An Algorithm for the η_T Pairing Calculation in Characteristic Three and its Hardware Implementation

Jean-Luc Beuchat[1], Masaaki Shirase[2], Tsuyoshi Takagi[2], and Eiji Okamoto[1]
[1]University of Tsukuba, Japan
[2]Future University-Hakodate, Japan

Abstract

In this paper, we propose a modified η_T pairing algorithm in characteristic three which does not need any cube root extraction. We also discuss its implementation on a low cost platform which hosts an Altera Cyclone II FPGA device. Our pairing accelerator is ten times faster than previous known FPGA implementations in characteristic three.

Keywords: Tate pairing, η_T pairing, characteristic three, elliptic curve, hardware accelerator, FPGA.

1. Introduction

Since the introduction of pairings over (hyper)elliptic curves in constructive cryptographic applications, an ever increasing number of protocols based on Weil or Tate pairings have appeared in the literature: identity-based encryption [8], short signature [10], and efficient broadcast encryption [9] to mention but a few. Nowadays pairing-based cryptosystems have become a central research topic in cryptography.

Miller's algorithm [19] was the only way to compute the Tate pairing until 2002, where significant improvements were independently proposed by Barreto *et al.* [5] and Galbraith *et al.* [13]. One year later, Duursma and Lee gave a closed formula in the case of characteristic three [11]. They described an iterative scheme involving additions, multiplications, cubing operations, and cube root extractions over $\mathbb{F}_{3^m}$. This work was then extended by Kwon, who proposed a closed formula for the Tate pairing computation for supersingular elliptic curves over $\mathbb{F}_{2^m}$ with odd dimension m [18]. Furthermore, he proved that both his algorithm and Duursma-Lee algorithm can be modified so that no inverse Frobenius map (*i.e.* square root in characteristic two or cube root in characteristic three) is required.

Fong *et al.* showed that extracting a square root in $\mathbb{F}_{2^m}$ requires approximately the time of a field multiplication and proposed an improved scheme for trinomials [12]. Barreto extended this approach to cube root in characteristic three [3]: if $\mathbb{F}_{3^m}$ admits an irreducible trinomial $x^m + ax^k + b$ $(a, b \in \{-1, 1\})$ with the property $k \equiv m \pmod 3$, then five shifts and five additions allow to implement this operation. However, these algorithms restrict the choice of curves and it seems interesting to design pairing algorithms without inverse Frobenius maps. Hardware implementations also benefit from such pairing algorithms: removing the inverse Frobenius maps allows to design simpler arithmetic and logic units.

By introducing the η_T pairing, Barreto *et al.* reduced the number of iterations of Duursma-Lee algorithm by half [4]. However, this algorithm reintroduces inverse Frobenius maps. Recently, Shu *et al.* described how to get rid of square roots in characteristic two [22]. In this paper, we introduce a modified η_T pairing algorithm in characteristic three which does not require any cube root (Section 2). Then, we discuss its hardware implementation on a low cost Field Programmable Gate Array (FPGA) board hosting Altera Cyclone II technology (Section 3) and we compare this pairing accelerator against several software and hardware architectures reported in the literature (Section 4).

2. An Algorithm for the η_T Pairing Calculation

Let E be an elliptic curve over $\mathbb{F}_q$, where q is a power of a prime number. A formal symbol (P) is defined for each point P of the curve. A divisor D on E is then a finite linear combination of such symbols with integer coefficients: $D = \sum_j a_j(P_j)$, $a_j \in \mathbb{Z}$. The degree of a divisor is defined by $\deg(\sum_j a_j(P_j)) = \sum_j a_j \in \mathbb{Z}$. For an introduction to divisors, we refer the reader to [23]. Let $l > 0$ be an integer relatively prime to q. The least positive integer k satisfying $q^k \equiv 1 \pmod l$ is called *embedding degree* or *security multiplier*. Let $E(\mathbb{F}_q)[l]$ be the set of points $P \in E(\mathbb{F}_q)$ such that $lP = \mathcal{O}$, where $\mathcal{O}$ is the point at infinity. Consider $P \in E(\mathbb{F}_q)[l]$ and $Q \in E(\mathbb{F}_{q^k})[l]$. The reduced Tate pairing is the map $e_l : E(\mathbb{F}_q)[l] \times E(\mathbb{F}_{q^k})[l] \rightarrow \mathbb{F}^*_{q^k}$, given by

$$e_l(P, Q) = f_{l,P}(D_Q)^{\frac{q^k-1}{l}}, \qquad (1)$$

where $f_{l,P}$ is a rational function on E whose divisor is equivalent to $l(P) - l(\mathcal{O})$, and D_Q is a divisor of degree 0 equivalent to $(Q) - (\mathcal{O})$. $f_{l,P}$ and D_Q have disjoint supports. The computation of the $(q^k - 1)/l$-th power is referred to as *final exponentiation*. The reduced Tate pairing satisfies the following properties:

- Bilinearity: let a be an integer; then $e_l(aP, Q) = e_l(P, aQ) = e_l(P, Q)^a$, for all $P \in E(\mathbb{F}_q)[l]$ and $Q \in E(\mathbb{F}_{q^k})[l]$.
- Non-degeneracy. If $e_l(P, Q) = 1$ for all $Q \in E(\mathbb{F}_{q^k})[l]$, then $P = \mathcal{O}$.

