Algorithms and Arithmetic Operators for Computing the η_T Pairing in Characteristic Three

Jean-Luc Beuchat, Nicolas Brisebarre, Jérémie Detrey, Eiji Okamoto, Masaaki Shirase, and Tsuyoshi Takagi

***Abstract*— Since their introduction in constructive cryptographic applications, pairings over (hyper)elliptic curves are at the heart of an ever increasing number of protocols. Software implementations being rather slow, the study of hardware architectures became an active research area.**

In this paper, we discuss several algorithms to compute the η_T pairing in characteristic three and suggest further improvements. These algorithms involve addition, multiplication, cubing, inversion, and sometimes cube root extraction over $\mathbb{F}_{3^m}$. We propose a hardware accelerator based on a unified arithmetic operator able to perform the operations required by a given algorithm. We describe the implementation of a compact coprocessor for the field $\mathbb{F}_{3^{97}}$ given by $\mathbb{F}_3[x]/(x^{97}+x^{12}+2)$, which compares favorably with other solutions described in the open literature.

***Index Terms*— η_T pairing, finite field arithmetic, elliptic curve, hardware accelerator, FPGA.**

I. Introduction

In 2001, Boneh, Lynn & Shacham [1] proposed a remarkable short signature scheme whose principle is the following. They consider an additive group $G_1 = \langle P \rangle$ of prime order q and a map-to-point hash function $H : \{0,1\}^* \to G_1$. The secret key is an element x of $\{1, 2, \ldots, q-1\}$ and the public key is $xP \in G_1$ for a signer. Let $m \in \{0,1\}^*$ be a message, they compute the signature $xH(m)$. To do the verification, they use a map called bilinear pairing that we now define.

Let $G_1 = \langle P \rangle$ be an additive group and G_2 a multiplicative group with identity 1. We assume that the discrete logarithm problem is hard in both G_1 and G_2. A bilinear pairing on (G_1, G_2) is a map $e : G_1 \times G_1 \to G_2$ that satisfies the following conditions:

1) *Bilinearity.* For all Q, R, $S \in G_1$,

$$e(Q+R,S) = e(Q,S)e(R,S) \quad \text{and}$$
$$e(Q,R+S) = e(Q,R)e(Q,S).$$

2) *Non-degeneracy.* $e(P,P) \neq 1$.
3) *Computability.* e can be efficiently computed.

Modifications of the Weil and Tate pairings provide such maps.

The verification in the BLS scheme is done by checking if the values $e(P, xH(m))$ and $e(xP, H(m))$ coincide. Actually, if $x' \in \{1, 2, \ldots, q-1\}$ satisfies $e(xP, H(m)) = e(P, x'H(m))$, then we obtain $e(P,H(m))^x = e(P,H(m))^{x'}$ thanks to the bilinearity property of the pairing. From the non-degeneracy of the pairing we know that $e(P,H(m))^x = e(P,H(m))^{x'}$ implies $x = x'$. The total cost is one hashing operation, one modular exponentiation and two pairing computations and the signature is twice as short as the one in DSA for similar level of security.

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.

N. Brisebarre is with Laboratoire LIP/Arénaire (CNRS – ENS Lyon – INRIA – UCBL), ENS Lyon, 46, allée d'Italie, F-69364 Lyon Cedex 07, France.

J. Detrey is with the Cosec group, b-it (Bonn-Aachen International Center for Information Technology), Dahlmannstraße 2, D-53113 Bonn, Germany.

M. Shirase and T. Takagi are with the School of Systems Information Science, Future University-Hakodate, 116-2 Kamedanakano-cho, Hakodate, Hokkaido, 041-8655, Japan.

A. Pairings in Cryptology

Pairings were first introduced in cryptology by Menezes, Okamoto & Vanstone [2] and Frey & Rück [3] for code-breaking purposes. Mitsunari, Sakai & Kasahara [4] and Sakai, Oghishi & Kasahara [5] seem to be the first to have discovered their constructive properties. Since the foundational work of Joux [6], an already large and ever increasing number of pairing-based protocols has been found. Most of them are described in the survey by Dutta, Barua & Sarkar [7]. As noticed in that survey, such protocols rely critically on efficient algorithms and implementations of pairing primitives.

According to [8], [9], when dealing with general curves providing common levels of security, the Tate pairing seems to be more efficient for computation than the Weil pairing and we now describe it.

Let E be a supersingular[1] elliptic curve over $\mathbb{F}_{p^m}$, where p is a prime and m a positive integer, and let $E(\mathbb{F}_{p^m})$ denote the group of its points. Let $\ell > 0$ be an integer relatively prime to p. The *embedding degree* (or *security multiplier*) is the least positive integer k satisfying $p^{km} \equiv 1 \pmod{\ell}$. Let $E(\mathbb{F}_{p^m})[\ell]$ denote the ℓ-torsion subgroup of $E(\mathbb{F}_{p^m})$, *i.e.* the set of elements P of $E(\mathbb{F}_{p^m})$ that satisfy $[\ell]P = \mathcal{O}$, where $\mathcal{O}$ is the point at infinity of the elliptic curve. Let $P \in E(\mathbb{F}_{p^m})[\ell]$ and $Q \in E(\mathbb{F}_{p^{km}})[\ell]$, let $f_{\ell,P}$ be a rational function on the curve with divisor $\ell(P) - \ell(\mathcal{O})$ (see [10] for an account of divisors), there exists a divisor D_Q equivalent to $(Q) - (\mathcal{O})$, with a support disjoint from the support of $f_{\ell,P}$. Then the Tate pairing[2] of order ℓ is the map $e : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^{km}})[\ell] \to \mathbb{F}^*_{p^{km}}$ defined by $e(P,Q) = f_{\ell,P}(D_Q)^{(p^{km}-1)/\ell}$. The kind of powering that occurs in this definition is called the final exponentiation; it makes it possible to get values in a multiplicative subgroup of $\mathbb{F}^*_{p^{km}}$ (which is required by most of the cryptographic applications) instead of a multiplicative subgroup of a quotient of $\mathbb{F}^*_{p^{km}}$.

In [11], Barreto *et al.* proved that this pairing can be computed as $e(P,Q) = f_{\ell,P}(Q)^{\frac{p^{km}-1}{\ell}}$, where $f_{\ell,P}$ is evaluated on a point rather than on a divisor. Thanks to a distortion map $\psi : E(\mathbb{F}_{p^m})[\ell] \to E(\mathbb{F}_{p^{km}})[\ell]$ (the concept of a distortion map was introduced in [12]), one can define the modified Tate pairing $\hat{e}$ by $\hat{e}(P,Q) = e(P, \psi(Q))$ for all P, $Q \in E(\mathbb{F}_{p^m})[\ell]$.

Miller [13], [14] proposed in 1986 the first algorithm for computing Weil and Tate pairings. Different ways for computing the

[1] See Theorem V.3.1 of [10] for a definition.

[2] We give here the definition from [11], slightly different from the initial one given in [3].

Tate pairing can be found in [11], [15]–[17]. In [18], Barreto *et al.* introduced the η_T pairing which extended and improved the Duursma-Lee techniques [16]. It makes it possible to efficiently compute the Tate pairing. The η_T pairing is presented in Section II in which we recall the relation between it and the modified Tate pairing.

B. Implementation Challenges

The software implementations of these successive algorithmic improvements being rather slow, the need for fine hardware implementations is strong. This is a critical issue to make pairings popular and of common use in cryptography and in particular in view of a successful industrial transfer. The papers [19]–[27] address that problem.

In this paper, we deal with the characteristic three case and, given a positive integer m coprime to 6, we consider E, a supersingular elliptic curve over $\mathbb{F}_{3^m}$, defined by the equation $y^2 = x^3 - x + b$, with $b \in \{-1, 1\}$. Following the discussion at the beginning of Section 5 of [18], there is no loss of generality from considering this case since these curves offer the same level of security for pairing applications as any supersingular elliptic curve over $\mathbb{F}_{3^m}$. The considered curve has an embedding degree of 6, which is the maximum value possible for supersingular elliptic curves, and hence seems to be an attractive choice for pairing implementation.

C. Our Contribution

The algorithm given in [18] for computing the η_T pairing halves the number of iterations used in the approach by Duursma & Lee [16] but has the drawback of using inverse Frobenius maps. In [25] Beuchat *et al.* proposed a modified η_T pairing algorithm in characteristic three that does not require any inverse Frobenius map. Moreover, they designed a novel arithmetic operator implementing addition, cubing, and multiplication over $\mathbb{F}_{3^{97}}$ which performs in a fast and cheap way the step of final exponentiation [26]. Then they extended in [27] this approach to the computation of the reduced η_T pairing (*i.e.* the combination of the η_T pairing and the final exponentiation).

In this article, we present a synthesis and an improvement of the results of the papers [25]–[27]. The outline of the paper is the following. In Section II, we define the η_T pairing and its reduced form, we give different algorithms to compute them and we provide exact cost evaluations for these algorithms. Section III is dedicated to the presentation of a reduced η_T pairing coprocessor that is based on a unified arithmetic operator that implements the various required elementary operations over $\mathbb{F}_{3^m}$. We want to mention that all the material (i.e. algorithms and architectures) presented in this section can be easily adapted to work on any field $\mathbb{F}_p[x]/(f(x))$ for any prime p and any polynomial f irreducible over $\mathbb{F}_p$. We implemented our coprocessor on several Field-Programmable Gate Array (FPGA) families for the field $\mathbb{F}_{3^{97}}$ given by $\mathbb{F}_3[x]/(x^{97} + x^{12} + 2)$. We provide the reader with a comprehensive comparison against state-of-the-art η_T pairing accelerators in Section IV and conclude our article in Section V.

II. Computation of the η_T Pairing in Characteristic Three

A. Preliminary Definitions

We use here the definition of the η_T pairing as introduced by Barreto *et al.* in [18]. The interested reader shall find in that article all the details related to the mathematical construction of the pairing, which we will deliberately not mention here for clarity's sake.

Let E be the supersingular elliptic curve defined by the equation $E : y^2 = x^3 - x + b$, where $b \in \{-1, 1\}$. Considering a positive integer m coprime to 6, the number of rational points of E over the finite field $\mathbb{F}_{3^m}$ is given by $N = \#E(\mathbb{F}_{3^m}) = 3^m + 1 + \mu b 3^{\frac{m+1}{2}}$, with

$$\mu = \begin{cases} +1 & \text{if } m \equiv 1, 11 \pmod{12}, \text{ or} \\ -1 & \text{if } m \equiv 5, 7 \pmod{12}. \end{cases}$$

The embedding degree k of E is then 6.

Choosing $T = 3^m - N = -\mu b 3^{\frac{m+1}{2}} - 1$ and an integer ℓ dividing N, we define the η_T pairing of two points P and Q of the ℓ-torsion $E(\mathbb{F}_{3^m})[\ell]$ as

$$\eta_T(P, Q) = \begin{cases} f_{T,P}(\psi(Q)) & \text{if } T > 0 \text{ (\textit{i.e.} } \mu b = -1), \text{ or} \\ f_{-T,-P}(\psi(Q)) & \text{if } T < 0 \text{ (\textit{i.e.} } \mu b = \ \ 1), \end{cases}$$

where:

- ψ is a distortion map from $E(\mathbb{F}_{3^m})[\ell]$ to $E(\mathbb{F}_{3^{6m}})[\ell]$ defined as $\psi(x, y) = (\rho - x, y\sigma)$ for all $(x, y) \in E(\mathbb{F}_{3^m})[\ell]$, as given in [11], where ρ and σ are elements of $\mathbb{F}_{3^{6m}}$ satisfying the equations $\rho^3 - \rho - b = 0$ and $\sigma^2 + 1 = 0$.
  As already remarked in [20], this allows for representing $\mathbb{F}_{3^{6m}}$ as an extension of $\mathbb{F}_{3^m}$ using the basis $(1, \sigma, \rho, \sigma\rho, \rho^2, \sigma\rho^2)$: $\mathbb{F}_{3^{6m}} = \mathbb{F}_{3^m}[\sigma, \rho] \cong \mathbb{F}_{3^m}[X, Y]/(X^2 + 1, Y^3 - Y - b)$. Hence, all the computations over $\mathbb{F}_{3^{6m}}$ can be replaced by computations over $\mathbb{F}_{3^m}$, as explicitly shown in Appendices V and VI.
- $f_{n,P}$, for $n \in \mathbb{N}$ and $P \in E(\mathbb{F}_{3^m})[\ell]$, is a rational function defined over $E(\mathbb{F}_{3^{6m}})[\ell]$ with divisor $(f_{n,P}) = n(P) - ([n]P) - (n - 1)(\mathcal{O})$.

In order to ensure that the obtained pairing values belong to the group of the ℓth roots of unity of $\mathbb{F}^*_{3^{6m}}$, we actually have to compute the *reduced* η_T pairing, defined as $\eta_T(P, Q)^M$, where

$$M = \frac{3^{6m} - 1}{N} = \left(3^{3m} - 1\right)\left(3^m + 1\right)\left(3^m + 1 - \mu b 3^{\frac{m+1}{2}}\right).$$

In the following, we will refer to this additional step as *final exponentiation*.

One should also note that, in characteristic 3, we have the following relation between the reduced η_T and modified Tate pairings:

$$\left(\eta_T(P, Q)^M\right)^{3T^2} = \left(\hat{e}(P, Q)^M\right)^L,$$

with $L = -\mu b 3^{\frac{m+3}{2}}$. Using v as a shorthand for $\eta_T(P, Q)^M$, we can compute the modified Tate pairing according to the following formula:

$$\hat{e}(P, Q)^M = v^{-2}\left(v^{3^{\frac{m+1}{2}}} \sqrt[3^m]{v^{3^{\frac{m-1}{2}}}}\right)^{-\mu b}.$$

Noting $T' = -\mu b T = 3^{\frac{m+1}{2}} + \mu b$ and $P' = [-\mu b]P$, we now have to compute $\eta_T(P, Q)^M = f_{T',P'}(\psi(Q))^M$. Using the Duursma-Lee techniques [16] to simplify the computation of $f_{n,P}$ in Miller's algorithm, we obtain

$$f_{T',P'}(\psi(Q)) = \left(\prod_{i=0}^{\frac{m-1}{2}} g_{[3^i]P'}(\psi(Q))^{3^{\frac{m-1}{2}-i}}\right) l_{P'}(\psi(Q)),$$

where:

- g_V, for all $V = (x_V, y_V) \in E(\mathbb{F}_{3^m})[\ell]$, is the rational function introduced by Duursma and Lee in [16], defined over $E(\mathbb{F}_{3^{6m}})[\ell]$ and having divisor $(g_V) = 3(V) + ([-3]V) - 4(\mathcal{O})$. For all $(x, y) \in E(\mathbb{F}_{3^{6m}})[\ell]$, we have
$$g_V(x, y) = y_V^3 y - (x_V^3 - x + b)^2.$$
- l_V, for all $V = (x_V, y_V) \in E(\mathbb{F}_{3^m})[\ell]$, is the equation of the line corresponding to the addition of $\left[3^{\frac{m+1}{2}}\right] V$ with $[\mu b]V$, defined for all $(x, y) \in E(\mathbb{F}_{3^{6m}})[\ell]$:
$$l_V(x, y) = y - \lambda y_V (x - x_V) - \mu b y_V,$$
with
$$\lambda = (-1)^{\frac{m+1}{2}} = \begin{cases} +1 & \text{if } m \equiv 7, 11 \pmod{12}, \text{ or} \\ -1 & \text{if } m \equiv 1, 5 \pmod{12}. \end{cases}$$
We can also rewrite the equation of l_V as
$$l_V(x, y) = y + \lambda y_V (x_V - x - \nu b),$$
introducing
$$\nu = \mu\lambda = \begin{cases} +1 & \text{if } m \equiv 5, 11 \pmod{12}, \text{ or} \\ -1 & \text{if } m \equiv 1, 7 \pmod{12}. \end{cases}$$

The remaining of this section will present and discuss various algorithms that can be used to effectively compute the reduced η_T pairing. The next three subsections will focus on the computation of $\eta_T(P, Q)$ only, the details of the final exponentiation being given in Section II-E. Finally, cost evaluations and comparisons will be presented in Section II-F.

