computation*, 78:171–177, 1988.
17. 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.
18. T. Kerins, E. Popovici, and W. Marnane. Algorithms and architectures for use in FPGA implementations of identity based encryption schemes. In J. Becker, M. Platzner, and S. Vernalde, editors, *Field-Programmable Logic and Applications*, number 3203 in Lecture Notes in Computer Science, pages 74–83. Springer, 2004.
19. S. Kwon. Efficient Tate pairing computation for supersingular elliptic curves over binary fields. Cryptology ePrint Archive, Report 2004/303, 2004.
20. A. Menezes, T. Okamoto, and S. A. Vanstone. Reducing elliptic curves logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, September 1993.
21. V. S. Miller. Short programs for functions on curves. Unpublished manuscript available at http://crypto.stanford.edu/miller/miller.pdf, 1986.

22. R. Ronan, C. Ó hÉigeartaigh, C. Murphy, T. Kerins, and P. S. L. M. Barreto. Hardware implementation of the η_T pairing in characteristic 3. Cryptology ePrint Archive, Report 2006/371, 2006.
23. R. Ronan, C. Ó hÉigeartaigh, C. Murphy, M. Scott, T. Kerins, and W.P. Marnane. An embedded processor for a pairing-based cryptosystem. In *Proceedings of the Third International Conference on Information Technology: New Generations (ITNG'06)*. IEEE Computer Society, 2006.
24. M. Shirase, T. Takagi, and E. Okamoto. Some efficient algorithms for the final exponentiation of η_T pairing. In *3rd Information Security Practice and Experience Conference – ISPEC 2007*, Lecture Notes in Computer Science. Springer, 2007.
25. C. Shu, S. Kwon, and K. Gaj. FPGA accelerated Tate pairing based cryptosystem over binary fields. In *Proceedings of 2006 IEEE International Conference on Field Programmable Technology (FPT 2006)*, pages 173–180. IEEE Computer Society, 2006.
26. 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.
27. A. Vithanage. Personal communication.

A Proof of Correctness of Algorithm 4

Let $a \in \mathbb{F}_{3^{97}}$. According to Fermat's little theorem, $a^{-1} = a^{3^{97}-2}$. Note that the ternary representation of $3^{97}-2$ is $(\underbrace{22\ldots22}_{96\times}1)_3$. In order to prove the correctness of Algorithm 4, it suffices to show that $y_9 = a^k$, where $k = (\underbrace{22\ldots22}_{96\times}0)_3$:

$$
\begin{array}{lll}
z_0 = y_0^3 = a^{(10)_3}, & y_1 = a^{(11)_3}, & z_1 = y_1^{3^2} = a^{(1100)_3}, \\
y_2 = a^{(1111)_3}, & z_2 = y_2^{3^4} = a^{(11110000)_3}, & y_3 = a^{(11111111)_3}, \\
z_3 = y_3^{3^8} = a^{(\overbrace{1\ldots1}^{8\times}\overbrace{0\ldots0}^{8\times})_3}, & y_4 = a^{(\overbrace{1\ldots1}^{16\times})_3}, & z_4 = y_4^{3^{16}} = a^{(\overbrace{1\ldots1}^{16\times}\overbrace{0\ldots0}^{16\times})_3}, \\
y_5 = a^{(\overbrace{1\ldots1}^{32\times})_3}, & z_5 = y_5^{3^{32}} = a^{(\overbrace{1\ldots1}^{32\times}\overbrace{0\ldots0}^{32\times})_3}, & y_6 = a^{(\overbrace{1\ldots1}^{64\times})_3}, \\
z_6 = y_6^{3^{32}} = a^{(\overbrace{1\ldots1}^{64\times}\overbrace{0\ldots0}^{32\times})_3}, & y_7 = y_5 z_6 = a^{(\overbrace{1\ldots1}^{96\times})_3}, & y_8 = y_7^2 = a^{(\overbrace{2\ldots2}^{96\times})_3}, \\
y_9 = y_8^3 = a^{(\overbrace{2\ldots20}^{96\times})_3} & &
\end{array}
$$

Then, Algorithm 4 returns $y_0 y_9 = a^{(\overbrace{22\ldots221}^{96\times})_3} = a^{3^{97}-2}$.