Equation (1) was initially computed according to an algorithm introduced by Miller in the context of Weil pairing [19]. Several improvements have been proposed since 2002 (see for example [5, 13, 11, 18]). Barreto *et al.* [5] proved that the reduced pairing can be computed as $e_l(P, Q) = f_{l,P}(Q)^{\frac{q^k-1}{l}}$, where $f_{l,P}$ is evaluated on a point rather than on a divisor. In the same paper, the authors exploited a distortion map to further enhance Miller's algorithm.

This work is devoted to the computation of pairing in characteristic three (i. e. $q = 3^m$, where m is odd). Let E^b be a supersingular elliptic curve over $\mathbb{F}_{3^m}$:

$$E^b : y^2 = x^3 - x + b, \text{ with } b \in \{-1, 1\}.$$

The distortion map $\psi : E^b(\mathbb{F}_{3^m}) \rightarrow E^b(\mathbb{F}_{3^{6m}})$ is then defined as follows: $\psi(Q) = \psi(x_q, y_q) = (-x_q + \rho, y_q\sigma)$, where σ and ρ belong to $\mathbb{F}_{3^{6m}}$ and respectively satisfy $\sigma^2 = -1$ and $\rho^3 = \rho + b$. The modified Tate pairing $\hat{e}(P, Q)$ is then given by: $\hat{e}(P, Q) = e_l(P, \psi(Q))$. Note that $\{1, \sigma, \rho, \sigma\rho, \rho^2, \sigma\rho^2\}$ is a basis of $\mathbb{F}_{3^{6m}}$ over $\mathbb{F}_{3^m}$. We will therefore represent an element $A \in \mathbb{F}_{3^{6m}}$ as $A = (a_0, a_1, a_2, a_3, a_4, a_5) = a_0 + a_1\sigma + a_2\rho + a_3\sigma\rho + a_4\rho^2 + a_5\sigma\rho^2$, where the a_i's belong to $\mathbb{F}_{3^m}$. This representation is equivalent to a tower extension of $\mathbb{F}_{3^m}$ (see for instance [17]): $\mathbb{F}_{3^{2m}} = \mathbb{F}_{3^m}[y]/(y^2 + 1)$ and $\mathbb{F}_{3^{6m}} = \mathbb{F}_{3^{2m}}[z]/(z^3 - z - b)$, where $y^2 + 1$ and $z^3 - z - b$ are respectively irreducible polynomials over $\mathbb{F}_{3^m}$ and $\mathbb{F}_{3^{2m}}$. This tower field representation allows one to replace arithmetic over $\mathbb{F}_{3^{6m}}$ by arithmetic over $\mathbb{F}_{3^m}$.

Barreto *et al.* defined the η_T pairing as $\eta_T(P, Q) = f_{T,P}(\psi(Q))$, for some $T \in \mathbb{Z}$ [4]. This formula does not always give a non-degenerate, bilinear pairing. However, Barreto *et al.* described some cases where $\eta_T(P, Q)^W$ is a non-degenerate and bilinear map (a final exponentiation is therefore required for pairing-based cryptosystems). In such cases, this approach reduces the number of iterations by half (Algorithm 1). In characteristic three, the relationship between the η_T pairing and the modified Tate pairing is given by:

$$\left(\eta_T(P, Q)^W\right)^{3T^2} = \hat{e}(P, Q)^Z \qquad (2)$$

where $T = -b3^{\frac{m+1}{2}} - 1$, $Z = -b3^{\frac{m+3}{2}}$, and $W = (3^{3m} - 1)(3^m + 1)(3^m - b3^{\frac{m+1}{2}} + 1)$. Let $v = \eta_T(P, Q)^W$. The modified Tate pairing can be computed as follows:

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

This method is more efficient than the one proposed by Barreto *et. al* in [4]. $\eta_T(P, Q)$ can be calculated according to Algorithm 1. As mentioned in Section 1, this scheme involves two cube root extractions at each iteration.

Algorithm 1 Computation of η_T pairing in characteristic three [4].

```
Input: $\tilde{P} = (\tilde{x}_p, \tilde{y}_p)$ and $\tilde{Q} = (\tilde{x}_q, \tilde{y}_q) \in E^b(\mathbb{F}_{3^m})[l]$.
    The algorithm requires $\tilde{R}_0$ and $\tilde{R}_1 \in \mathbb{F}_{3^{6m}}$, as well as
    $\tilde{r}_0 \in \mathbb{F}_{3^m}$ for intermediate computations.
Output: $\eta_T(\tilde{P}, \tilde{Q})$
 1: if $b = 1$ then
 2:     $\tilde{y}_p \leftarrow -\tilde{y}_p$;
 3: end if
 4: $\tilde{r}_0 \leftarrow \tilde{x}_p + \tilde{x}_q + b$;
 5: $\tilde{R}_0 \leftarrow -\tilde{y}_p\tilde{r}_0 + \tilde{y}_q\sigma + \tilde{y}_p\rho$;
 6: for $i = 0$ to $(m-1)/2$ do
 7:     $\tilde{r}_0 \leftarrow \tilde{x}_p + \tilde{x}_q + b$;
 8:     $\tilde{R}_1 \leftarrow -\tilde{r}_0^2 + \tilde{y}_p\tilde{y}_q\sigma - \tilde{r}_0\rho - \rho^2$;
 9:     $\tilde{R}_0 \leftarrow \tilde{R}_0\tilde{R}_1$;
10:     $\tilde{x}_p \leftarrow \tilde{x}_p^{1/3}$; $\tilde{y}_p \leftarrow \tilde{y}_p^{1/3}$; $\tilde{x}_q \leftarrow \tilde{x}_q^3$; $\tilde{y}_q \leftarrow \tilde{y}_q^3$;
11: end for
12: Return $\tilde{R}_0$;
```