B. Direct Approaches

1) Direct Algorithm: From the expression of $f_{T',P'}$, noting $\tilde{Q} = \psi(Q)$, we can write

$$f_{T',P'}(\tilde{Q}) = \left(\dots \left(g_{P'}(\tilde{Q})^3 \cdot g_{[3]P'}(\tilde{Q}) \right)^3 \dots \right)^3 g_{\left[3^{\frac{m-1}{2}}\right]P'}(\tilde{Q}) \cdot l_{P'}(\tilde{Q}).$$

Noting $P' = (x_{P'}, y_{P'})$ and $Q = (x_Q, y_Q)$, we have $[3^i]P' = \left(x_{P'}^{3^{2i}} - ib, (-1)^i y_{P'}^{3^{2i}}\right)$ and $\tilde{Q} = \psi(Q) = (\rho - x_Q, y_Q\sigma)$. Injecting these in the expressions of $g_{[3^i]P'}$ and $l_{P'}$ and defining $m' = \frac{m-1}{2}$, we obtain

$$g_{[3^i]P'}(\tilde{Q}) = (-1)^i y_{P'}^{3^{2i+1}} y_Q\sigma - \left(x_{P'}^{3^{2i+1}} + x_Q + (1-i)b - \rho\right)^2, \text{ and}$$
$$l_{P'}(\tilde{Q}) = y_Q\sigma - (-1)^{m'} y_{P'}^{3^{2m'+1}} \left(x_{P'}^{3^{2m'+1}} + x_Q + (1-m')b - \rho\right).$$

An iterative implementation of the η_T pairing following this construction is given in Algorithm 1. The cost of each pseudo-code instruction is given as comments in terms of additions/subtractions (A), multiplications (M) and cubings (C) over the underlying field $\mathbb{F}_{3^m}$.

A few remarks concerning this algorithm:

- The multiplication by $-\mu b$ on line 1 is for free. Indeed, $-\mu b$ being a constant (1 or -1) for fixed m and b, one can just compute the value of $-\mu b$ when those parameters are chosen, and propagate sign corrections on y_P throughout the whole algorithm.

Algorithm 1 Direct algorithm for computing the η_T pairing.

Input: $P, Q \in E(\mathbb{F}_{3^m})[\ell]$.
Output: $\eta_T(P, Q) \in \mathbb{F}^*_{3^{6m}}$.

1. $y_P \leftarrow -\mu b y_P$;
2. $x_P \leftarrow x_P^3$; $y_P \leftarrow y_P^3$; (2C)
3. $t \leftarrow x_P + x_Q + b$; $u \leftarrow y_P y_Q$; (1M, 2A)
4. $R \leftarrow (-t^2 + u\sigma - t\rho - \rho^2)^3$; (1M, 2C, 3A)
5. $x_P \leftarrow x_P^9$; $y_P \leftarrow -y_P^9$; (4C)
6. $t \leftarrow x_P + x_Q$; $u \leftarrow y_P y_Q$; (1M, 1A)
7. $S \leftarrow -t^2 + u\sigma - t\rho - \rho^2$; (1M)
8. $R \leftarrow R \cdot S$; (6M, 21A)
9. **for** $i \leftarrow 2$ **to** $\frac{m-1}{2}$ **do**
10. $\quad R \leftarrow R^3$; (6C, 6A)
11. $\quad x_P \leftarrow x_P^9 - b$; $y_P \leftarrow -y_P^9$; (4C, 1A)
12. $\quad t \leftarrow x_P + x_Q$; $u \leftarrow y_P y_Q$; (1M, 1A)
13. $\quad S \leftarrow -t^2 + u\sigma - t\rho - \rho^2$; (1M)
14. $\quad R \leftarrow R \cdot S$; (12M, 59A)
15. **end for**
16. $S \leftarrow -y_P t + y_Q\sigma + y_P\rho$; (1M)
17. $R \leftarrow R \cdot S$; (12M, 51A)
18. **return** R;

- Similarly, multiplications by λ, ν and b do not have any impact on the cost of the algorithm. The value of these constants are known in advance, and actually only represent sign changes in the algorithm.
- Since the representation of $-t^2+u\sigma-t\rho-\rho^2$ as an element of the tower field $\mathbb{F}_{3^{6m}}$ is sparse, the cubing on line 4 involves only 1 multiplication, 2 cubings and 3 additions over $\mathbb{F}_{3^m}$, as detailed in Appendix V-B.
- Additionally, $(-t^2+u\sigma-t\rho-\rho^2)^3$ has the same sparsity, and therefore the product of R and S on line 8 can be computed by means of only 6 multiplications and 21 additions over $\mathbb{F}_{3^m}$, as per Appendix VI-C.
- Inside the loop, the cubing of R on line 10 is computed in 6 cubings and 6 additions over $\mathbb{F}_{3^m}$ (Appendix V-A).
- The multiplication of R by S on line 14 involves only 12 multiplications and 59 additions over $\mathbb{F}_{3^m}$, as S is sparse (Appendix VI-B).
- The final product on line 17 is in turn computed by means of 12 multiplications and 51 additions, also thanks to the sparsity of S, as detailed in Appendix VI-B.

2) Simplification using Cube Roots: Cubing the intermediate result $R \in \mathbb{F}^*_{3^{6m}}$ at each iteration of Algorithm 1 is quite expensive. But one can use the fact that, due to the bilinearity of the reduced η_T pairing,

$$\eta_T(P, Q)^M = \left(\eta_T\left(P, \left[3^{-\frac{m-1}{2}}\right] Q\right)^{3^{\frac{m-1}{2}}} \right)^M,$$

to compute instead

$$f_{T',P'}(\tilde{Q})^{3^{\frac{m-1}{2}}} = \left(\prod_{i=0}^{\frac{m-1}{2}} g_{[3^i]P'}(\tilde{Q})^{3^{m-1-i}} \right) l_{P'}(\tilde{Q})^{3^{\frac{m-1}{2}}},$$

with $\tilde{Q} = \psi\left(\left[3^{-\frac{m-1}{2}}\right] Q\right) = \left(\rho - x_Q^3 - (\nu+1)b, -\lambda y_Q^3 \sigma\right)$.

Expanding everything, we obtain the following expressions, again with $m' = \frac{m-1}{2}$:

$$g_{[3^i]P'}(\tilde{Q})^{3^{m-1-i}} = -\lambda y_{P'}^{3^i} y_Q^{3^{-i}} \sigma - \left(x_{P'}^{3^i} + x_Q^{3^{-i}} - \nu b - \rho\right)^2, \text{ and}$$

$$l_{P'}(\tilde{Q})^{3^{\frac{m-1}{2}}} = y_Q^{3^{-m'}} \sigma + \lambda y_{P'}^{3^{m'}} \left(x_{P'}^{3^{m'}} + x_Q^{3^{-m'}} - \nu b - \rho\right).$$

This naturally gives another iterative method to compute $\eta_T(P,Q)$, presented in Algorithm 2. Here, the cubings over $\mathbb{F}_{3^{6m}}$ are traded for cube roots (noted R) over $\mathbb{F}_{3^m}$, which can be efficiently computed by means of a specific operator (see III-E for further details).

Algorithm 2 Simplified algorithm for computing the η_T pairing, with cube roots.

Input: $P, Q \in E(\mathbb{F}_{3^m})[\ell]$.
Output: $\eta_T(P,Q) \in \mathbb{F}^*_{3^{6m}}$.

1. $x_P \leftarrow x_P - \nu b$; (1A)
2. $y_P \leftarrow -\mu b y_P$;
3. $t \leftarrow x_P + x_Q$; $u \leftarrow y_P y_Q$; (1M, 1A)
4. $R \leftarrow -t^2 - \lambda u\sigma - t\rho - \rho^2$; (1M)
5. $x_P \leftarrow x_P^3$; $y_P \leftarrow y_P^3$; (2C)
6. $x_Q \leftarrow \sqrt[3]{x_Q}$; $y_Q \leftarrow \sqrt[3]{y_Q}$; (2R)
7. $t \leftarrow x_P + x_Q$; $u \leftarrow y_P y_Q$; (1M, 1A)
8. $S \leftarrow -t^2 - \lambda u\sigma - t\rho - \rho^2$; (1M)
9. $R \leftarrow R \cdot S$; (6M, 21A)
10. **for** $i \leftarrow 2$ **to** $\frac{m-1}{2}$ **do**
11. $x_P \leftarrow x_P^3$; $y_P \leftarrow y_P^3$; (2C)
12. $x_Q \leftarrow \sqrt[3]{x_Q}$; $y_Q \leftarrow \sqrt[3]{y_Q}$; (2R)
13. $t \leftarrow x_P + x_Q$; $u \leftarrow y_P y_Q$; (1M, 1A)
14. $S \leftarrow -t^2 - \lambda u\sigma - t\rho - \rho^2$; (1M)
15. $R \leftarrow R \cdot S$; (12M, 59A)
16. **end for**
17. $S \leftarrow \lambda y_P t + y_Q \sigma - \lambda y_P \rho$; (1M)
18. $R \leftarrow R \cdot S$; (12M, 51A)
19. **return** R;

3) Tabulating the Cube Roots: Even if cube roots can be computed with only a slight hardware overhead, it is sometimes advisable to restrict the hardware complexity of the arithmetic unit in order to achieve higher clock frequencies. The previous algorithm can easily be adapted to cube-root-free coprocessors by simply noticing that, as x_Q and $y_Q \in \mathbb{F}_{3^m}$, $x_Q^{3^{-i}} = x_Q^{3^{m-i}}$ and $y_Q^{3^{-i}} = y_Q^{3^{m-i}}$.

Therefore, computing the $m-1$ successive cubings of x_Q and y_Q, it is possible to tabulate the pre-computed values of $x_Q^{3^{-i}}$ and $y_Q^{3^{-i}}$ which will be looked-up on lines 6 and 12 of Algorithm 2 instead of computing the actual cube roots.

The $m-1$ cube roots of Algorithm 2 are hence traded for $2m-2$ cubings, at the expense of extra registers required to store the tabulated values as $m-1$ elements of $\mathbb{F}_{3^m}$.

This idea, originally suggested by Barreto *et al.* [18], was for instance applied by Ronan *et al.* in [23] in the case $m \equiv 1 \pmod{12}$, although they curiously do not compute the actual η_T pairing, but the value

$$\eta_T\left(P, [3^{-m}]Q\right)^{3^{\frac{m-1}{2}}} = \eta_T(P,Q)^{3^{-\frac{m+1}{2}}}.$$

C. Reversed-Loop Approaches

In [18], Barreto *et al.* suggest reversing the loop to compute the η_T pairing. To that purpose, they introduce a new index $j = 3^{\frac{m-1}{2}} - i$ for the loop. Taking $\tilde{Q} = \psi(Q)$, we find

$$f_{T',P'}(\tilde{Q}) = l_{P'}(\tilde{Q}) \left(\prod_{j=0}^{\frac{m-1}{2}} g_{\left[3^{\frac{m-1}{2}-j}\right]P'}(\tilde{Q})^{3^j} \right).$$

1) Reversed-Loop Algorithm: Directly injecting the expression of $\left[3^{\frac{m-1}{2}-j}\right] P' = \left(x_{P'}^{3^{-2j-1}} - (\nu+1-j)b, -\lambda(-1)^j y_{P'}^{3^{-2j-1}}\right)$ into the formulas, we obtain

$$l_{P'}(\tilde{Q}) = y_Q \sigma + \lambda y_{P'} \left(x_{P'} + x_Q - \nu b - \rho\right), \text{ and}$$

$$g_{\left[3^{\frac{m-1}{2}-j}\right]P'}(\tilde{Q})^{3^j} = -\lambda y_{P'}^{3^{-j}} y_Q^{3^j} \sigma - \left(x_{P'}^{3^{-j}} + x_Q^{3^j} - \nu b - \rho\right)^2.$$

Following this expression, a third iterative scheme for computing the η_T pairing can be directly devised, as detailed in Algorithm 3. In the case $m \equiv 1 \pmod{12}$, this is the exact same algorithm as described by Barreto *et al.* in [18].

Algorithm 3 Reversed-loop algorithm for computing the η_T pairing, with cube roots.

Input: $P, Q \in E(\mathbb{F}_{3^m})[\ell]$.
Output: $\eta_T(P,Q) \in \mathbb{F}^*_{3^{6m}}$.

1. $x_P \leftarrow x_P - \nu b$; (1A)
2. $y_P \leftarrow -\mu b y_P$;
3. $t \leftarrow x_P + x_Q$; (1A)
4. $R \leftarrow (\lambda y_P t + y_Q \sigma - \lambda y_P \rho) \cdot (-t^2 - \lambda y_P y_Q \sigma - t\rho - \rho^2)$; (6M, 1C, 6A)
5. **for** $j \leftarrow 1$ **to** $\frac{m-1}{2}$ **do**
6. $x_P \leftarrow \sqrt[3]{x_P}$; $y_P \leftarrow \sqrt[3]{y_P}$; (2R)
7. $x_Q \leftarrow x_Q^3$; $y_Q \leftarrow y_Q^3$; (2C)
8. $t \leftarrow x_P + x_Q$; $u \leftarrow y_P y_Q$; (1M, 1A)
9. $S \leftarrow -t^2 - \lambda u\sigma - t\rho - \rho^2$; (1M)
10. $R \leftarrow R \cdot S$; (12M, 59A)
11. **end for**
12. **return** R;

It is to be noted that given the expression of its operands, the multiplication on line 4 is computed by means of only 6 multiplications, 1 cubing and 6 additions over $\mathbb{F}_{3^m}$, as described in Appendix VI-D.

As for Algorithm 2, Algorithm 3 also requires the computation of cube roots. A similar technique of pre-computation and tabulation of the cube roots thanks to successive cubings of x_P and y_P can be also be used, although we will not detail it here.

2) Eliminating the Cube Roots: The apparent duality between Algorithms 2 and 3 can be exploited to find another cube-free algorithm, still based on the reversed loop but similar to Algorithm 1.

For that purpose, we once again compute the reduced η_T pairing of P and Q as

$$\eta_T(P,Q)^M = \left(\eta_T\left(P, \left[3^{-\frac{m-1}{2}}\right] Q\right)^{3^{\frac{m-1}{2}}} \right)^M.$$

Noting $\tilde{Q} = \psi\left(\left[3^{-\frac{m-1}{2}}\right] Q\right)$, the reversed loop becomes

$$\begin{aligned}
&f_{T',P'}(\tilde{Q})^{3^{\frac{m-1}{2}}} \\
&= l_{P'}(\tilde{Q})^{3^{\frac{m-1}{2}}} \left(\prod_{j=0}^{\frac{m-1}{2}} g_{\left[3^{\frac{m-1}{2}-j}\right]P'}(\tilde{Q})^{3^{\frac{m-1}{2}+j}} \right) \\
&= l_{P'}(\tilde{Q})^{3^{\frac{m-1}{2}}} \left(\prod_{j=0}^{\frac{m-1}{2}} h_{j,P'}(\tilde{Q})^{3^{\frac{m-1}{2}-j}} \right) \\
&= \left(\ldots \left(\left(l_{P'}(\tilde{Q}) \cdot h_{0,P'}(\tilde{Q}) \right)^3 h_{1,P'}(\tilde{Q}) \right)^3 \ldots \right)^3 h_{\frac{m-1}{2},P'}(\tilde{Q}),
\end{aligned}$$

with the rational function $h_{j,P'}(\tilde{Q})$ defined as

$$h_{j,P'}(\tilde{Q}) = g_{\left[3^{\frac{m-1}{2}-j}\right]P'}(\tilde{Q})^{3^{2j}}.$$

We then compute the explicit expressions of $l_P(\tilde{Q})$ and $h_{j,P'}(\tilde{Q})$:

$$\begin{aligned}
l_{P'}(\tilde{Q}) &= -\lambda y_Q^3 \sigma + \lambda y_{P'} \left(x_{P'} + x_Q^3 + b - \rho \right), \text{ and} \\
h_{j,P'}(\tilde{Q}) &= \\
&(-1)^j y_{P'} y_Q^{3^{2j+1}} \sigma - \left(x_{P'} + x_Q^{3^{2j+1}} + (1-j)b - \rho \right)^2.
\end{aligned}$$

Algorithm 4 is a direct implementation of the previous computation of $\eta_T(P,Q)$. Similarly to Algorithm 1, it uses cubings over $\mathbb{F}_{3^{6m}}$ in order to avoid the cube roots of Algorithm 3. In the case $m \equiv 1 \pmod{12}$, this algorithm corresponds to the η_T pairing computation described by Beuchat *et al.* in [25].