B Arithmetic over $\mathbb{F}_{3^{2m}}$, $\mathbb{F}_{3^{3m}}$, and $\mathbb{F}_{3^{6m}}$

This Appendix summarizes classical algorithms for arithmetic over $\mathbb{F}_{3^{2m}}$, $\mathbb{F}_{3^{3m}}$, and $\mathbb{F}_{3^{6m}}$. Proofs of correctness of such algorithms are for instance provided in [17]. In order to compute the number of operations over $\mathbb{F}_{3^m}$, we assume that

the ALU is able to compute $a_i a_j$, $\pm a_i \pm a_j$ and $\pm a_i^3$, where a_i and $a_j \in \mathbb{F}_{3^m}$. We consider the case where the elliptic curve is given by $y^2 = x^3 - x + 1$ (i.e. $b = 1$ and $\rho^3 = \rho + 1$).

Multiplication over $\mathbb{F}_{3^{2m}}$. Let $A = a_0 + a_1\sigma$ and $B = b_0 + b_1\sigma$, where a_0, a_1, b_0, and $b_1 \in \mathbb{F}_{3^m}$. The product $AB = (a_0b_0 - a_1b_1) + ((a_0+a_1)(b_0+b_1) - a_0b_0 - a_1b_1)\sigma$ requires 3 multiplications and 5 additions (or subtractions) over $\mathbb{F}_{3^m}$.

Multiplication over $\mathbb{F}_{3^{3m}}$. Assume that $A = a_0 + a_1\rho + a_2\rho^2$ and $B = b_0 + b_1\rho + b_2\rho^2$, where $a_i, b_i \in \mathbb{F}_{3^m}$, $0 \leq i \leq 2$. The product $C = AB$ is then given by:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} (a_1 + a_2)(b_1 + b_2) + a_0b_0 - a_1b_1 - a_2b_2 \\ (a_0 + a_1)(b_0 + b_1) + (a_1 + a_2)(b_1 + b_2) - a_0b_0 + a_1b_1 \\ (a_0 + a_2)(b_0 + b_2) + a_1b_1 - a_0b_0 \end{bmatrix}.$$

This operation requires 6 multiplications and 14 additions (or subtractions) over $\mathbb{F}_{3^m}$.

Inversion over $\mathbb{F}_{3^{3m}}$. Let $A = a_0 + a_1\rho + a_2\rho^2$, where $a_i \in \mathbb{F}_{3^m}$, $0 \leq i \leq 2$. The inverse C of A is the given by:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = d^{-1} \begin{bmatrix} a_0^2 - (a_1^2 - a_2^2) - a_2(a_0 + a_1) \\ -a_0a_1 + a_2^2 \\ a_1^2 - a_2^2 - a_0a_2 \end{bmatrix},$$

where $d = a_0^2(a_0 - a_2) + a_1^2(-a_0 + a_1) + a_2^2(-(-a_0 + a_1) + a_2)$. This operation involves 12 multiplications, 11 additions (or subtractions), and 1 inversion over $\mathbb{F}_{3^m}$.

Multiplication over $\mathbb{F}_{3^{6m}}$. Let $A = \underbrace{a_0 + a_1\sigma}_{\tilde{a}_0} + (\underbrace{a_2 + a_3\sigma}_{\tilde{a}_1})\rho + (\underbrace{a_4\rho^2 + a_5\sigma}_{\tilde{a}_2})\rho^2$ and $B = \underbrace{b_0 + b_1\sigma}_{\tilde{b}_0} + (\underbrace{b_2 + b_3\sigma}_{\tilde{b}_1})\rho + (\underbrace{b_4\rho^2 + b_5\sigma}_{\tilde{b}_2})\rho^2$. The product $C = AB$ is then given by (6 multiplications and 14 additions over $\mathbb{F}_{3^{2m}}$):

$$\begin{bmatrix} \tilde{c}_0 \\ \tilde{c}_1 \\ \tilde{c}_2 \end{bmatrix} = \begin{bmatrix} (\tilde{a}_1 + \tilde{a}_2)(\tilde{b}_1 + \tilde{b}_2) + \tilde{a}_0\tilde{b}_0 - \tilde{a}_1\tilde{b}_1 - \tilde{a}_2\tilde{b}_2 \\ (\tilde{a}_0 + \tilde{a}_1)(\tilde{b}_0 + \tilde{b}_1) + (\tilde{a}_1 + \tilde{a}_2)(\tilde{b}_1 + \tilde{b}_2) - \tilde{a}_0\tilde{b}_0 + \tilde{a}_1\tilde{b}_1 \\ (\tilde{a}_0 + \tilde{a}_2)(\tilde{b}_0 + \tilde{b}_2) + \tilde{a}_1\tilde{b}_1 - \tilde{a}_0\tilde{b}_0 \end{bmatrix}.$$

Thus, multiplication over $\mathbb{F}_{3^{6m}}$ requires 18 multiplications and 58 additions (or subtractions) over $\mathbb{F}_{3^m}$.