We propose here a modified η_T pairing algorithm in characteristic three which computes $R_0 = \tilde{R}_0^{3^{i+1}}$ at step i (Algorithm 2). This trick allows one to get rid of cube roots and our algorithm returns $\eta_T(P, Q)^{3^{(m+1)/2}}$. A proof of correctness of this new scheme is provided in an extended version of this paper [7]. Let us describe now how to implement the original $\eta_T(P, Q)$ pairing with our algorithm. Recall that tripling a point requires only four cubing operations in characteristic three for supersingular elliptic curves (see for instance [15]): $3(x_p, y_p) = (x_p^9 - b, -y_p^9)$. Therefore, we suggest to compute $3^{\frac{m-1}{2}}P$ by means of $2(m-1)$ cubings and to take advantage of the bilinearity of $\eta_T(P, Q)^W$:

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

Note that cubing over $\mathbb{F}_{3^m}$ is efficiently performed in hardware (Section 3.2). A postprocessing step involving a 3^m-th root is further required. However, this operation is carried out by means of six additions (or subtractions) and a negation over $\mathbb{F}_{3^m}$. Assume that $b = 1$. Raising $\eta_T(P,Q)^{3^{(m+1)/2}}$ to the W-th power is based on the following observation:

$$W = 3^{5m} + 2 \cdot 3^{4m} + 3^{3m} + 3^{m+(m+1)/2} + 3^{(m+1)/2} - (3^{4m+(m+1)/2} + 3^{3m+(m+1)/2} + 3^{2m} + 2 \cdot 3^m + 1).$$

This operation requires 11 multiplications and a single inversion over $\mathbb{F}_{3^{6m}}$, as well as additions over $\mathbb{F}_{3^m}$.

Algorithm 2 Proposed computation of $\eta_T(P,Q)^{3^{(m+1)/2}}$.

Input: $P = (x_p, y_p)$ and $Q = (x_q, y_q) \in E^b(\mathbb{F}_{3^m})[l]$. The algorithm requires R_0 and $R_1 \in \mathbb{F}_{3^{6m}}$, as well as $r_0 \in \mathbb{F}_{3^m}$ and $d \in \mathbb{F}_3$ for intermediate computations.

Output: $\eta_T(P,Q)^{3^{(m+1)/2}}$

```
1: if $b = 1$ then
2:     $y_p \leftarrow -y_p$;
3: end if
4: $r_0 \leftarrow x_p + x_q + b$;
5: $d \leftarrow b$;
6: $R_0 \leftarrow -y_p r_0 + y_q \sigma + y_p \rho$;
7: for $i = 0$ to $(m-1)/2$ do
8:     $r_0 \leftarrow x_p + x_q + d$;
9:     $R_1 \leftarrow -r_0^2 + y_p y_q \sigma - r_0 \rho - \rho^2$;
10:    $R_0 \leftarrow (R_0 R_1)^3$;
11:    $y_p \leftarrow -y_p$;
12:    $x_q \leftarrow x_q^9$; $y_q \leftarrow y_q^9$;
13:    $d \leftarrow (d - b) \bmod 3$;
14: end for
15: Return $R_0$;
```

3. Hardware Implementation

This section describes the hardware implementation of Algorithm 2 for the field $\mathbb{F}_3[x]/(x^{97} + x^{12} + 2)$ and the curve $y^2 = x^3 - x + 1$ (*i.e.* $b = 1$). This choice of parameters allows us to easily compare our work against the many pairing accelerators for $m = 97$ described in the open literature. A first approach consists in designing an architecture able to compute both pairing and final exponentiation. However, it does not allow to take advantage of the constant coefficients of R_1 (see Algorithms 1 and 2) to optimize the multiplication over $\mathbb{F}_{3^{6m}}$. Therefore, we suggest to design a pairing accelerator evaluating $\eta_T(P,Q)^{3^{(m+1)/2}}$ and a coprocessor responsible for final exponentiation working in parallel. In this paper, we will only focus on the computation of the modified η_T pairing. Algorithm 2 and final exponentiation require respectively $(m-1)/2 + 1 = 49$ and 11 multiplications over $\mathbb{F}_{3^{6m}}$. The inversion over $\mathbb{F}_{3^{6m}}$ can be replaced by a few multiplications and additions over $\mathbb{F}_{3^m}$ and a single inversion over $\mathbb{F}_{3^m}$ [17]. Consequently, the final exponentiation requires less operations (and thus less hardware) than the computation of the η_T pairing.