Algorithm 4 Cube-root-free reversed-loop algorithm for computing the η_T pairing.

Input: $P, Q \in E(\mathbb{F}_{3^m})[\ell]$.
Output: $\eta_T(P,Q) \in \mathbb{F}_{3^{6m}}^*$.

1. $x_P \leftarrow x_P + b$; (1A)
2. $y_P \leftarrow -\mu b y_P$;
3. $x_Q \leftarrow x_Q^3$; $y_Q \leftarrow y_Q^3$; (2C)
4. $t \leftarrow x_P + x_Q$; (1A)
5. $R \leftarrow (\lambda y_P t - \lambda y_Q \sigma - \lambda y_P \rho) \cdot (-t^2 + y_P y_Q \sigma - t\rho - \rho^2)$; (6M, 1C, 6A)
6. **for** $j \leftarrow 1$ **to** $\frac{m-1}{2}$ **do**
7. $\quad R \leftarrow R^3$; (6C, 6A)
8. $\quad x_Q \leftarrow x_Q^9 - b$; $y_Q \leftarrow -y_Q^9$; (4C, 1A)
9. $\quad t \leftarrow x_P + x_Q$; $u \leftarrow y_P y_Q$; (1M, 1A)
10. $\quad S \leftarrow -t^2 + u\sigma - t\rho - \rho^2$; (1M)
11. $\quad R \leftarrow R \cdot S$; (12M, 59A)
12. **end for**
13. **return** R;

D. Loop Unrolling

Granger *et al.* [28] proposed a loop unrolling technique for the Duursma-Lee algorithm. They exploit the sparsity of g_V in order to reduce the number of multiplications over $\mathbb{F}_{3^m}$, exactly in the same way as we reduced the first two iterations of Algorithms 1 and 2.

By noting that $h_{j,P'}(\tilde{Q})^3$ is also as sparse as $h_{j,P'}(\tilde{Q})$ (see Appendix V-B for details), we can apply the same approach to Algorithm 4.

In two successive iterations $2j'-1$ and $2j'$ of the loop, for $1 \le j' \le \lfloor \frac{m-1}{4} \rfloor$, we compute the new value of R as

$$\begin{aligned}
R &\leftarrow \left(R^3 \cdot h_{2j'-1,P'}(\tilde{Q}) \right)^3 \cdot h_{2j',P'}(\tilde{Q}) \\
&= R^9 \cdot h_{2j'-1,P'}(\tilde{Q})^3 \cdot h_{2j',P'}(\tilde{Q}).
\end{aligned}$$

The values of $h_{2j'-1,P'}(\tilde{Q})$ and $h_{2j',P'}(\tilde{Q})$, computed at iterations $2j'-1$ and $2j'$ respectively, are both of the form $-t^2 + u\sigma - t\rho - \rho^2$. Therefore, given t and u, the computation of $h_{2j'-1,P'}(\tilde{Q})^3$ requires only 1 multiplication, 2 cubings and 3 additions over $\mathbb{F}_{3^m}$, as per Appendix V-B. Similarly, the product of $h_{2j'-1,P'}(\tilde{Q})^3$ and $h_{2j',P'}(\tilde{Q})$ can be computed by means of only 6 multiplications and 21 additions, as explained in Appendix VI-C. Finally, multiplying this product by R^9 requires a full $\mathbb{F}_{3^{6m}}$ multiplication, which can be performed with 15 multiplications and 67 additions over $\mathbb{F}_{3^m}$ (see Appendix VI-A).

Hence, the cost of such a double iteration would be of 25 multiplications (neglecting the other operations), whereas two iterations of the original loop from Algorithm 4 cost $2 \times 14 = 28$ multiplications.

Following this, we can unroll the main loop of Algorithm 4 in order to save multiplications by computing two iterations at a time. The resulting scheme is shown in Algorithm 5, for the case where $\frac{m-1}{2}$ is even. If $\frac{m-1}{2}$ is actually odd, one just has to restrict the loop on j' from 1 to $\frac{m-3}{4}$, and compute the last product by an extra iteration of the original loop, for the additional cost of 14 multiplications, 10 cubings and 68 additions over $\mathbb{F}_{3^m}$.

Algorithm 5 Unrolled loop for the computation of the η_T pairing when $\frac{m-1}{2}$ is even.

Input: $P, Q \in E(\mathbb{F}_{3^m})[\ell]$.
Output: $\eta_T(P,Q) \in \mathbb{F}_{3^{6m}}^*$.

1. $x_P \leftarrow x_P + b$; (1A)
2. $y_P \leftarrow -\mu b y_P$;
3. $x_Q \leftarrow x_Q^3$; $y_Q \leftarrow y_Q^3$; (2C)
4. $t \leftarrow x_P + x_Q$; (1A)
5. $R \leftarrow (\lambda y_P t - \lambda y_Q \sigma - \lambda y_P \rho) \cdot (-t^2 + y_P y_Q \sigma - t\rho - \rho^2)$; (6M, 1C, 6A)
6. **for** $j' \leftarrow 1$ **to** $\frac{m-1}{4}$ **do**
7. $\quad R \leftarrow R^9$; (12C, 12A)
8. $\quad x_Q \leftarrow x_Q^9 - b$; $y_Q \leftarrow y_Q^9$; (4C, 1A)
9. $\quad t \leftarrow x_P + x_Q$; $u \leftarrow y_P y_Q$; (1M, 1A)
10. $\quad S \leftarrow (-t^2 - u\sigma - t\rho - \rho^2)^3$; (1M, 2C, 3A)
11. $\quad x_Q \leftarrow x_Q^9 - b$; $y_Q \leftarrow y_Q^9$; (4C, 1A)
12. $\quad t \leftarrow x_P + x_Q$; $u \leftarrow y_P y_Q$; (1M, 1A)
13. $\quad S' \leftarrow -t^2 + u\sigma - t\rho - \rho^2$; (1M)
14. $\quad S \leftarrow S \cdot S'$; (6M, 21A)
15. $\quad R \leftarrow R \cdot S$; (15M, 67A)
16. **end for**
17. **return** R;

It is to be noted that one could also straightforwardly apply a similar loop unrolling technique to Algorithm 1. However, we will not detail this point any further, for it is rigourously identical to the previous case.

E. Final Exponentiation

As already stated in Section II-A, the η_T pairing has to be reduced in order to be uniquely defined, and not only up to

ℓth powers. This reduction is achieved by means of a final exponentiation, in which $\eta_T(P,Q)$ is raised to the Mth power, with

$$M = \left(3^{3m}-1\right)\left(3^m+1\right)\left(3^m+1-\mu b 3^{\frac{m+1}{2}}\right).$$

For this particular exponentiation, we use the scheme presented by Shirase *et al.* in [29].

Taking $U = \eta_T(P,Q) \in \mathbb{F}^*_{3^{6m}}$, we first compute $U^{3^{3m}-1}$. Writing U as $U_0 + U_1\sigma$, where U_0 and $U_1 \in \mathbb{F}^*_{3^{3m}}$, and seeing that

$$U^{3^{3m}} = U_0 - U_1\sigma, \text{ and}$$
$$U^{-1} = \frac{U_0 - U_1\sigma}{U_0^2 + U_1^2},$$

we obtain the following expression for $U^{3^{3m}-1}$:

$$U^{3^{3m}-1} = \frac{(U_0^2 - U_1^2) + U_0U_1\sigma}{U_0^2 + U_1^2}.$$

This computation is directly implemented in Algorithm 6, where the multiplication (line 3), the squarings (lines 1 and 2), and the inversion (line 5) over $\mathbb{F}_{3^{3m}}$ are performed following the algorithms presented in Appendices II, III and IV respectively.

Algorithm 6 Computation of $U^{3^{3m}-1}$ in $\mathbb{F}^*_{3^{6m}}$.

Input: $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2 \in \mathbb{F}^*_{3^{6m}}$.
Output: $V = U^{3^{3m}-1} \in T_2(\mathbb{F}_{3^{3m}})$.

1. $m_0 \leftarrow (u_0 + u_2\rho + u_4\rho^2)^2$; (5M, 7A)
2. $m_1 \leftarrow (u_1 + u_3\rho + u_5\rho^2)^2$; (5M, 7A)
3. $m_2 \leftarrow (u_0 + u_2\rho + u_4\rho^2)\cdot(u_1 + u_3\rho + u_5\rho^2)$; (6M, 12A)
4. $a_0 \leftarrow m_0 - m_1$; $a_1 \leftarrow m_0 + m_1$; (6A)
5. $i \leftarrow a_1^{-1}$; (12M, 11A, 1I)
6. $V_0 \leftarrow a_0 \cdot i$; (6M, 12A)
7. $V_1 \leftarrow m_2 \cdot i$; (6M, 12A)
8. **return** $V_0 + V_1\sigma$;

One can then remark that

$$\frac{(U_0^2 - U_1^2)^2 + (U_0U_1)^2}{(U_0^2 + U_1^2)^2} = 1,$$

meaning that $U^{3^{3m}-1}$ is in fact an element of $T_2(\mathbb{F}_{3^{3m}})$, where $T_2(\mathbb{F}_{3^{3m}}) = \{X_0 + X_1\sigma \in \mathbb{F}^*_{3^{6m}} : X_0^2 + X_1^2 = 1\}$ is the torus as introduced by Granger *et al.* for the case of the Tate pairing in [28].

This is a crucial point here, since arithmetic on the torus $T_2(\mathbb{F}_{3^{3m}})$ is much simpler than arithmetic on $\mathbb{F}^*_{3^{6m}}$. Thus, given $U \in T_2(\mathbb{F}_{3^{3m}})$, Algorithm 7 computes U^{3^m+1} in only 9 multiplications and 18 or 19 (depending on the value of m modulo 6) additions over $\mathbb{F}_{3^m}$.

Finally, Algorithm 8 implements the complete final exponentiation. Given $U \in \mathbb{F}^*_{3^{6m}}$ as input, it first computes $U^{3^{3m}-1}$ thanks to Algorithm 6, then calls Algorithm 7 to obtain $U^{(3^{3m}-1)(3^m+1)}$. Then $W = U^{(3^{3m}-1)(3^m+1)3^{(m+1)/2}}$ is computed by successive cubings over $\mathbb{F}_{3^{6m}}$, while $V = U^{(3^{3m}-1)(3^m+1)(3^m+1)}$ is obtained by a second call to Algorithm 7. The value to be computed is then

$$U^M = \begin{cases} V \cdot W^{-1} & \text{when } \mu b = \ \ 1, \text{ or} \\ V \cdot W & \text{when } \mu b = -1, \end{cases}$$

hence the computation of $W' = W^{-\mu b}$ on line 8. When $\mu b = -1$, this is just a dummy operation, but it is an actual inversion when $\mu b = 1$. However, as $W \in T_2(\mathbb{F}_{3^{3m}})$, writing $W = W_0 + W_1\sigma$, we have

$$W^{-1} = \frac{W_0 - W_1\sigma}{W_0^2 + W_1^2} = W_0 - W_1\sigma.$$

Inversion over $T_2(\mathbb{F}_{3^{3m}})$ is therefore completely free, as it suffices to propagate the sign corrections in the final product $V \cdot W'$, implemented as a full multiplication over $\mathbb{F}^*_{3^{6m}}$.

Algorithm 7 Computation of U^{3^m+1} in the torus $T_2(\mathbb{F}_{3^{3m}})$.

Input: $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2 \in T_2(\mathbb{F}_{3^{3m}})$.
Output: $V = U^{3^m+1} \in T_2(\mathbb{F}_{3^{3m}})$.

1. $a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow u_2 + u_3$; $a_2 \leftarrow u_4 - u_5$; (3A)
2. $m_0 \leftarrow u_0 \cdot u_4$; $m_1 \leftarrow u_1 \cdot u_5$; $m_2 \leftarrow u_2 \cdot u_4$; (3M)
3. $m_3 \leftarrow u_3 \cdot u_5$; $m_4 \leftarrow a_0 \cdot a_2$; $m_5 \leftarrow u_1 \cdot u_2$; (3M)
4. $m_6 \leftarrow u_0 \cdot u_3$; $m_7 \leftarrow a_0 \cdot a_1$; $m_8 \leftarrow a_1 \cdot a_2$; (3M)
5. $a_3 \leftarrow m_5 + m_6 - m_7$; $a_4 \leftarrow -m_2 - m_3$; (3A)
6. $a_5 \leftarrow -m_2 + m_3$; $a_6 \leftarrow -m_0 + m_1 + m_4$; (3A)
7. **if** $m \equiv 1 \pmod 6$ **then**
8. $v_0 \leftarrow 1 + m_0 + m_1 + ba_4$; (3A)
9. $v_1 \leftarrow bm_5 - bm_6 + a_6$; (2A)
10. $v_2 \leftarrow -a_3 + a_4$; (1A)
11. $v_3 \leftarrow m_8 + a_5 - ba_6$; (2A)
12. $v_4 \leftarrow -ba_3 - ba_4$; (1A)
13. $v_5 \leftarrow bm_8 + ba_5$; (1A)
14. **else if** $m \equiv 5 \pmod 6$ **then**
15. $v_0 \leftarrow 1 + m_0 + m_1 - ba_4$; (3A)
16. $v_1 \leftarrow -bm_5 + bm_6 + a_6$; (2A)
17. $v_2 \leftarrow a_3$;
18. $v_3 \leftarrow m_8 + a_5 + ba_6$; (2A)
19. $v_4 \leftarrow -ba_3 - ba_4$; (1A)
20. $v_5 \leftarrow -bm_8 - ba_5$; (1A)
21. **end if**
22. **return** $v_0 + v_1\sigma + v_2\rho + v_3\sigma\rho + v_4\rho^2 + v_5\sigma\rho^2$;

Algorithm 8 Final exponentiation of the reduced η_T pairing [29].

Input: $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2 \in \mathbb{F}^*_{3^{6m}}$.
Output: $U^M \in T_2(\mathbb{F}_{3^{3m}}) \subset \mathbb{F}^*_{3^{6m}}$, with the exponent $M = (3^{3m}-1)(3^m+1)(3^m+1-\mu b 3^{\frac{m+1}{2}})$.

1. $V \leftarrow U^{3^{3m}-1}$; (40M, 67A, 1I)
2. $V \leftarrow V^{3^m+1}$; (9M, 18 or 19A)
3. $W \leftarrow V$;
4. **for** $i \leftarrow 1$ **to** $\frac{m+1}{2}$ **do**
5. $W \leftarrow W^3$; (6C, 6A)
6. **end for**
7. $V \leftarrow V^{3^m+1}$; (9M, 18 or 19A)
8. $W' \leftarrow W^{-\mu b}$;
9. **return** $V \cdot W'$; (15M, 67A)

F. Overall Cost Evaluations and Comparisons

The costs of all the previously detailed algorithms are summarized in Table I, in terms of additions (or subtractions), multiplications, cubings, cube roots and inversions over $\mathbb{F}_{3^m}$.

From this table, we can see that the additional cost for cube-root-free algorithms is approximately $4m$ extra cubings and $7m/2$ extra additions, when compared to the equivalent algorithms with cube roots. The choice of a type of algorithm instead of the other

TABLE I
COST OF THE PRESENTED ALGORITHMS FOR COMPUTING THE η_T PAIRING AND THE FINAL EXPONENTIATION, IN TERMS OF OPERATIONS OVER THE UNDERLYING FIELD $\mathbb{F}_{3^m}$.

		Additions	Multiplications	Cubings	Cube roots	Inversions
Direct loop	No cube root (Algorithm 1)	$67\frac{m-1}{2}+11$	$7m+2$	$5m-7$	0	0
	With cube roots (Algorithm 2)	$30m-15$	$7m+2$	$m-1$	$m-1$	0
Reversed loop	With cube roots (Algorithm 3)	$30m-22$	$7m-1$	m	$m-1$	0
	No cube root (Algorithm 4)	$67\frac{m-1}{2}+8$	$7m-1$	$5m-2$	0	0
Unrolled loop (Algorithm 5)	$\frac{m-1}{2}$ is even	$107\frac{m-1}{4}+8$	$25\frac{m-1}{4}+6$	$11\frac{m-1}{2}+3$	0	0
	$\frac{m-1}{2}$ is odd	$107\frac{m-3}{4}+76$	$25\frac{m-3}{4}+20$	$11\frac{m-1}{2}+2$	0	0
Final exp. (Algorithm 8)	$m \equiv 1 \pmod 6$	$3m+175$	73	$3m+3$	0	1
	$m \equiv 5 \pmod 6$	$3m+173$	73	$3m+3$	0	1

will therefore depend on the practicality of the computation of cube roots in the given finite field $\mathbb{F}_{3^m}$ (see the discussion in Section III-E).

This table also shows a slight superiority of reversed-loop algorithms versus direct-loop approaches. This is the reason why we chose to apply the loop unrolling technique to Algorithm 4.

The advantage of such a loop unrolling becomes also clearer when looking at Table I. From Algorithm 4 to Algorithm 5, we trade approximately $27m/4$ additions and $3m/4$ multiplications for $m/2$ cubings over $\mathbb{F}_{3^m}$.

The costs of these algorithms for $m = 97$, on which we focus more closely in this paper, is given in Table II. As detailed in Section III-B, we can compute the inversion over $\mathbb{F}_{3^{97}}$ according to Fermat's little theorem in 9 multiplications and 96 cubings, which allows us to express these costs in terms of additions, multiplications, cubings and cube roots only. The total number of operations for the complete computation of the reduced η_T pairing, using Algorithm 5 for the η_T pairing and Algorithm 8 for the final exponentiation, is also given.

TABLE II
COST EVALUATIONS OF THE REDUCED η_T PAIRING FOR $m = 97$. INVERSION OVER $\mathbb{F}_{3^{97}}$ IS CARRIED OUT ACCORDING TO FERMAT'S LITTLE THEOREM IN 9 MULTIPLICATIONS AND 96 CUBINGS.

		A	M	C	R
Direct loop	(Algorithm 1)	3227	681	478	0
	(Algorithm 2)	2895	681	96	96
Reversed loop	(Algorithm 3)	2888	678	97	96
	(Algorithm 4)	3224	678	483	0
Unrolled loop	(Algorithm 5)	2576	606	531	0
Final exp.	(Algorithm 8)	466	82	390	0
Total	(Algorithms 5 and 8)	3042	688	921	0

III. A COPROCESSOR FOR ARITHMETIC OVER $\mathbb{F}_{3^m}$

The η_T pairing calculation in characteristic 3 requires addition, multiplication, cubing, inversion, and sometimes cube root extraction over $\mathbb{F}_{3^m}$. We propose here a unified arithmetic operator which implements the required operations, and describe a hardware accelerator for pairing-based cryptography.

In the following, elements of the field extension $\mathbb{F}_{3^m}$ will be represented using a polynomial basis. Given a degree-m irreducible polynomial $f(x) \in \mathbb{F}_3[x]$, we have $\mathbb{F}_{3^m} \cong \mathbb{F}_3[x]/(f(x))$. Each element of $\mathbb{F}_{3^m}$ will then be represented as a polynomial $p(x)$ of degree $(m-1)$ and coefficients in $\mathbb{F}_3$:

$$p(x) = p_{m-1}x^{m-1} + \ldots + p_1 x + p_0.$$

Several researchers reported implementations of the Tate and η_T pairings on a supersingular curve defined on the field $\mathbb{F}_{3^{97}}$. Therefore, we discuss the implementation of Algorithm 5 for the field $\mathbb{F}_3[x]/(x^{97}+x^{12}+2)$ and the curve $y^2 = x^3 - x + 1$ (*i.e.* $b = 1$) on our coprocessor.

It is nonetheless important to note that the architectures and algorithms presented here can be easily adapted to different parameters. For instance a different irreducible polynomial $f(x)$, a different field extension degree m, or even a different characteristic p (cubing and cube root extraction, being respectively Frobenius and inverse Frobenius maps in characteristic 3, then replaced by raising to the pth power and pth root extraction).

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

Three families of algorithms allow one to compute $d0(x) \cdot d1(x) \bmod f(x)$ (see for instance [30]–[32] for an account of modular multiplication). In parallel-serial schemes, a single coefficient of the multiplier $d0(x)$ is processed at each step. This leads to small operators performing a multiplication in m clock cycles. 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 [33], offer a good trade-off between computation time and circuit area and are at the heart of several pairing coprocessors (see for instance [19], [20], [22], [23], [25], [34]).

Depending on the order in which coefficients of $d0(x)$ are processed, array multipliers can be implemented according to two schemes: most-significant element (MSE) first and least-significant element (LSE) first. Algorithm 9 summarizes the MSE-first scheme proposed by Shu, Kwon & Gaj [22]. Figure 1a illustrates the architecture of this operator for $D = 3$. It mainly consists of three Partial Product Generators (PPGs), three modulo $f(x)$ reduction units, a multioperand adder, and registers to store operands and intermediate results. Five bits allow for the control of the multiplier. If the irreducible polynomial over $\mathbb{F}_{3^m}$ is a

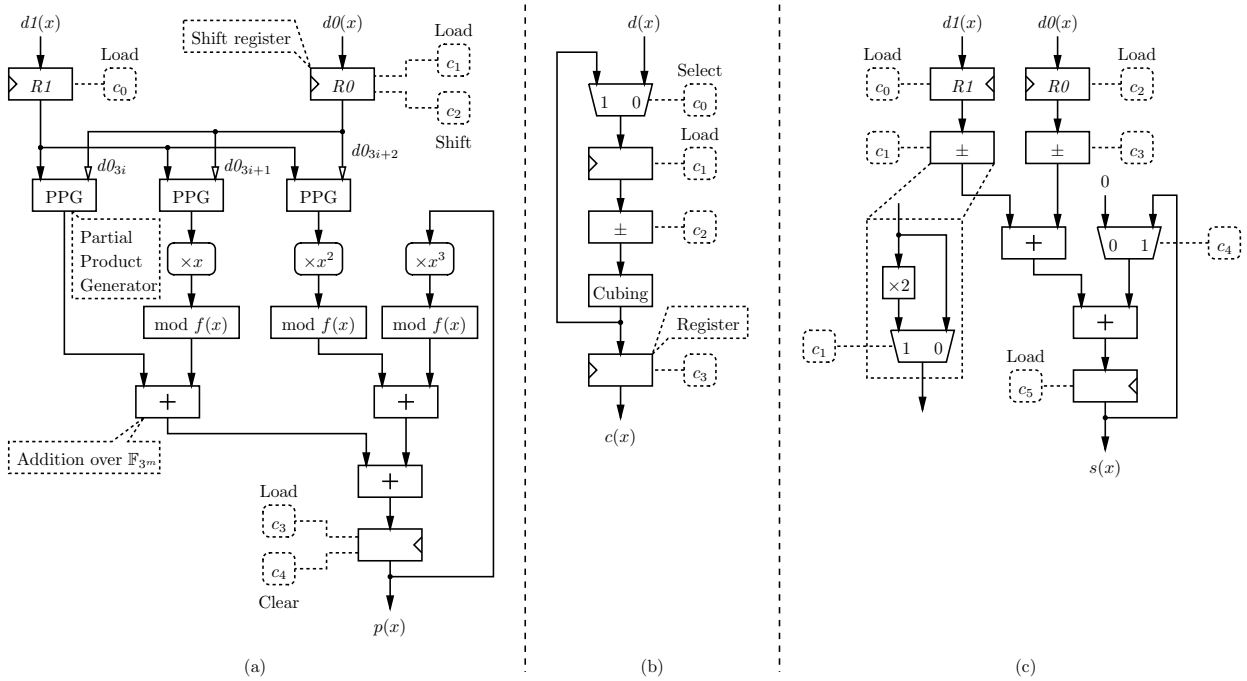

Fig. 1. Arithmetic operators over $\mathbb{F}_{3^m}$. (a) Multiplication ($D = 3$ coefficients of $d0(x)$ are processed at each clock cycle) [22]. (b) Cubing. (c) Addition/subtraction of two operands and accumulation. Boxes with rounded corners involve only wiring. The c_i's denote control bits.

trinomial or a pentanomial, modulo $f(x)$ operations are easy to implement. Consider for instance $f(x) = x^{97} + x^{12} + 2$ and let $u(x) = x \cdot d1(x)$ be a degree-97 polynomial. It suffices to remove $u_{97} \cdot f(x) = u_{97}x^{97} + u_{97}x^{12} + 2u_{97}$ from $u(x)$ to get $u(x) \bmod f(x)$. This involves only two multiplications and two subtractions over $\mathbb{F}_3$, namely $u_{12} - 1 \cdot u_{97}$ and $u_0 - 2 \cdot u_{97}$.

Elements of $\mathbb{F}_3$ are often represented as 2-bit unsigned integers. Let $d0_i = 2d0_i^H + d0_i^L$ and $d1_j = 2d1_j^H + d1_j^L$. Multiplication over $\mathbb{F}_3 = \{0, 1, 2\}$ is then defined as follows:

$$d0_i \cdot d1_j = 2\left(d0_i^H d1_j^L \vee d0_i^L d1_j^H\right) + \left(d0_i^L d1_j^L \vee d0_i^H d1_j^H\right),$$

and can be implemented by means of two 4-input Look-Up Tables (LUTs). Since $d0_i$ multiplies all coefficients of $d1$, the fanout of our array multiplier is equal to $2m$.

However, a careful encoding of the elements of $\mathbb{F}_3$ can reduce the fanout of the operator [35]. Since $2 \equiv -1 \pmod 3$, we take advantage of the borrow-save system [36] in order to represent the elements of $\mathbb{F}_3 = \{0, 1, -1\}$: $d0_i$ is encoded by a positive bit $d0_i^+$ and a negative bit $d0_i^-$ such that $d0_i = d0_i^+ - d0_i^-$. Multiplication over $\mathbb{F}_3$ is now defined by:

$$\begin{aligned} d0_i \cdot d1_j = &\left((1 - d1_j^-)d1_j^+ d0_i^+ \vee d1_j^-(1 - d1_j^+)(1 - d0_i^+)\right) - \\ &\left((1 - d1_j^-)d1_j^+ d0_i^- \vee d1_j^-(1 - d1_j^+)(1 - d0_i^-)\right), \end{aligned}$$

and requires two 3-input LUTs: the first one depends on $d0_i^+$, and the second one on $d0_i^-$. Thus, the fanout of the array multiplier is now equal to m. Since it is performed component-wise, addition over $\mathbb{F}_{3^m}$ is also a rather straightforward operation. If elements of $\mathbb{F}_3$ are represented by two bits, addition modulo 3 is for instance carried out by means of two 4-input LUTs.

B. Inversion over $\mathbb{F}_{3^m}$

The final exponentiation of the η_T pairing involves a single inversion over $\mathbb{F}_{3^m}$. Instead of designing a specific operator based on the Extended Euclidean Algorithm (EEA), we suggest to keep the circuit area as small as possible by performing this inversion according to Fermat's little theorem and Itoh and Tsujii's work [37] (Algorithm 10). Since this scheme requires only multiplications and cubings over $\mathbb{F}_{3^m}$, we do not have to include dedicated hardware for inversion in our coprocessor.

Algorithm 9 Multiplication over $\mathbb{F}_{3^m}$ [22].

Input: A degree-m monic polynomial $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1 x + f_0$ and two degree-$(m-1)$ polynomials $d0(x)$ and $d1(x)$. A parameter D which defines the number of coefficients of $d0(x)$ processed at each clock cycle. The algorithm requires a degree-$(m-1)$ polynomial $a(x)$ for intermediate computations.

Output: $p(x) = d0(x)d1(x) \bmod f(x)$

1. $p(x) \leftarrow 0$;
2. **for** $i \leftarrow \lceil m/D \rceil - 1$ **downto** 0 **do**
3. $\quad a(x) \leftarrow \sum_{j=0}^{D-1} \left(d0_{Di+j} \cdot d1(x) \cdot x^j\right) \bmod f(x)$;
4. $\quad p(x) \leftarrow a(x) + (p(x) \cdot x^D \bmod f(x))$;
5. **end for**
6. **return** $p(x)$;

Starting with an element d of $\mathbb{F}_{3^m}$, $d \neq 0$, we first raise it to the power of the base-3 repunit $(3^{m-1} - 1)/2$ to obtain r. This particular powering can be achieved using only $m - 2$ cubings over $\mathbb{F}_{3^m}$ and a few multiplications over $\mathbb{F}_{3^m}$ as detailed below. By cubing r and then multiplying the result by d, we successively obtain

$$u = d^{(3^m-3)/2}, \text{ and}$$
$$v = d^{(3^m-1)/2}.$$

A final product gives us the result

$$u \cdot v = d^{(3^m-3)/2} \cdot d^{(3^m-1)/2} = d^{3^m-2} = d^{-1}.$$

Since $v \neq 0$ and $v^2 = d^{3^m-1} = 1$, $v \in \mathbb{F}_3$ and this operation could be performed in a single clock cycle at the price of a modification of our MSE-first multiplier: adding an extra control bit and a multiplexer allows one to select the value of the coefficient $d0_{3i}$ between its normal value (the D most significant coefficients of the multiplier) and the D least significant coefficients of the multiplier. Indeed, as $v \in \mathbb{F}_3$, its coefficients v_i are zero for all $i \neq 0$. Therefore, we only need v_0 to compute the final multiplication $u \cdot v = u \cdot v_0$. As our multiplier operates in a most-significant-coefficient-first fashion, instead of performing the full multiplication over $\mathbb{F}_{3^m}$, this multiplexer would allow us to bypass the whole shift register mechanism and compute the product $u \cdot v$ in a single iteration of the multiplier. Since we consider $m = 97$ for our implementation, this trick would allow us to save only $\lceil m/D \rceil - 1 = \lceil 97/3 \rceil - 1 = 32$ clock cycles at the price of a longer critical path and a larger control word. Thus, we do not include this modification in our coprocessor.

Algorithm 10 Inversion over $\mathbb{F}_{3^m}$.

Input: A positive integer m, and $d \in \mathbb{F}_{3^m}$, $d \neq 0$.
Output: $d^{-1} \in \mathbb{F}_{3^m}$.
1. $r \leftarrow d^{(3^{m-1}-1)/2}$; (See Algorithm 11)
2. $u \leftarrow r^3$; (1C)
3. $v \leftarrow u \cdot d$; (1M)
4. **return** $u \cdot v$; (1M)

As already shown in [38] and [39], addition chains can prove to be perfectly suited to raise elements of $\mathbb{F}_{3^m}$ to particular powers, such as the radix-3 repunit $(3^{m-1}-1)/2$ required by our inversion algorithm. In the following, we will restrict ourselves to Brauer-type addition chains[3], whose definition follows.

A Brauer-type addition chain C of length l is a sequence of l integers $S = (j_1, \ldots, j_l)$ such that $0 \leq j_i < i$ for all $1 \leq i \leq l$. We can then construct another sequence $(n_0, \ldots, n_l)$ satisfying

$$\begin{cases} n_0 = 1, \text{ and} \\ n_i = n_{i-1} + n_{j_i}, \text{ for all } 1 \leq i \leq l. \end{cases}$$

C is said to *compute* n_l, the last element of the sequence. From [40], we also have the following additional property, for all $1 \leq l' \leq l$:

$$\sum_{i=1}^{l'} n_{j_i} = n_{l'} - 1.$$

Moreover, we can see that we have, for $n \leq n'$

$$d^{(3^{n+n'}-1)/2} = d^{(3^n-1)/2} \cdot \left(d^{(3^{n'}-1)/2} \right)^{3^n}.$$

Consequently, given a Brauer-type addition chain C of length l for $m-1$, we can compute the required $d^{(3^{m-1}-1)/2}$ as shown in Algorithm 11. This algorithm simply ensures that, for each iteration i, we have $z_i = d^{(3^{n_i}-1)/2}$, where $(n_0, \ldots, n_l)$ is the integer sequence associated with the addition chain C, verifying $n_l = m-1$. It requires l multiplications and $n_{j_1} + \cdots + n_{j_l} = m-2$ cubings over $\mathbb{F}_{3^m}$.