In order to compare our architecture against software implementations, we decided to choose a design board whose price is comparable to that of an entry level desktop computer. We selected a DE2 development and education board [2] which costs \$495 and hosts an Altera Cyclone II EP2C35F672C6 FPGA. Note that Altera provides free simulation and design tools for the Cyclone II family. The smallest unit of logic in a Cyclone II is called *Logic Element* (LE). Each LE includes a 4-input Look-Up Table (LUT), carry logic, and a programmable register. A Cyclone II EP2C35F672C6 device contains for instance 33216 LEs. Readers who are not familiar with Cyclone II devices should refer to [1] for further details. Since we leave the study of final exponentiation for further work, our pairing accelerator should not utilize all resources of our target FPGA. Thus, we impose a size constraint: our design must require less than 50% of the available configurable logic.

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

Since they are performed component-wise, addition and subtraction over $\mathbb{F}_{3^m}$ are rather straightforward operations. Each element a_i of $\mathbb{F}_3$ is encoded by two bits a_i^L and a_i^H such that [14]: $a_i^L = a_i \bmod 2$ and $a_i^H = a_i \operatorname{div} 2$. Thus, the addition of a_i and b_i on a Cyclone-II FPGA requires two 4-input LUTs. Our processor includes an operator which adds or subtracts up to three elements of $\mathbb{F}_{3^m}$ and stores the result in a register (Figure 1a).

3.2. Cubing over $\mathbb{F}_{3^m}$ and $\mathbb{F}_{3^{6m}}$

Cubing is also a pretty simple arithmetic operation. Since $\mathbb{F}_{3^{6m}}$ is constructed as an extension field of $\mathbb{F}_{3^m}$, the computation of R_0^3 involved in Algorithm 2 is replaced by six cubing, six additions (or subtractions), and a negation over $\mathbb{F}_{3^m}$. Indeed, by noting that $\sigma^3 = -\sigma$, $(\rho^2)^3 = \rho^2 - \rho + 1$, $\rho^3 = \rho + 1$, $(\sigma\rho^2)^3 = -\sigma\rho^2 + \sigma\rho - \sigma$, and $(\sigma\rho)^3 = -\sigma\rho - \sigma$, we obtain: $C^3 = (c_0^3 + c_2^3 + c_4^3) + (-c_1^3 - c_3^3 - c_5^3)\sigma + (c_2^3 - c_4^3)\rho + (-c_3^3 + c_5^3)\sigma\rho + c_4^3\rho^2 + (-c_5^3)\sigma\rho^2$, where $C = (c_0, c_1, c_2, c_3, c_4, c_5)$ belongs to $\mathbb{F}_{3^{6m}}$. Let us now consider the computation of $b(x) = a(x)^3$ over $\mathbb{F}_{3^m}$. We have:

$$b(x) = a(x)^3 = \left(\sum_{i=0}^{m-1} a_i x^{3i} \right) \bmod f(x),$$

where $f(x)$ is a degree m irreducible polynomial over $\mathbb{F}_3$. Since we set $f(x) = x^{97} + x^{12} + 2$, a simple Maple or

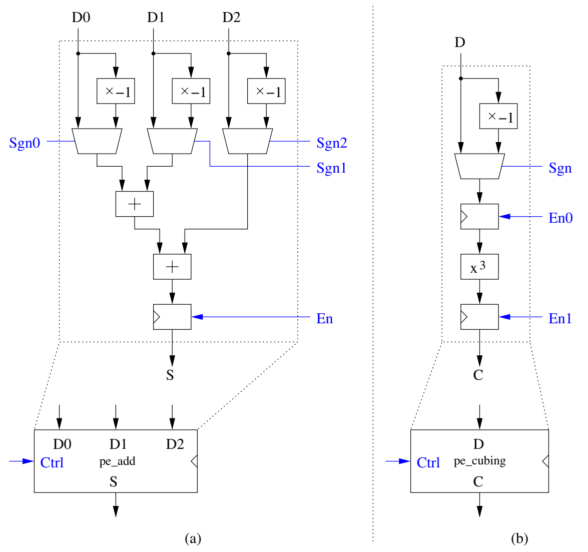

Figure 1. (a) Addition over $\mathbb{F}_{3^m}$. (b) Cubing over $\mathbb{F}_{3^m}$.

Pari program provides us with a closed formula for cubing over $\mathbb{F}_{3^m}$:

$$\begin{array}{lll} b_0 = a_{93} + a_{89} + a_0, & b_2 = a_{33}, & \ldots \\ b_1 = a_{65} - a_{61}, & b_3 = a_{94} + a_{90} + a_1, & b_{96} = a_{32}. \end{array}$$

The most complex operation involved in cubing is the addition of three elements of $\mathbb{F}_3$. Therefore, the critical path includes only two LUTs. Our pairing accelerator embeds a single cubing unit (Figure 1b) which computes either $a(x)^3$ or $(-a(x))^3$ according to a control bit. In order to guarantee a short critical path, the operator includes two pipeline stages. It is worth noticing that the only degree 97 irreducible trinomial over $\mathbb{F}_3$ allowing a simple cube root extraction [3] has a more complex closed formula for cubing. Thus, Algorithm 2 offers additional flexibility to select parameters leading to the smallest hardware operators.