[3]Brauer-type addition chains are proved to be optimal for all numbers up to and including 12508 [40], which is more than enough for our needs.

Algorithm 11 Computation of $d^{(3^{m-1}-1)/2}$ over $\mathbb{F}_{3^m}$.

Output: A positive integer m, $d \in \mathbb{F}_{3^m}$, $d \neq 0$, a Brauer-type addition chain $S = (j_1, \ldots, j_l)$ for $m-1$, and the integer sequence $(n_0, \ldots, n_l)$ associated with C.
Input: $d^{(3^{m-1}-1)/2} \in \mathbb{F}_{3^m}$.
1. $z_0 \leftarrow d$;
2. **for** $i \leftarrow 1$ **to** l **do**
3. $\quad z_i \leftarrow z_{j_i} \cdot z_{i-1}^{3^{n_{j_i}}}$; (1M, n_{j_i}C)
4. **end for**
5. **return** z_l;

Therefore, our inversion scheme requires a total of $l+2$ multiplications and $m-1$ cubings over $\mathbb{F}_{3^m}$. For $m = 97$, an addition chain of length $l = 7$ allows us to compute $d^{(3^{96}-1)/2}$, and the overall cost of inversion is equal to 9 multiplications and 96 cubings over $\mathbb{F}_{3^{97}}$.

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

Cubing over $\mathbb{F}_{3^m}$ consists in reducing the following expression modulo $f(x)$:

$$c(x) = d(x)^3 \bmod f(x) = \sum_{i=0}^{m-1} d_i x^{3i} \bmod f(x).$$

This general expression can be seen as a sum of D' elements of $\mathbb{F}_{3^m}$. The coefficients of those polynomials can be directly matched to the coefficients of the operand, possibly multiplied by 2. Thus, cubing requires a multioperand adder and some extra wiring for the permutation of the coefficients. Multiplication by 2 consists in swapping the positive and negative bits of an element of $\mathbb{F}_3$. For instance, if $f(x) = x^{97} + x^{12} + 2$, we have to compute a sum of $D' = 3$ operands:

$$\begin{aligned} \nu_0(x) &= d_{32}x^{96} + 2d_{60}x^{95} + d_{88}x^{94} + \ldots + \\ &\quad d_1x^3 + d_{33}x^2 + 2d_{61}x + d_0, \\ \nu_1(x) &= d_{64}x^{95} + d_{92}x^{94} + \ldots + d_{90}x^3 + d_{65}x + d_{89}, \\ \nu_2(x) &= d_{96}x^{94} + \ldots + d_{94}x^3 + d_{93}, \end{aligned}$$

where $\nu_i(x) \in \mathbb{F}_{3^{97}}$, $0 \leq i \leq 2$, and

$$c(x) = d(x)^3 = \nu_0(x) + \nu_1(x) + \nu_2(x).$$

Recall that our inversion algorithm involves successive cubings. Since storing intermediate results in memory would be too time consuming, our cubing unit should include a feedback mechanism to efficiently implement Algorithm 11. Furthermore, cubing over $\mathbb{F}_{3^{6m}}$ requires the computation of $-u_5^3$, where $u_5 \in \mathbb{F}_{3^m}$ (see Appendix V-A for details). These considerations suggest the design of the operator depicted by Figure 1b.

If we have a closer look at the scheduling of the reduced η_T pairing algorithm, we note that there is no parallelism between multiplications and cubings over $\mathbb{F}_{3^m}$. If the array multiplier processes $D \geq D'$ coefficients at each clock cycle, we could take advantage of its multioperand adder to perform cubing. Figure 2 describes how to modify the multiplier when $D = D' = 3$:

- The feedback loop responsible for the accumulation of partial products must be deactivated while cubing. An array of m AND gates performs this task and allows one to carry out the initialization step of the modular multiplication (instruction $p(x) \leftarrow 0$ in Algorithm 9).

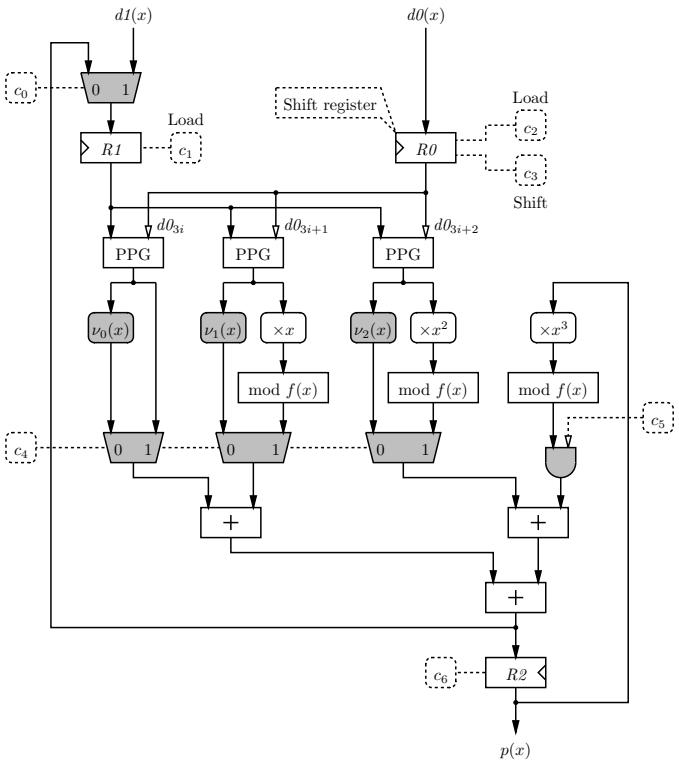

Fig. 2. Operator for multiplication and cubing over $\mathbb{F}_3[x]/(x^{97}+x^{12}+2)$. Boxes with rounded corners involve only wiring. The c_i's denote control bits. Grayed boxes outline the modifications of the array multiplier of Figure 1a.

- Multiplexers select the input of the multioperand adders between modulo $f(x)$ reduced partial products and the $\nu_i(x)$'s.
- The shift register of the multiplier and the PPGs allow for the control of cubing operations. If we store a control word in register *R0* such that $d0_{3i} = d0_{3i+1} = d0_{3i+2} = -1$, the operator returns $-d1(x)^3$. If $d0_{3i} = d0_{3i+1} = d0_{3i+2} = 1$, we obtain $d1(x)^3$.

D. Addition over $\mathbb{F}_{3^m}$

The reduced η_T pairing algorithms discussed in this paper involve additions, subtractions, and accumulations over $\mathbb{F}_{3^m}$. Figure 1c describes an operator implementing these functionalities. Again, a closer look at the reduced η_T pairing algorithms as well as at the algorithms for arithmetic over $\mathbb{F}_{3^{3m}}$ and $\mathbb{F}_{3^{6m}}$ indicates that there is almost no parallelism between additions and multiplications over $\mathbb{F}_{3^m}$. We suggest to further modify our array multiplier to include addition, subtraction, and accumulation (Figure 3):

- An additional register is needed to store the second operand of an addition. Again, the shift register stores a control word to control additions. Assume for instance that we have to compute $-d2(x) + d1(x)$. We respectively load $d2(x)$ and $d1(x)$ in registers *R2* and *R1* and define a control word stored in *R0* so that $d0_{3i} = 1$, $d0_{3i+1} = 2$, and $d0_{3i+2} = 0$. We will thus compute $(d1(x)+2\cdot d2(x)+0\cdot d1(x)) \bmod f(x) = (d1(x) - d2(x)) \bmod f(x)$. Since the reduced η_T pairing algorithm involves successive additions and cubings, each control word loaded in the shift register manages a sequence of operations. Note that
  - while performing a multiplication or a cubing, registers *R1* and *R2* must store the same value;
  - $d0_{3i+2}$ is always equal to zero in the case of addition.
- A multiplexer in the accumulation loop allows one to select between the content of register *R3* (accumulation) or the content of *R3* shifted and reduced modulo $f(x)$ (multiplication).
- An additional multiplexer is required to select the second input of the multioperand adder: $d2(x)$ (addition), $(d2(x) \cdot d0_{3i+1} \cdot x) \bmod f(x)$ (multiplication), or $\nu_1(x)$ (cubing).

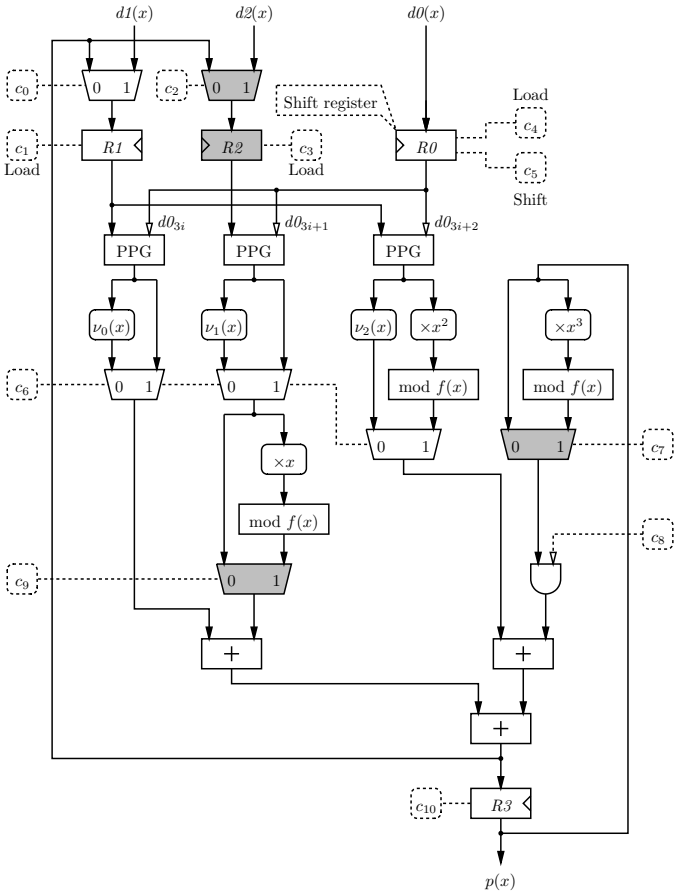

Fig. 3. Operator for addition, multiplication, and cubing over $\mathbb{F}_3[x]/(x^{97}+x^{12}+2)$. Boxes with rounded corners involve only wiring. The c_i's denote control bits. Grayed boxes outline the modifications of the operator of Figure 2.

E. Cube Root over $\mathbb{F}_{3^m}$

Some of the η_T pairing algorithms in characteristic 3 described in Section II involve cube roots over $\mathbb{F}_{3^m}$. This function is computed exactly in the same way as cubing: first, the normal form of $\sqrt[3]{d(x)} \bmod f(x)$ is obtained by solving the m-dimensional linear system given by the equation $\left(\sqrt[3]{d(x)}\right)^3 \bmod f(x) = d(x)$. The result is then expressed as a sum of polynomials, each one being a permutation of the coefficients of the operand $d(x)$ multiplied by a constant. The number of polynomials we have to add depends on $f(x)$. Barreto gives a list of irreducible polynomials leading to efficient cube root operators in [41].

F. Architecture of the Coprocessor

Figure 4 describes the architecture of our η_T pairing coprocessor. It consists of a single processing element (unified operator for addition, multiplication, and cubing), registers implemented by means of a dual-port RAM (6 Virtex-II Pro SelectRAM+ blocks or 13 Cyclone II M4K memory blocks), and a control unit which consists of a Finite State Machine (FSM) and an instruction memory (ROM). Each instruction consists of four fields: an 11-bit word which specifies the functionality of the processing element, address and write enable signal for port B of the dual-port RAM, address for port A of the dual-port RAM, and a 6-bit control word which manages jump instructions and indicates how many times an instruction must be repeated. This approach makes it possible for instance to execute the consecutive steps appearing in the multiplication over $\mathbb{F}_{3^m}$ with a single instruction.

The architecture described by Figure 4 was captured in the VHDL language and prototyped on several Altera and Xilinx FPGAs. We selected the following parameters: $m = 97$, $b = 1$, and $f(x) = x^{97} + x^{12} + 2$. Both synthesis and place-and-route steps were performed with Quartus II 7.1 Web Edition and ISE WebPACK 9.2i. The implementation on this coprocessor of the reduced η_T pairing (using Algorithm 5 for the η_T pairing and Algorithm 8 for the final exponentiation) takes 900 instructions which are executed in 27800 clock cycles. Table III summarizes the area (in slices on Xilinx FPGAs and Logic Elements (LEs) on the Altera device) and the calculation time.

It is worth noticing that an operator for inversion over $\mathbb{F}_{3^{97}}$ based on the EEA occupies 3422 LEs on a Cyclone-II device [42], and 2210 slices on a Virtex-II FPGA [43]. The implementation of the algorithm based on Itoh and Tsujii's work requires 394 clock cycles on our coprocessor for $m = 97$. The EEA needs $2m = 194$ clock cycles to return the inverse. Therefore, introducing specific hardware for inversion would double the circuit area while reducing the calculation time by less than 1%.

We also described a naive coprocessor embedding the multiplier, the cubing unit, and the adder depicted in Figure 1. The outputs of the these operators are connected to the register file by means of a 3-input multiplexer controlled by 2 additional bits. Place-and-route results indicate that such a coprocessor (without control unit) occupies 2199 slices on a Spartan-3 FPGA, and 3345 LEs on a Cyclone-II device. Furthermore, we need 17 bits to control this ALU. Thus, our unified operator reduces both the area of the coprocessor and the width of the control words.

In order to guarantee the security of pairing-based cryptosystems in a near future, larger extension degrees will probably have to be considered, thus raising the question of designing such a unified operator for other extension fields. For this purpose, we wrote a C++ program which automatically generates a synthesizable VHDL description of a unified operator according to the characteristic and the irreducible polynomial $f(x)$.

IV. Comparisons

Grabher and Page designed a coprocessor dealing with arithmetic over $\mathbb{F}_{3^m}$, which is controlled by a general purpose processor [19]. The ALU embeds an adder, a subtracter, a multiplier (with $D = 4$), a cubing unit, and a cube root operator based on the method highlighted by Barreto [41]. This architecture occupies 4481 slices and allows one to perform the Duursma-Lee algorithm and its final exponentiation in $432.3\,\mu$s. The main advantage is that the control can be compiled using a re-targeted GCC toolchain and other algorithms should easily be implemented on this architecture. Our approach leads however to a much simpler control unit and allows us to divide the number of slices by 2.4.

Another implementation of the Duursma-Lee algorithm was proposed by Kerins *et al.* in [20]. It features a parallel multiplier over $\mathbb{F}_{3^{6m}}$ based on Karatsuba-Ofman's scheme. Since the final exponentiation requires a general multiplication over $\mathbb{F}_{3^{6m}}$, the authors can not take advantage of the optimizations described in this paper and in [21] for the pairing calculation. Therefore, the hardware architecture consists of 18 multipliers and 6 cubing circuits over $\mathbb{F}_{3^{97}}$, along with, quoting [20], "a suitable amount of simpler $\mathbb{F}_{3^m}$ arithmetic circuits for performing addition, subtraction, and negation". Since the authors claim that roughly 100% of available resources are required to implement their pairing accelerator, the cost can be estimated to 55616 slices [22]. The approach proposed in this paper reduces the area and the computation time by 30 and 4.4 respectively. Note that a multiplier over $\mathbb{F}_{3^{6m}}$ based on the fast Fourier transform [44] would save three multipliers over $\mathbb{F}_{3^m}$. Since all multiplications over $\mathbb{F}_{3^m}$ are performed in parallel, this approach would only slightly reduce the circuit area without decreasing the calculation time.

Beuchat *et al.* described a fast architecture for the computation of the η_T pairing [25]. The authors introduced a novel multiplication algorithm over $\mathbb{F}_{3^{6m}}$ which takes advantage of the constant coefficients of R_1. Thus, this design must be supplemented with a coprocessor for final exponentiation and the full pairing accelerator requires around 18000 LEs on a Cyclone II FPGA [26]. The computation of the pairing and the final exponentiation require 4849 and 4082 clock cycles respectively. Since both steps are pipelined, we can consider that a new result is returned after 4849 clock cycles if we perform a sufficient amount of consecutive full η_T pairings. In order to compare our accelerator against this architecture, we implemented it on an Altera Cyclone II 5 FPGA with Quartus II 7.1 Web Edition. Our design occupies 3216 LEs and the maximal clock frequency of 152 MHz allows one to compute a pairing in $183\,\mu$s. The architecture proposed in this paper is therefore 6 times slower, but 5.6 times smaller.

In order to study the trade-off between circuit area and calculation time of the η_T pairing, Ronan *et al.* wrote a C program which automatically generates a VHDL description of a coprocessor and its control unit according to the number of multipliers over $\mathbb{F}_{3^m}$ to be included and the parameter D [23]. An architecture embedding five multipliers processing $D = 4$ coefficients at each clock cycle computes for instance a full pairing in $187\,\mu$s. Though slightly faster, this design requires five times the amount of slices of our pairing accelerator. Our approach offers a better compromise between area and calculation time.

To our best knowledge, the fastest η_T pairing processor described in the open literature was designed by Jiang [24]. Unfortunately, Jiang does not give any detail about his architecture. Since a pairing is computed in 1627 clock cycles and that multiplication over $\mathbb{F}_{3^m}$ is based on an LSE array multiplier processing $D = 7$ coefficients at each clock cycle, we can however guess that the design includes a hardwired multiplier over $\mathbb{F}_{3^{6m}}$. Though 6.5 faster than the coprocessor based on our unified arithmetic operator, the design by Jiang requires 40 times more slices.

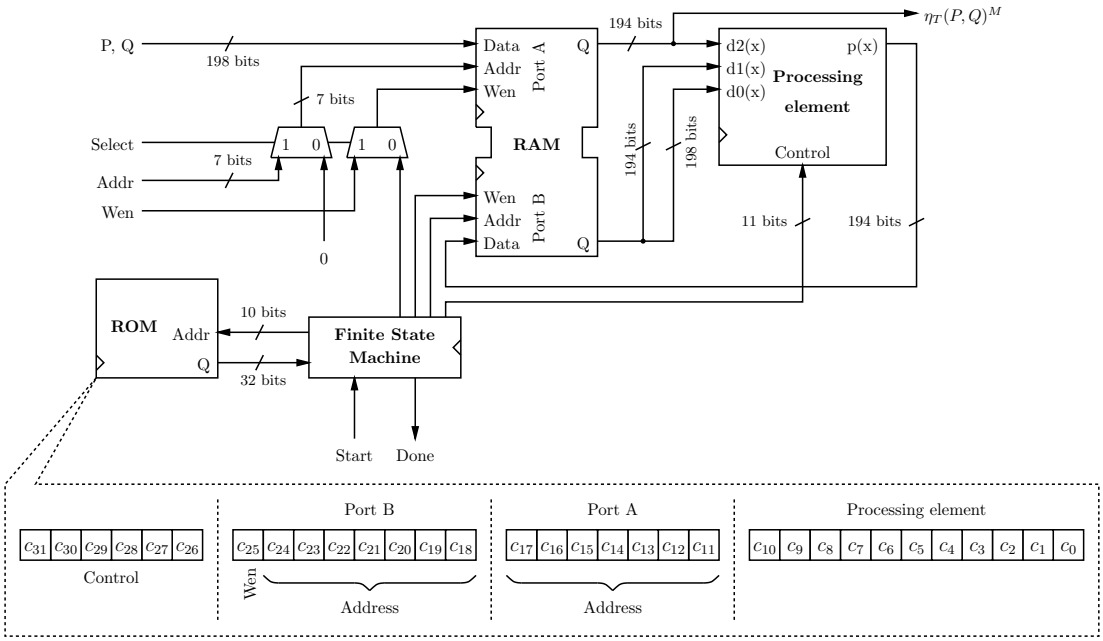

Fig. 4. Architecture of the coprocessor for arithmetic over $\mathbb{F}_{3^m}$.

TABLE III
AREA AND CALCULATION TIME OF A $\mathbb{F}_{3^{97}}$ REDUCED η_T PAIRING COPROCESSOR.

	Virtex-II Pro 4	**Virtex-4 LX 15**	**Spartan-3 200**	**Cyclone-II 5**
Area	1833 slices	1851 slices	1857 slices	3216 LEs
Clock cycles	27800 cycles			
Clock frequency	145 MHz	203 MHz	100 MHz	152 MHz
Calculation time	192 μs	137 μs	278 μs	183 μs

TABLE IV
FPGA-BASED ACCELERATORS OVER $\mathbb{F}_{3^{97}}$ IN THE LITERATURE. THE PARAMETER D REFERS TO THE NUMBER OF COEFFICIENTS PROCESSED AT EACH CLOCK CYCLE BY A MULTIPLIER.

	Grabher and Page [19]	**Kerins *et al.* [20]**	**Beuchat *et al.* [25], [26]**
Algorithm	Modified Tate pairing	Modified Tate pairing	Reduced η_T pairing
FPGA	Virtex-II Pro 4	Virtex-II Pro 125	Cyclone II 35
Multiplier(s)	1 ($D = 4$)	18 ($D = 4$)	9 ($D = 3$)
Area	4481 slices	55616 slices	$\sim$ 18000 LEs
Clock cycles	59946	12866	4849
Clock frequency	150 MHz	15 MHz	149 MHz
Calculation time	432.3 μs	Estimated to 850 μs	33 μs

	Ronan *et al.* [23]		**Jiang [24]**
Algorithm	Reduced η_T pairing	Reduced η_T pairing	Reduced η_T pairing
FPGA	Virtex-II Pro 100	Virtex-II Pro 100	Virtex-4 LX200
Multiplier(s)	5 ($D = 4$)	8 ($D = 4$)	($D = 7$)
Area	10540 slices	15401 slices	74105 slices
Clock cycles	15853	15529	1627
Clock frequency	84.8 MHz	84.8 MHz	77.7 MHz
Calculation time	187 μs	183 μs	20.9 μs

V. Conclusion

We discussed several algorithms to compute the η_T pairing and its final exponentiation in characteristic three. We proposed a compact implementation of the reduced η_T pairing in characteristic three over $\mathbb{F}_3[x]/(x^{97}+x^{12}+2)$. Our architecture is based on a unified arithmetic operator which leads to the smallest circuit proposed in the open literature while demonstrating competitive performances.

Future works should include studies of the η_T pairing in characteristic 2, where the wired multipliers embedded in most of the current FPGAs should allow for cheaper and faster array- and even fully parallel multipliers over $\mathbb{F}_{2^m}$. Such more efficient architectures would then allow us to investigate the η_T pairing over hyperelliptic curves.

The study of the Ate pairing [45] would also be of interest, for it presents a large speedup when compared to the Tate pairing and also supports non-supersingular curves.

Acknowledgment

The authors would like to thank Francisco Rodríguez-Henríquez and Guerric Meurice de Dormale for their valuable comments and Guillaume Hanrot for his fine explanations of some algorithmical and theoretical issues about pairings. This work was supported by the New Energy and Industrial Technology Development Organization (NEDO), Japan.

References

[1] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *Advances in Cryptology – ASIACRYPT 2001*, ser. Lecture Notes in Computer Science, C. Boyd, Ed., no. 2248. Springer, 2001, pp. 514–532.

[2] A. Menezes, T. Okamoto, and S. A. Vanstone, "Reducing elliptic curves logarithms to logarithms in a finite field," *IEEE Transactions on Information Theory*, vol. 39, no. 5, pp. 1639–1646, Sept. 1993.

[3] G. Frey and H.-G. Rück, "A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves," *Mathematics of Computation*, vol. 62, no. 206, pp. 865–874, Apr. 1994.

[4] S. Mitsunari, R. Sakai, and M. Kasahara, "A new traitor tracing," *IEICE Trans. Fundamentals*, vol. E85-A, no. 2, pp. 481–484, Feb 2002.

[5] R. Sakai, K. Ohgishi, and M. Kasahara, "Cryptosystems based on pairing," in *2000 Symposium on Cryptography and Information Security (SCIS2000), Okinawa, Japan*, Jan. 2000, pp. 26–28.

[6] A. Joux, "A one round protocol for tripartite Diffie-Hellman," in *Algorithmic Number Theory – ANTS IV*, ser. Lecture Notes in Computer Science, W. Bosma, Ed., no. 1838. Springer, 2000, pp. 385–394.

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

[8] R. Granger, D. Page, and N. P. Smart, "High security pairing-based cryptography revisited," in *Algorithmic Number Theory – ANTS VII*, ser. Lecture Notes in Computer Science, F. Hess, S. Pauli, and M. Pohst, Eds., no. 4076. Springer, 2006, pp. 480–494.

[9] N. Koblitz and A. Menezes, "Pairing-based cryptography at high security levels," in *Cryptography and Coding*, ser. Lecture Notes in Computer Science, N. P. Smart, Ed., no. 3796. Springer, 2005, pp. 13–36.

[10] J. H. Silverman, *The Arithmetic of Elliptic Curves*, ser. Graduate Texts in Mathematics. Springer-Verlag, 1986, no. 106.

[11] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott, "Efficient algorithms for pairing-based cryptosystems," in *Advances in Cryptology – CRYPTO 2002*, ser. Lecture Notes in Computer Science, M. Yung, Ed., no. 2442. Springer, 2002, pp. 354–368.

[12] E. R. Verheul, "Evidence that XTR is more secure than supersingular elliptic curve cryptosystems," *Journal of Cryptology*, vol. 17, no. 4, pp. 277–296, 2004.

[13] V. S. Miller, "Short programs for functions on curves," 1986, available at http://crypto.stanford.edu/miller.

[14] ——, "The Weil pairing, and its efficient calculation," *Journal of Cryptology*, vol. 17, no. 4, pp. 235–261, 2004.

[15] S. D. Galbraith, K. Harrison, and D. Soldera, "Implementing the Tate pairing," in *Algorithmic Number Theory – ANTS V*, ser. Lecture Notes in Computer Science, C. Fieker and D. Kohel, Eds., no. 2369. Springer, 2002, pp. 324–337.

[16] I. Duursma and H. S. Lee, "Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$," in *Advances in Cryptology – ASIACRYPT 2003*, ser. Lecture Notes in Computer Science, C. S. Laih, Ed., no. 2894. Springer, 2003, pp. 111–123.

[17] S. Kwon, "Efficient Tate pairing computation for elliptic curves over binary fields," in *Information Security and Privacy – ACISP 2005*, ser. Lecture Notes in Computer Science, C. Boyd and J. M. González Nieto, Eds., vol. 3574. Springer, 2005, pp. 134–145.

[18] P. S. L. M. Barreto, S. D. Galbraith, C. Ó hÉigeartaigh, and M. Scott, "Efficient pairing computation on supersingular Abelian varieties," in *Designs, Codes and Cryptography*. Springer Netherlands, Mar. 2007, vol. 42(3), pp. 239–271.

[19] P. Grabher and D. Page, "Hardware acceleration of the Tate pairing in characteristic three," in *Cryptographic Hardware and Embedded Systems – CHES 2005*, ser. Lecture Notes in Computer Science, J. R. Rao and B. Sunar, Eds., no. 3659. Springer, 2005, pp. 398–411.

[20] T. Kerins, W. P. Marnane, E. M. Popovici, and P. Barreto, "Efficient hardware for the Tate pairing calculation in characteristic three," in *Cryptographic Hardware and Embedded Systems – CHES 2005*, ser. Lecture Notes in Computer Science, J. R. Rao and B. Sunar, Eds., no. 3659. Springer, 2005, pp. 412–426.

[21] 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.

[22] 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)*. IEEE, 2006, pp. 173–180.

[23] R. Ronan, C. Murphy, T. Kerins, C. Ó hÉigeartaigh, and P. S. Barreto, "A flexible processor for the characteristic 3 η_T pairing," *Int. J. High Performance Systems Architecture*, vol. 1, no. 2, pp. 79–88, 2007.

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

[25] 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*, P. Kornerup and J.-M. Muller, Eds. IEEE Computer Society, 2007, pp. 97–104.

[26] 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 *Proceedings of Waifi 2007*, ser. Lecture Notes in Computer Science, C. Carlet and B. Sunar, Eds., no. 4547. Springer, 2007, pp. 25–39.

[27] J.-L. Beuchat, N. Brisebarre, J. Detrey, and E. Okamoto, "Arithmetic operators for pairing-based cryptography," in *Cryptographic Hardware and Embedded Systems – CHES 2007*, ser. Lecture Notes in Computer Science, P. Paillier and I. Verbauwhede, Eds., no. 4727. Springer, 2007, pp. 239–255.

[28] R. Granger, D. Page, and M. Stam, "On small characteristic algebraic tori in pairing-based cryptography," *LMS Journal of Computation and Mathematics*, vol. 9, pp. 64–85, Mar. 2006.

[29] M. Shirase, T. Takagi, and E. Okamoto, "Some efficient algorithms for the final exponentiation of η_T pairing," in *3rd International Information Security Practice and Experience Conference, (ISPEC'07)*, ser. Lecture Notes in Computer Science, E. Dawson and D. S. Wong, Eds., no. 4464. Hong Kong, China: Springer-Verlag, May 2007, pp. 254–268.

[30] J.-L. Beuchat, T. Miyoshi, J.-M. Muller, and E. Okamoto, "Horner's rule-based multiplication over GF(p) and GF(p^n): A survey," *International Journal of Electronics*, 2008, to appear.

[31] S. E. Erdem, T. Yamk, and Ç. K. Koç, "Polynomial basis multiplication over GF(2^m)," *Acta Applicandae Mathematicae*, vol. 93, no. 1–3, pp. 33–55, Sept. 2006.

[32] J. Guajardo, T. Güneysu, S. Kumar, C. Paar, and J. Pelzl, "Efficient hardware implementation of finite fields with applications to cryptography," *Acta Applicandae Mathematicae*, vol. 93, no. 1–3, pp. 75–118, Sept. 2006.

[33] L. Song and K. K. Parhi, "Low energy digit-serial/parallel finite field multipliers," *Journal of VLSI Signal Processing*, vol. 19, no. 2, pp. 149–166, July 1998.

[34] R. Ronan, C. Ó hÉigeartaigh, C. Murphy, M. Scott, T. Kerins, and W. Marnane, "An embedded processor for a pairing-based cryptosystem," in *Proceedings of the Third International Conference on Informa-*

tion Technology: New Generations (ITNG'06). IEEE Computer Society, 2006.
[35] G. Meurice de Dormale, personal communication.
[36] J.-C. Bajard, J. Duprat, S. Kla, and J.-M. Muller, "Some operators for on-line radix-2 computations," *Journal of Parallel and Distributed Computing*, vol. 22, pp. 336–345, 1994.
[37] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases," *Information and Computation*, vol. 78, pp. 171–177, 1988.
[38] J. von zur Gathen and M. Nöcker, "Computing special powers in finite fields," *Mathematics of Computation*, vol. 73, no. 247, pp. 1499–1523, 2003.
[39] F. Rodríguez-Henríquez, G. Morales-Luna, N. A. Saqib, and N. Cruz-Cortés, "A parallel version of the Itoh-Tsujii multiplicative inversion algorithm," in *Reconfigurable Computing: Architectures, Tools and Applications – Proceedings of ARC 2007*, ser. Lecture Notes in Computer Science, P. C. Diniz, E. Marques, K. Bertels, M. M. Fernandes, and J. M. P. Cardoso, Eds., no. 4419. Springer, 2007, pp. 226–237.
[40] D. E. Knuth, *The Art of Computer Programming*, 3rd ed. Addision-Wesley, 1998, vol. 2, *Seminumerical Algorithms*.
[41] P. S. L. M. Barreto, "A note on efficient computation of cube roots in characteristic 3," 2004, cryptology ePrint Archive, Report 2004/305.
[42] A. Vithanage, "Personal communication."
[43] T. Kerins, E. Popovici, and W. Marnane, "Algorithms and architectures for use in FPGA implementations of identity based encryption schemes," in *Field-Programmable Logic and Applications*, ser. Lecture Notes in Computer Science, J. Becker, M. Platzner, and S. Vernalde, Eds., no. 3203. Springer, 2004, pp. 74–83.
[44] E. Gorla, C. Puttmann, and J. Shokrollahi, "Explicit formulas for efficient multiplication in $\mathbb{F}_{3^{6m}}$," in *Proceedings of SAC 2007*, ser. Lecture Notes in Computer Science. Springer, 2007.
[45] F. Hess, N. Smart, and F. Vercauteren, "The Eta pairing revisited," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4595–4602, Oct. 2006.