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

We designed a Most Significant Element (MSE) first multiplier over $\mathbb{F}_{3^m}$ based on a paper by Song and Parhi [24] to compute $a(x)b(x) \bmod f(x)$. At step i we compute a degree $(m + D - 2)$ polynomial $t(x)$ which is the sum of D partial products: $t(x) = \sum_{j=0}^{D-1} a_{Di+j}x^j b(x)$. A degree $(m + D - 1)$ polynomial $s(x)$, updated according to the celebrated Horner's rule, allows to accumulate the partial products:

$$s(x) \leftarrow t(x) + x^D \cdot (s(x) \bmod f(x)).$$

Thus, after $\lceil m/D \rceil$ steps, this algorithm returns a degree $(m + D - 1)$ polynomial $s(x)$, which is congruent with $a(x)b(x)$ modulo $f(x)$. The circuit described by Song and Parhi requires dedicated hardware to compute $p(x) = s(x) \bmod f(x)$ [24]. We suggest to achieve this final modulo $f(x)$ reduction by performing an additional iteration with $a_{-j} = 0, 1 \le j \le D$. Since $t(x)$ is now equal to zero, we have: $s(x) = x^D \cdot (a(x)b(x) \bmod f(x))$. Therefore, it suffices to consider the m most significant coefficients of $s(x)$ to get the result: $p(x) = s(x)/x^D$. Algorithm 3 summarizes this multiplication scheme. Synthesis results indicate that for $D = 3$ and $D = 4$, such a multiplier requires respectively 1170 and 1560 LEs. According to our size constraint, up to ten multipliers can be included in our pairing accelerator.

Algorithm 3 MSE multiplication over $\mathbb{F}_{3^m}$.

Input: A degree m monic polynomial $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1x + f_0$, two degree $(m-1)$ polynomial $a(x)$, and $b(x)$. We assume that $a_{-j} = 0, 1 \le j \le D$. The algorithm requires a degree $(m + D - 1)$ polynomial $s(x)$ as well as a degree $(m + D - 2)$ polynomial $t(x)$ for intermediate computations.

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

```
1: $s(x) \leftarrow 0$;
2: for $i$ in $\lceil m/D \rceil - 1$ downto $-1$ do
3:     $t(x) \leftarrow \sum_{j=0}^{D-1} a_{Di+j}x^j b(x)$;
4:     $s(x) \leftarrow t(x) + x^D \cdot (s(x) \bmod f(x))$;
5: end for
6: $p(x) \leftarrow s(x)/x^D$;
```

3.4. Multiplication over $\mathbb{F}_{3^{6m}}$

The cost of Algorithm 2 is dominated by the multiplication of R_0 by R_1 over $\mathbb{F}_{3^{6m}}$. By applying Karatsuba-Ofman's algorithm (see for instance [25]) and taking advantage of the constant coefficients of R_1, the product R_0R_1 could be computed in parallel by means of 13 multiplications and 50 additions (or subtractions) over $\mathbb{F}_{3^m}$ [6]. Two further multiplications are needed to compute y_py_q as well as r_0^2 (a straightforward modification of the scheduling of Algorithm 2 allows to compute r_0^2, y_py_q, and R_0R_1 in parallel). However, according to our size constraints, it is impossible to implement 15 multipliers on our target FPGA. Furthermore, our processor embeds only three adders over $\mathbb{F}_{3^m}$ and scheduling 50 additions could be a complex task. We propose here an algorithm which offers a better trade-off between the number of additions and multiplications.

Let $A = a_0 + a_1\sigma + a_2\rho + a_3\sigma\rho + a_4\rho^2 + a_5\sigma\rho^2$ and $C = c_0+c_1\sigma+c_2\rho+c_3\sigma\rho+c_4\rho^2+c_5\sigma\rho^2$ be two elements of $\mathbb{F}_{3^{6m}}$. We write each coefficient c_i as a sum of two elements

$c_i^{(0)}$ and $c_i^{(1)} \in \mathbb{F}_{3^m}$. Thanks to this notation we define the product $C = A \cdot (-r_0^2 + y_p y_q \sigma - r_0 \rho - \rho^2)$ as follows:

$$\begin{aligned}
c_0^{(0)} &= -a_4 r_0 - a_2, & c_0^{(1)} &= -a_0 r_0^2 - a_1 y_p y_q,\\
c_1^{(0)} &= -a_5 r_0 - a_3, & c_1^{(1)} &= a_0 y_p y_q - a_1 r_0^2,\\
c_2^{(0)} &= -a_0 r_0 - a_4 + c_0^{(0)}, & c_2^{(1)} &= -a_2 r_0^2 - a_3 y_p y_q,\\
c_3^{(0)} &= -a_1 r_0 - a_5 + c_1^{(0)}, & c_3^{(1)} &= a_2 y_p q_q - a_3 r_0^2,\\
c_4^{(0)} &= -a_2 r_0 - a_0 - a_4, & c_4^{(1)} &= -a_4 r_0^2 - a_5 y_p y_q,\\
c_5^{(0)} &= -a_3 r_0 - a_1 - a_5, & c_5^{(1)} &= a_4 y_p y_q - a_5 r_0^2.
\end{aligned}$$