APPENDICES

We describe here how to implement the arithmetic operations over $\mathbb{F}_{3^{2m}}$, $\mathbb{F}_{3^{3m}}$, and $\mathbb{F}_{3^{6m}}$ involved in the η_T pairing calculation. In order to compute the number of operations over $\mathbb{F}_{3^m}$, we assume that the ALU is able to compute $u \cdot v$, $\pm u \pm v$ and $\pm u^3$, where u and $v \in \mathbb{F}_{3^m}$.

APPENDIX I
MULTIPLICATION OVER $\mathbb{F}_{3^{2m}}$

Let $U = u_0 + u_1\sigma$ and $V = v_0 + v_1\sigma$, where u_0, u_1, v_0, and $v_1 \in \mathbb{F}_{3^m}$. The product UV is carried out according to Karatsuba-Ofman's algorithm:

$$U \cdot V = (u_0v_0 - u_1v_1) + ((u_0 + u_1)(v_0 + v_1) - u_0v_0 - u_1v_1)\sigma.$$

It requires 3 multiplications and 5 additions over $\mathbb{F}_{3^m}$.

APPENDIX II
MULTIPLICATION OVER $\mathbb{F}_{3^{3m}}$

Assume that $U = u_0 + u_1\rho + u_2\rho^2$ and $V = v_0 + v_1\rho + v_2\rho^2$, where u_i, $v_i \in \mathbb{F}_{3^m}$, $0 \le i \le 2$. The product $W = U \cdot V$ is then given by

$$\begin{aligned} w_0 &= b(bu_1 + u_2)(v_1 + bv_2) + u_0v_0 - u_1v_1 - u_2v_2, \\ w_1 &= (u_0 + u_1)(v_0 + v_1) + (bu_1 + u_2)(v_1 + bv_2) \\ &\quad -u_0v_0 - (b+1)u_1v_1, \text{ and} \\ w_2 &= (u_0 + u_2)(v_0 + v_2) - u_0v_0 + u_1v_1. \end{aligned}$$

Multiplication over $\mathbb{F}_{3^{3m}}$ involves 6 multiplications and 12 additions overs $\mathbb{F}_{3^m}$ (Algorithm 12).

Algorithm 12 Multiplication over $\mathbb{F}_{3^{3m}}$.

Input: $U = u_0 + u_1\rho + u_2\rho^2$ and $V = v_0 + v_1\rho + v_2\rho^2 \in \mathbb{F}_{3^{3m}}$.
Output: $W = U \cdot V \in \mathbb{F}_{3^{3m}}$.

1. $a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow u_0 + u_2$; $a_2 \leftarrow bu1 + u_2$; (3A)
2. $a_3 \leftarrow v_0 + v_1$; $a_4 \leftarrow v_0 + v_2$; $a_5 \leftarrow v_1 + bv_2$; (3A)
3. $m_0 \leftarrow u_0 \cdot v_0$; $m_1 \leftarrow u_1 \cdot v_1$; $m_2 \leftarrow u_2 \cdot v_2$; (3M)
4. $m_3 \leftarrow a_0 \cdot a_3$; $m_4 \leftarrow a_1 \cdot a_4$; $m_5 \leftarrow a_2 \cdot a_5$; (3M)
5. $a_6 \leftarrow m_0 - m_1$; (1A)
6. $w_0 \leftarrow a_6 - m_2 + bm_5$ (2A)
7. **if** $b = 1$ **then**
8. $\quad w_1 \leftarrow -a_6 + m_3 + m_5$; (2A)
9. **else**
10. $\quad w_1 \leftarrow -m_0 + m_3 + m_5$; (2A)
11. **end if**
12. $w_2 \leftarrow -a_6 + m_4$; (1A)
13. **return** $w_0 + w_1\rho + w_2\rho^2$;

APPENDIX III
SQUARING OVER $\mathbb{F}_{3^{3m}}$

Let $U = u_0 + u_1\rho + u_2\rho^2 \in \mathbb{F}_{3^{3m}}$, with $u_i \in \mathbb{F}_{3^m}$, $0 \le i \le 2$. $V = U^2$ is given by

$$\begin{aligned} v_0 &= u_0^2 - bu_1u_2, \\ v_1 &= bu_2^2 - u_0u_1 - u_1u_2, \text{ and} \\ v_2 &= (u_0 + u_1) \cdot (u_0 + u_1 + u_2) - u_0^2 + u_0u_1 + u_1u_2. \end{aligned}$$

Thus, squaring over $\mathbb{F}_{3^{3m}}$ requires 5 multiplications and 7 additions over $\mathbb{F}_{3^m}$ (Algorithm 13).

Algorithm 13 Squaring over $\mathbb{F}_{3^{3m}}$.

Input: $U = u_0 + u_1\rho + u_2\rho^2 \in \mathbb{F}_{3^{3m}}$.
Output: $V = U^2 \in \mathbb{F}_{3^{3m}}$.

1. $a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow a_0 + u_2$; (2A)
2. $m_0 \leftarrow u_0^2$; $m_1 \leftarrow u_0 \cdot u_1$; $m_2 \leftarrow u_1 \cdot u_2$; (3M)
3. $m_3 \leftarrow u_2^2$; $m_4 \leftarrow a_1^2$; (2M)
4. $a_2 \leftarrow m_1 + m_2$; (1A)
5. $v_0 \leftarrow m_0 - bm_2$; (1A)
6. $v_1 \leftarrow bm_3 - a_2$; (1A)
7. $v_2 \leftarrow m_4 + a_2 - m_0$; (2A)
8. **return** $v_0 + v_1\rho + v_2\rho^2$;

APPENDIX IV
INVERSION OVER $\mathbb{F}_{3^{3m}}$

Let $V = v_0 + v_1\rho + v_2\rho^2 \in \mathbb{F}_{3^{3m}}$ be the multiplicative inverse of $U = u_0 + u_1\rho + u_2\rho^2 \in \mathbb{F}_{3^{3m}}$, $U \neq 0$, where the u_i, $v_i \in \mathbb{F}_{3^m}$, $0 \le i \le 2$. Since $U \cdot V = 1$, we obtain

$$\begin{cases} u_0v_0 + bu_2v_1 + bu_1v_2 = 1, \\ u_1v_0 + (u_0 + u_2)v_1 + (u_1 + bu_2)v_2 = 0, \\ u_2v_0 + u_1v_1 + (u_0 + u_2)v_2 = 0. \end{cases}$$

The solution of this system of equations is then given by

$$\begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} = w^{-1} \begin{bmatrix} u_0^2 - (u_1^2 - u_2^2) - u_2(u_0 + bu_1) \\ bu_2^2 - u_0u_1 \\ u_1^2 - u_2^2 - u_0u_2 \end{bmatrix},$$

where $w = u_0^2(u_0 - u_2) + u_1^2(-u_0 + bu_1) + u_2^2(-(-u_0 + bu_1) + u_2) \in \mathbb{F}_{3^m}$. This operation involves 12 multiplications,

11 additions (or subtractions), and a single inversion over $\mathbb{F}_{3^m}$ (Algorithm 14).

Algorithm 14 Inversion over $\mathbb{F}_{3^{3m}}$.

Input: $U = u_0 + u_1\rho + u_2\rho^2 \in \mathbb{F}_{3^{3m}}$, $U \neq 0$.
Output: $V = U^{-1} \in \mathbb{F}_{3^{3m}}$.

1. $a_0 \leftarrow u_0 + bu_1$; $a_1 \leftarrow u_0 - u_2$; (2A)
2. $a_2 \leftarrow -u_0 + u_1$; $a_3 \leftarrow -a_2 + u_2$; (2A)
3. $m_0 \leftarrow u_0^2$; $m_1 \leftarrow u_1^2$; $m_2 \leftarrow u_2^2$; (3M)
4. $m_3 \leftarrow u_0 \cdot u_1$; $m_4 \leftarrow u_0 \cdot u_2$; $m_5 \leftarrow u_2 \cdot a_0$; (3M)
5. $m_6 \leftarrow m_0 \cdot a_1$; $m_7 \leftarrow m_1 \cdot a_2$; $m_8 \leftarrow m_2 \cdot a_3$; (3M)
6. $w \leftarrow m_6 + m_7 + m_8$; (2A)
7. $i \leftarrow w^{-1}$; (1I)
8. $a_4 \leftarrow m_1 - m_2$; $a_5 \leftarrow -a_4 + m_0 - m_5$; (3A)
9. $a_6 \leftarrow bm_2 - m_3$; $a_7 \leftarrow a_4 - m_4$; (2A)
10. $v_0 \leftarrow i \cdot a_5$; $v_1 \leftarrow i \cdot a_6$; $v_2 \leftarrow i \cdot a_7$; (3M)
11. **return** $v_0 + v_1\rho + v_2\rho^2$;

APPENDIX V
CUBING OVER $\mathbb{F}_{3^{6m}}$

A. General Algorithm

Let $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2 \in \mathbb{F}_{3^{6m}}$. $U^3 \in \mathbb{F}_{3^{6m}}$ is defined as follows:

$$U^3 = u_0^3 + u_1^3\sigma^3 + u_2^3\rho^3 + u_3^3(\sigma\rho)^3 + u_4^3\left(\rho^2\right)^3 + u_5^3\left(\sigma\rho^2\right)^3.$$

Since

$$\begin{cases} \rho^3 & = \rho + b, \\ \left(\rho^2\right)^3 & = \rho^2 - b\rho + 1, \end{cases} \quad \begin{cases} \sigma^3 & = -\sigma, \\ (\sigma\rho)^3 & = -\sigma\rho - b\sigma, \\ \left(\sigma\rho^2\right)^3 & = -\sigma\rho^2 + b\sigma\rho - \sigma, \end{cases}$$

we obtain the following coefficients for $V = U^3$:

$$\begin{aligned} v_0 &= u_0^3 + bu_2^3 + u_4^3, & v_1 &= -u_1^3 - bu_3^3 - u_5^3, \\ v_2 &= u_2^3 - bu_4^3, & v_3 &= -u_3^3 + bu_5^3, \\ v_4 &= u_4^3, & v_5 &= -u_5^3. \end{aligned}$$

As our unified operator computes $-u_5^3$ in one clock cycle, cubing over $\mathbb{F}_{3^{6m}}$ requires 6 cubings and 6 additions over $\mathbb{F}_{3^m}$.

B. Computation of $(-t^2 + u\sigma - t\rho - \rho^2)^3$

Let $U = -t^2 + u\sigma - t\rho - \rho^2$. According to the previous formula for cubing over $\mathbb{F}_{3^{6m}}$, we have

$$V = U^3 = v_0 + v_1\sigma + v_2\rho - \rho^2,$$

where

$$\begin{aligned} v_0 &= -t^6 - bt^3 - 1, \\ v_1 &= -u^3, \text{ and} \\ v_2 &= -t^3 + b. \end{aligned}$$

Therefore, U^3 is as sparse as U and this specific cubing involves a single multiplication, 2 cubings, and 3 additions over $\mathbb{F}_{3^m}$ (Algorithm 15).

This operation is usually followed by a multiplication which is optimized to take advantage of $v_4 = -1$ (see for instance Appendix VI-C). Thus, our coprocessor does not explicitely compute $v_4 \leftarrow -1$.

Algorithm 15 Computation of $(-t^2 + u\sigma - t\rho - \rho^2)^3$.

Input: t and $u \in \mathbb{F}_{3^m}$.
Output: $V = (-t^2 + u\sigma - t\rho - \rho^2)^3 \in \mathbb{F}_{3^{6m}}$.

1. $c_0 \leftarrow t^3$; $c_1 \leftarrow -u^3$; (2C)
2. $m_0 \leftarrow c_0^2$; (1M)
3. $v_0 \leftarrow -m_0 - bc_0 - 1$; (2A)
4. $v_1 \leftarrow c_1$;
5. $v_2 \leftarrow b - c_0$; (1A)
6. **return** $v_0 + v_1\sigma + v_2\rho - \rho^2$;

APPENDIX VI
MULTIPLICATION OVER $\mathbb{F}_{3^{6m}}$

A. General Algorithm

Elements of $\mathbb{F}_{3^{6m}}$ can be represented as degree-2 polynomials over $\mathbb{F}_{3^{2m}}$. Gorla *et al.* introduced an evaluation-interpolation scheme to perform multiplication over $\mathbb{F}_{3^{6m}}$ by means of five multiplications over $\mathbb{F}_{3^{2m}}$ [44]. Then, Karatsuba-Ofman's algorithm allows one to compute each multiplication over $\mathbb{F}_{3^{2m}}$ by means of three multiplications over $\mathbb{F}_{3^m}$ (see Appendix I). Thus, the scheme proposed by Gorla *et al.* to multiply two elements of $\mathbb{F}_{3^{6m}}$ involves 15 multiplications over $\mathbb{F}_{3^m}$ (Algorithm 16).

B. Multiplication by a Sparse Operand

The last multiplication over $\mathbb{F}_{3^{6m}}$ of the η_T pairing algorithms discussed in Section II-B is cheaper: it consists in computing the product $(u_0 + u_1\sigma + u_2\rho) \cdot (v_0 + v_1\sigma + v_2\rho + v_3\sigma\rho + v_4\rho^2 + v_5\sigma\rho^2)$ and requires 12 multiplications and 51 additions over $\mathbb{F}_{3^m}$ (Algorithm 17).

The first multiplication of the η_T pairing algorithms based on the reversed-loop approach (Section II-C) also benefits from this optimization. Since

$$(u_0 + u_1\sigma + u_2\rho - \rho^2) \cdot V = (u_0 + u_1\sigma + u_2\rho) \cdot V - \rho^2 \cdot V, \quad (1)$$

it suffices to subtract $\rho^2 V$ from the element of $\mathbb{F}_{3^{6m}}$ returned by Algorithm 17. Recall that $\rho^3 = \rho + b$ and note that Algorithm 17 requires two intermediate variables $r_1 = v_0 + v_4$ and $r_2 = v_1 + v_5$. We then have

$$\begin{aligned} -\rho^2 \cdot V &= -v_2 b - bv_3\sigma - (v_2 + bv_4)\rho \\ &\quad -(v_3 + bv_5)\sigma\rho - (v_0 + v_4)\rho^2 - (v_1 + v_5)\sigma\rho^2 \\ &= -v_2 b - bv_3\sigma - (v_2 + bv_4)\rho \\ &\quad -(v_3 + bv_5)\sigma\rho - r_1\rho^2 - r_2\sigma\rho^2. \end{aligned}$$

Therefore, subtracting $\rho^2 \cdot V$ involves 8 additions over $\mathbb{F}_{3^m}$ and the total cost of Equation (1) is 12 multiplications and 59 additions over $\mathbb{F}_{3^m}$.

C. Computation of $(u_0 + u_1\sigma + u_2\rho - \rho^2) \cdot (v_0 + v_1\sigma + v_2\rho - \rho^2)$

The multiplication of $U = u_0 + u_1\sigma + u_2\rho - \rho^2$ by $V = v_0 + v_1\sigma + v_2\rho - \rho^2$, where both U and V are in $\mathbb{F}_{3^{6m}}$, requires 6 multiplications and 21 additions over $\mathbb{F}_{3^m}$ (Algorithm 18).