Note that computation of the $c_i^{(0)}$'s, $0 \leq i \leq 5$, requires six multiplications over $\mathbb{F}_{3^m}$ and depends neither on r_0^2 nor on $y_p y_q$. Thus, we can perform eight multiplications over $\mathbb{F}_{3^m}$ in parallel (r_0^2, $y_p y_q$, and $a_i r_0$, $0 \leq i \leq 5$). Consider now $c_0^{(1)}$ and $c_1^{(1)}$ and assume that $(a_0 + a_1)$, as well as $(y_p y_q - r_0^2)$, are stored in registers. Karatsuba-Ofman's algorithm allows to compute $c_0^{(1)}$ and $c_1^{(1)}$ by means of three multiplications and three additions over $\mathbb{F}_{3^m}$:

$$c_0^{(1)} = -a_0 r_0^2 - a_1 y_p y_q, \qquad (4)$$

$$c_1^{(1)} = (a_0 + a_1)(y_p y_q - r_0^2) + a_0 r_0^2 - a_1 y_p y_q. \qquad (5)$$

Therefore, the computation of the $c_i^{(1)}$'s involves nine multiplications over $\mathbb{F}_{3^m}$, which can be carried out in parallel.

Algorithm 4 summarizes this multiplication scheme involving 17 multiplications and 29 additions (or subtractions) over $\mathbb{F}_{3^m}$. Since at most nine multiplications can be performed in parallel, our pairing accelerator hosts nine multipliers over $\mathbb{F}_{3^m}$ and the computation of $R_0 R_1$ involves two multiplication cycles. A careful scheduling allows to share operands between up to three operators, thus saving hardware resources (Table 1): during the first multiplication cycle, M_0, M_1, and M_2 respectively compute $a_0 r_0$, $a_2 r_0$, and $a_4 r_0$. The MSE multiplier described in Section 3.3 stores its first operand in a shift register, and its second operand in a standard register. Since a shift register is more complex (an operand is loaded in parallel, and then shifted), we load the common operand r_0 in this component. At the end of the first cycle, the three standard registers still contain a_0, a_2, and a_4. Therefore it suffices to load r_0^2 in the shift register before starting the second multiplication cycle. Figure 2a describes the operator we designed. This component is connected to the addition/subtraction operator described in Section 3.1 (Figure 2c). Note that the same architecture allows to compute $a_1 r_0$, $a_3 r_0$, $a_5 r_0$, $a_1 y_p y_q$, $a_3 y_p y_q$, and $a_5 y_p y_q$. The five remaining multiplications involve a slightly more complex component (Figure 2b). Two shift registers are required to compute r_0^2 and $y_p y_q$ since there is no common operand. At the end of the first multiplication cycle, a dedicated subtracter computes $y_p y_q - r_0^2$ and stores the result in the shift registers. Three clock cycles are requested to load $(a_0 + a_1)$, $(a_2 + a_3)$, and $(a_4 + a_5)$, which have been computed during the first multiplication cycle (see Algorithm 4). This approach could also be adopted to implement the multiplication of $\tilde{R}_0$ by $\tilde{R}_1$ in Algorithm 1.

Table 1. Multiplication over $\mathbb{F}_{3^m}$: scheduling.

	First cycle	Second cycle
$\mathbf{M_0}$	$a_0 \cdot r_0$	$a_0 \cdot r_0^2$
$\mathbf{M_1}$	$a_2 \cdot r_0$	$a_2 \cdot r_0^2$
$\mathbf{M_2}$	$a_4 \cdot r_0$	$a_4 \cdot r_0^2$
$\mathbf{M_3}$	$a_1 \cdot r_0$	$a_1 \cdot y_p y_q$
$\mathbf{M_4}$	$a_3 \cdot r_0$	$a_3 \cdot y_p y_q$
$\mathbf{M_5}$	$a_5 \cdot r_0$	$a_5 \cdot y_p y_q$
$\mathbf{M_6}$	$r_0 \cdot r_0$	$(a_0 + a_1) \cdot (y_p y_q - r_0^2)$
$\mathbf{M_7}$	$y_p \cdot y_q$	$(a_2 + a_3) \cdot (y_p y_q - r_0^2)$
$\mathbf{M_8}$	–	$(a_4 + a_5) \cdot (y_p y_q - r_0^2)$

3.5. Architecture of the Pairing Accelerator

Figure 2c shows the architecture of our hardware accelerator. Inputs and outputs, as well as intermediate results, are stored in registers implemented using embedded memory blocks available in the FPGA. The control unit mainly consists of a ROM containing the microcode of Algorithm 2 and a program counter. The size of the microcode depends on D, the number of coefficients processed at each clock cycle by a multiplier over $\mathbb{F}_{3^m}$. For $D = 3$, the initialization step of Algorithm 2 (copy of inputs in registers of multipliers and computation of r_0, d, and R_0) and the main loop respectively require 47 and 98 clock cycles. Since $m = 97$, a pairing is completed after $47 + 98 \cdot (m - 1)/2 = 47 + 98 \cdot 49 = 4849$ clock cycles. For $D = 4$, the initialization and the main loop respectively involve 39 and 80 microinstructions. Thus, the computation of a pairing requires $39 + 80 \cdot 49 = 3959$ clock cycles.

4. Results and Comparisons

The proposed architecture was captured in the VHDL language and prototyped on an Altera Cyclone II EP2C35F672C6 device. Both synthesis and place-and-route steps were performed with Quartus II 6.0 Web Edition. VHDL simulations and experiments with a DE2 board were carried out to extensively test our design. The area and the calculation time depend on D, the number of coefficients of a multiplier processed at each clock cycle (Section 3.3). The two rightmost columns of Table 2 summarize our results for $D = 3$ and $D = 4$. When $D = 3$, the pairing accelerator occupies 45% of the LEs, thus meeting our size constraint (Section 3). However, choosing $D = 4$

Algorithm 4 Multiplication over $\mathbb{F}_{3^{6m}}$.

```
Input: $A = a_0 + a_1\sigma + a_2\rho + a_3\sigma\rho + a_4\rho^2 + a_5\sigma\rho^2 \in \mathbb{F}_{3^{6m}}$. $r_0$, $y_p$, and $y_q \in \mathbb{F}_{3^m}$.
Output: $C = A \cdot (-r_0^2 + y_p y_q \sigma - r_o\rho - \rho^2)$
1: Compute in parallel (8 multiplications and 3 additions over $\mathbb{F}_{3^m}$): $p_i \leftarrow a_i r_0$, $0 \le i \le 5$; $p_6 \leftarrow r_0 r_0$; $p_7 \leftarrow y_p y_q$;
   $s_0 \leftarrow a_0 + a_1$; $s_1 \leftarrow a_2 + a_3$; $s_2 \leftarrow a_4 + a_5$;
2: Compute in parallel (7 additions over $\mathbb{F}_{3^m}$):
     $s_4 \leftarrow p_7 - p_6$;   // $y_p y_q - r_0^2$      $c_2 \leftarrow a_4 + p_0$;   // $a_4 + a_0 r_0$      $c_4 \leftarrow a_0 + p_2$;   // $a_0 + a_2 r_0$
     $c_0 \leftarrow a_2 + p_4$;   // $a_2 + a_4 r_0$      $c_3 \leftarrow a_5 + p_1$;   // $a_5 + a_1 r_0$      $c_5 \leftarrow a_1 + p_3$;   // $a_1 + a_3 r_0$
     $c_1 \leftarrow a_3 + p_5$;   // $a_3 + a_5 r_0$
3: Compute in parallel (9 multiplications and 4 additions over $\mathbb{F}_{3^m}$):
     $p_8 \leftarrow a_0 p_6$;   // $a_0 r_0^2$                      $p_{13} \leftarrow s_1 s_4$;   // $(a_2 + a_3)(y_p y_q - r_0^2)$      $c_2 \leftarrow c_2 + c_0$;
     $p_9 \leftarrow a_1 p_7$;   // $a_1 y_p y_q$                    $p_{14} \leftarrow a_4 p_6$;   // $a_4 r_0^2$                      $c_3 \leftarrow c_3 + c_1$;
     $p_{10} \leftarrow s_0 s_4$;   // $(a_0 + a_1)(y_p y_q - r_0^2)$   $p_{15} \leftarrow a_5 p_7$;   // $a_5 y_p y_q$                    $c_4 \leftarrow c_4 + a_4$;
     $p_{11} \leftarrow a_2 p_6$;   // $a_2 r_0^2$                    $p_{16} \leftarrow s_2 s_4$;   // $(a_4 + a_5)(y_p y_q - r_0^2)$      $c_5 \leftarrow c_5 + a_5$;
     $p_{12} \leftarrow a_3 p_7$;   // $a_3 y_p y_q$
4: Compute in parallel (15 additions over $\mathbb{F}_{3^m}$):
     $c_0 \leftarrow -c_0 - p_8 - p_9$;           $c_2 \leftarrow -c_2 - p_{11} - p_{12}$;              $c_4 \leftarrow -c_4 - p_{14} - p_{15}$;
     $c_1 \leftarrow -c_1 + p_{10} + p_8 - p_9$;   $c_3 \leftarrow -c_3 + p_{13} + p_{11} - p_{12}$;   $c_5 \leftarrow -c_5 + p_{16} + p_{14} - p_{15}$;
```

lead to an architecture which requires 56% of the configurable logic. Several researchers described implementations of pairing algorithms on Xilinx Virtex-II Pro FPGAs and reported the area in terms of *slices*. Each slice features two 4-input LUTs, carry logic, wide function multiplexers, and two storage elements. Let us assume that Xilinx design tools try to utilize both LUTs of a slice as often as possible (*i.e.* area optimization). Under this hypothesis, we consider that a slice is roughly equivalent to two LEs in our comparisons.