D. Computation of $(\lambda y_P t - \lambda y_Q\sigma - \lambda y_P\rho) \cdot (-t^2 + y_P y_Q\sigma - t\rho - \rho^2)$

We consider here the first multiplication over $\mathbb{F}_{3^{6m}}$ of the η_T pairing calculation based on the reversed-loop approach, as in Algorithms 4 and 5. The same multiplication is also found in

Algorithms 3, only with $-\lambda y_Q$ instead of y_Q (which has no impact on the algorithm presented here).

Noting

$$\begin{aligned} W &= w_0 + w_1\sigma + w_2\rho + w_3\sigma\rho + w_4\rho^2 + w_5\sigma\rho^2 \\ &= (\lambda y_P t - \lambda y_Q \sigma - \lambda y_P \rho) \cdot (-t^2 + y_P y_Q \sigma - t\rho - \rho^2), \end{aligned}$$

we easily check that

$$\begin{aligned} w_0 &= -\lambda y_P t^3 + \lambda y_P y_Q^2 + b\lambda y_P, \\ w_1 &= \lambda y_P^2 y_Q t + \lambda y_Q t^2, \\ w_2 &= \lambda y_P, \\ w_3 &= \lambda y_Q t - \lambda y_P^2 y_Q, \\ w_4 &= 0, \text{ and} \\ w_5 &= \lambda y_Q. \end{aligned}$$

These equations involve a single cubing, 6 additions, and 6 multiplications over $\mathbb{F}_{3^m}$.

Algorithm 16 Multiplication over $\mathbb{F}_{3^{6m}}$ [44].

Input: $U, V \in \mathbb{F}_{3^{6m}}$ with $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2$ and $V = v_0 + v_1\sigma + v_2\rho + v_3\sigma\rho + v_4\rho^2 + v_5\sigma\rho^2$.

Output: $W = U \cdot V$. The algorithm requires 15 multiplications and 67 additions over $\mathbb{F}_{3^m}$.

1. $r_0 \leftarrow u_0 + u_4$; $a_0 \leftarrow r_0 + u_2$; $a_{12} \leftarrow r_0 - u_2$; (3A)
2. $r_0 \leftarrow v_0 + v_4$; $a_3 \leftarrow r_0 + v_2$; $a_{15} \leftarrow r_0 - v_2$; (3A)
3. $r_0 \leftarrow u_0 - u_4$; $a_6 \leftarrow r_0 - u_3$; $a_{18} \leftarrow r_0 + u_3$; (3A)
4. $r_0 \leftarrow v_0 - v_4$; $a_9 \leftarrow r_0 - v_3$; $a_{21} \leftarrow r_0 + v_3$; (3A)
5. $r_0 \leftarrow u_1 + u_5$; $a_1 \leftarrow r_0 + u_3$; $a_{13} \leftarrow r_0 - u_3$; (3A)
6. $r_0 \leftarrow v_1 + v_5$; $a_4 \leftarrow r_0 + v_3$; $a_{16} \leftarrow r_0 - v_3$; (3A)
7. $r_0 \leftarrow u_1 - u_5$; $a_7 \leftarrow r_0 + u_2$; $a_{19} \leftarrow r_0 - u_2$; (3A)
8. $r_0 \leftarrow v_1 - v_5$; $a_{10} \leftarrow r_0 + v_2$; $a_{22} \leftarrow r_0 - v_2$; (3A)
9. $a_2 \leftarrow a_0 + a_1$; $a_5 \leftarrow a_3 + a_4$; $a_8 \leftarrow a_6 + a_7$; (3A)
10. $a_{11} \leftarrow a_9 + a_{10}$; $a_{14} \leftarrow a_{12} + a_{13}$; $a_{17} \leftarrow a_{15} + a_{16}$; (3A)
11. $a_{20} \leftarrow a_{18} + a_{19}$; $a_{23} \leftarrow a_{21} + a_{22}$; (2A)
12. $a_{24} \leftarrow u_4 + u_5$; $a_{25} \leftarrow v_4 + v_5$; (2A)
13. $m_0 \leftarrow a_0 \cdot a_3$; $m_1 \leftarrow a_2 \cdot a_5$; $m_2 \leftarrow a_1 \cdot a_4$; (3M)
14. $m_3 \leftarrow a_6 \cdot a_9$; $m_4 \leftarrow a_8 \cdot a_{11}$; $m_5 \leftarrow a_7 \cdot a_{10}$; (3M)
15. $m_6 \leftarrow a_{12} \cdot a_{15}$; $m_7 \leftarrow a_{14} \cdot a_{17}$; $m_8 \leftarrow a_{13} \cdot a_{16}$; (3M)
16. $m_9 \leftarrow a_{18} \cdot a_{21}$; $m_{10} \leftarrow a_{20} \cdot a_{23}$; $m_{11} \leftarrow a_{19} \cdot a_{22}$; (3M)
17. $m_{12} \leftarrow u_4 \cdot v_4$; $m_{13} \leftarrow a_{24} \cdot a_{25}$; $m_{14} \leftarrow u_5 \cdot v_5$; (3M)
18. **if** $b = 1$ **then**
19. $t_0 \leftarrow m_0 + m_4 + m_{12}$; $t_1 \leftarrow m_2 + m_{10} + m_{14}$; (4A)
20. $t_2 \leftarrow m_6 + m_{12}$; $t_3 \leftarrow -m_8 - m_{14}$; (2A)
21. $t_4 \leftarrow m_7 + m_{13}$; $t_5 \leftarrow t_3 + m_2$; (2A)
22. $t_6 \leftarrow t_2 - m_0$; $t_7 \leftarrow t_3 - m_2 + m_5 + m_{11}$; (4A)
23. $t_8 \leftarrow t_2 + m_0 - m_3 - m_9$; (3A)
24. $w_0 \leftarrow -t_0 + t_1 - m_3 + m_{11}$; (3A)
25. $w_1 \leftarrow t_0 + t_1 - m_1 + m_5 + m_9 - m_{13}$; (5A)
26. $w_2 \leftarrow t_5 + t_6$; (1A)
27. $w_3 \leftarrow t_5 - t_6 + t_4 - m_1$; (3A)
28. $w_4 \leftarrow t_7 + t_8$; (1A)
29. $w_5 \leftarrow t_7 - t_8 + t_4 + m_1 - m_4 - m_{10}$; (5A)
30. **else**
31. $t_0 \leftarrow m_4 + m_8 + m_{14}$; $t_1 \leftarrow m_6 + m_{12}$; (3A)
32. $t_2 \leftarrow t_1 + m_{10}$; $t_3 \leftarrow m_2 + m_{14}$; (2A)
33. $t_4 \leftarrow t_3 - m_8$; $t_5 \leftarrow -m_0 + m_6 - m_{12}$; (3A)
34. $t_6 \leftarrow -t_3 + m_5 - m_8 + m_{11}$; (3A)
35. $t_7 \leftarrow t_1 + m_0 - m_3 - m_9$; $t_8 \leftarrow m_1 + m_{13}$; (4A)
36. $w_0 \leftarrow t_0 - t_2 + m_5 - m_9$; (3A)
37. $w_1 \leftarrow t_0 + t_2 + m_3 - m_7 + m_{11} - m_{13}$; (5A)
38. $w_2 \leftarrow t_4 + t_5$; (1A)
39. $w_3 \leftarrow t_4 - t_5 - t_8 + m_7$; (3A)
40. $w_4 \leftarrow t_6 + t_7$; (1A)
41. $w_5 \leftarrow t_6 - t_7 + t_8 - m_4 + m_7 - m_{10}$; (5A)
42. **end if**
43. **return** $w_0 + w_1\sigma + w_2\rho + w_3\sigma\rho + w_4\rho^2 + w_5\sigma\rho^2$;

Algorithm 17 Computation of $(u_0 + u_1\sigma + u_2\rho) \cdot (v_0 + v_1\sigma + v_2\rho + v_3\sigma\rho + v_4\rho^2 + v_5\sigma\rho^2)$.

Input: U, $V \in \mathbb{F}_{3^{6m}}$ with $U = u_0 + u_1\sigma + u_2\rho$ and $V = v_0 + v_1\sigma + v_2\rho + v_3\sigma\rho + v_4\rho^2 + v_5\sigma\rho^2$.
Output: $W = U \cdot V$. The algorithm requires 12 multiplications and 51 additions over $\mathbb{F}_{3^m}$.

1. $a_0 \leftarrow u_0 + u_2$; $a_{10} \leftarrow u_0 - u_2$; (2A)
2. $r_1 \leftarrow v_0 + v_4$; $a_2 \leftarrow r_1 + v_2$; $a_{12} \leftarrow r_1 - v_2$; (3A)
3. $r_0 \leftarrow v_0 - v_4$; $a_7 \leftarrow r_0 - v_3$; $a_{17} \leftarrow r_0 + v_3$; (3A)
4. $r_2 \leftarrow v_1 + v_5$; $a_3 \leftarrow r_2 + v_3$; $a_{13} \leftarrow r_2 - v_3$; (3A)
5. $a_5 \leftarrow u_1 + u_2$; $a_{15} \leftarrow u_1 - u_2$; (2A)
6. $r_0 \leftarrow v_1 - v_5$; $a_8 \leftarrow r_0 + v_2$; $a_{18} \leftarrow r_0 - v_2$; (3A)
7. $a_1 \leftarrow a_0 + u_1$; $a_4 \leftarrow a_2 + a_3$; $a_6 \leftarrow u_0 + a_5$; (3A)
8. $a_9 \leftarrow a_7 + a_8$; $a_{11} \leftarrow a_{10} + u_1$; $a_{14} \leftarrow a_{12} + a_{13}$; (3A)
9. $a_{16} \leftarrow u_0 + a_{15}$; $a_{19} \leftarrow a_{17} + a_{18}$; (2A)
10. $m_0 \leftarrow a_0 \cdot a_2$; $m_1 \leftarrow a_1 \cdot a_4$; $m_2 \leftarrow u_1 \cdot a_3$; (3M)
11. $m_3 \leftarrow u_0 \cdot a_7$; $m_4 \leftarrow a_6 \cdot a_9$; $m_5 \leftarrow a_5 \cdot a_8$; (3M)
12. $m_6 \leftarrow a_{10} \cdot a_{12}$; $m_7 \leftarrow a_{11} \cdot a_{14}$; $m_8 \leftarrow u_1 \cdot a_{13}$; (3M)
13. $m_9 \leftarrow u_0 \cdot a_{17}$; $m_{10} \leftarrow a_{16} \cdot a_{19}$; $m_{11} \leftarrow a_{15} \cdot a_{18}$; (3M)
14. **if** $b = 1$ **then**
15. $t_0 \leftarrow m_0 + m_4$; $t_1 \leftarrow m_2 + m_{10}$; (2A)
16. $t_2 \leftarrow -m_8 + m_2$; $t_3 \leftarrow m_6 - m_0$; (2A)
17. $t_4 \leftarrow -m_8 - m_2 + m_5 + m_{11}$; (3A)
18. $t_5 \leftarrow m_6 + m_0 - m_3 - m_9$; (3A)
19. $w_0 \leftarrow -t_0 + t_1 - m_3 + m_{11}$; (3A)
20. $w_1 \leftarrow t_0 + t_1 - m_1 + m_5 + m_9$; (4A)
21. $w_2 \leftarrow t_2 + t_3$; (1A)
22. $w_3 \leftarrow t_2 - t_3 + m_7 - m_1$; (3A)
23. $w_4 \leftarrow t_4 + t_5$; (1A)
24. $w_5 \leftarrow t_4 - t_5 + m_7 + m_1 - m_4 - m_{10}$; (5A)
25. **else**
26. $t_0 \leftarrow m_4 + m_8$; $t_1 \leftarrow m_6 + m_{10}$; (2A)
27. $t_2 \leftarrow m_2 - m_8$; $t_3 \leftarrow -m_0 + m_6$; (2A)
28. $t_4 \leftarrow -m_2 + m_5 - m_8 + m_{11}$; (3A)
29. $t_5 \leftarrow m_6 + m_0 - m_3 - m_9$; (3A)
30. $w_0 \leftarrow t_0 - t_1 + m_5 - m_9$; (3A)
31. $w_1 \leftarrow t_0 + t_1 + m_3 - m_7 + m_{11}$; (4A)
32. $w_2 \leftarrow t_2 + t_3$; (1A)
33. $w_3 \leftarrow t_2 - t_3 - m_1 + m_7$; (3A)
34. $w_4 \leftarrow t_4 + t_5$; (1A)
35. $w_5 \leftarrow t_4 - t_5 + m_1 - m_4 + m_7 - m_{10}$; (5A)
36. **end if**
37. **return** $w_0 + w_1\sigma + w_2\rho + w_3\sigma\rho + w_4\rho^2 + w_5\sigma\rho^2$;

Algorithm 18 Computation of $(u_0+u_1\sigma+u_2\rho-\rho^2)\cdot(v_0+v_1\sigma+v_2\rho-\rho^2)$.

Input: $U = (u_0+u_1\sigma+u_2\rho-\rho^2)$ and $V = v_0+v_1\sigma+v_2\rho-\rho^2 \in \mathbb{F}_{3^{6m}}$.
Output: $W = U \cdot V \in \mathbb{F}_{3^{6m}}$.

1. $a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow u_0 + u_2$; $a_2 \leftarrow u_1 + u_2$; (3A)
2. $a_3 \leftarrow v_0 + v_1$; $a_4 \leftarrow v_0 + v_2$; $a_5 \leftarrow v_1 + v_2$; (3A)
3. $a_6 \leftarrow u_2 + v_2$; (1A)
4. $m_1 \leftarrow u_0 \cdot v_0$; $m_2 \leftarrow u_1 \cdot v_1$; $m_3 \leftarrow u_2 \cdot v_2$; (3M)
5. $m_4 \leftarrow a_0 \cdot a_3$; $m_5 \leftarrow a_1 \cdot a_4$; $m_6 \leftarrow a_2 \cdot a_5$; (3M)
6. $w_0 \leftarrow m_1 - m_2 - ba_6$; (2A)
7. $w_1 \leftarrow m_4 - m_1 - m_2$; (2A)
8. $w_2 \leftarrow m_5 - m_1 - m_3 - a_6 + b$; (4A)
9. $w_3 \leftarrow m_6 - m_2 - m_3$; (2A)
10. $w_4 \leftarrow 1 + m_3 - u_0 - v_0$; (3A)
11. $w_5 \leftarrow -u_1 - v_1$; (1A)
12. **return** $w_0 + w_1\sigma + w_2\rho + w_3\sigma\rho + w_4\rho^2 + w_5\sigma\rho^2$;

Algorithm 19 First multiplication of the reversed-loop η_T pairing calculation.

Input: $U = \lambda y_P t - \lambda y_Q\sigma - \lambda y_P\rho$ and $V = -t^2 + y_P y_Q\sigma - t\rho - \rho^2$.
Output: $W = U \cdot V \in \mathbb{F}_{3^{6m}}$.

1. $m_0 \leftarrow y_Q \cdot t$; $m_1 \leftarrow y_P \cdot y_Q$; $m_2 \leftarrow y_P \cdot m_1$; (3M)
2. $a_0 \leftarrow \lambda m_0 + \lambda m_2$; (1A)
3. $c_0 \leftarrow t^3$; (1C)
4. $m_3 \leftarrow a_0 \cdot t$; $m_4 \leftarrow y_P \cdot c_0$; $m_5 \leftarrow y_Q \cdot m_1$; (3M)
5. $w_0 \leftarrow -\lambda m_4 + \lambda m_5 + b\lambda y_P$; (2A)
6. $w_1 \leftarrow m_3$;
7. $w_2 \leftarrow \lambda y_P$; (1A)
8. $w_3 \leftarrow \lambda m_0 - \lambda m_2$; (1A)
9. $w_5 \leftarrow \lambda y_Q$; (1A)
10. **return** $w_0 + w_1\sigma + w_2\rho + w_3\sigma\rho + w_5\sigma\rho^2$;