To our best knowledge, the FPGA-based pairing accelerator described by Shu *et al.* in [22] is the fastest to date. It computes the Tate pairing over $\mathbb{F}_{2^{239}}$ in 34 μs on a Virtex-II Pro 100 device (25287 slices). Ronan *et al.* designed an embedded processor to compute the η_T pairing on genus 2 hyperelliptic curves [20]. This architecture requires 43986 slices on a Virtex-II Pro 125 device and computes a pairing in 749 μs. Kerins *et al.* proposed an implementation of the modified Duursma-Lee algorithm on a Xilinx Virtex-II Pro 125 FPGA [17]. Multiplication over $\mathbb{F}_{3^{6m}}$ is performed according to Karatsuba-Ofman's algorithm. However, since the authors do not take advantage of the constant terms of R_1, this operation requires 18 multiplications over $\mathbb{F}_{3^m}$. Thus, the hardware architecture consists of 18 multipliers and 6 cubing circuits over $\mathbb{F}_{3^{97}}$, along with "a suitable amount of simpler $\mathbb{F}_{3^m}$ arithmetic circuits for performing addition, subtraction, and negation" [17]. The authors claim that roughly 100% of available resources are required to implement their pairing accelerator. We can therefore estimate the cost to 55616 slices [22]. Remember that our target FPGA embeds 33216 LEs. Consequently, even if the final exponentiation unit we left for future work requires 50% of the device, our processor is smaller than the aforementioned solutions. Furthermore, our approach requires a less expensive FPGA technology for which free simulation and design tools are available.

Grabher and Page designed a coprocessor dealing with $\mathbb{F}_{3^m}$ arithmetic, which is controlled by a general purpose processor [14]. Their hardware accelerator embeds a single multiplier over $\mathbb{F}_{3^m}$. Our architecture requires roughly twice as much LEs, while performing up to nine multiplications in parallel.

Several researchers studied the software implementation of pairings on smartcards or mobile phones (see for instance [16] and [21]). For comparison purpose, they often provide the reader with timings on desktop computers. Table 3 summarizes such results which indicate that our FPGA architecture achieves a speedup of 100.

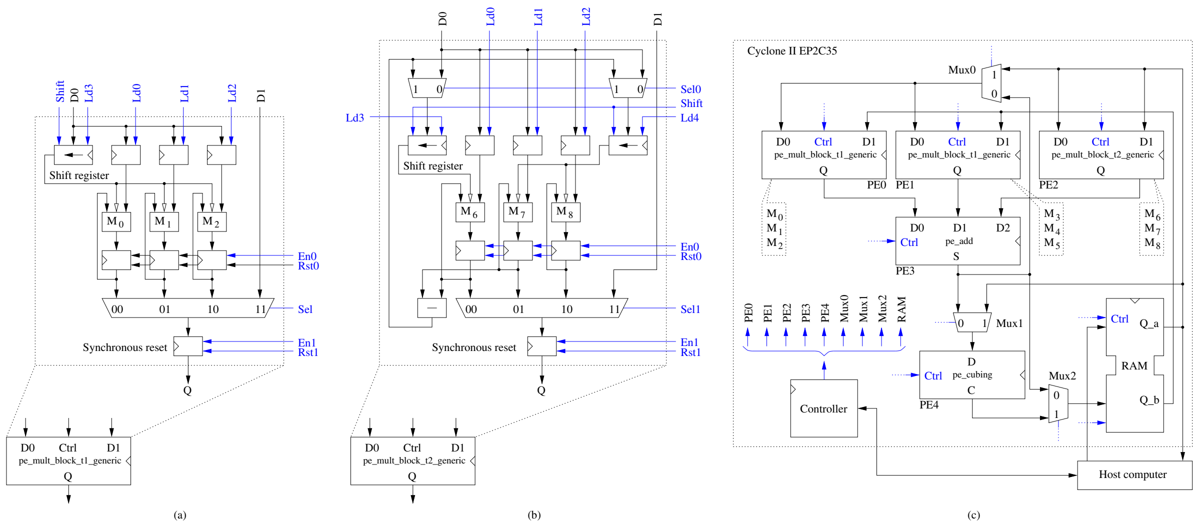

Figure 2. a) and b) Building blocks for multiplication over $\mathbb{F}_{3^{6m}}$. c) Architecture of the η_T pairing accelerator.

Table 3. Comparisons with software implementations on desktop computers.

	Kawahara *et al.* [16]	**Scott *et al.* [21]**	**Proposed architecture**
Algorithm	η_T pairing	η_T pairing	Algorithm 2
Processor	Pentium M	Pentium 4	FPGA
Clock frequency	1.73 GHz	3 GHz	0.149 GHz
Calculation time	10.15 ms	3.7 ms	0.033 ms

5. Conclusions

We have proposed a modified η_T pairing algorithm on supersingular elliptic curves over $\mathbb{F}_{3^m}$ which does not need any cube root. We have then described a pairing accelerator based on a low cost platform hosting an Altera Cyclone II FPGA. Since VHDL simulation and FPGA configuration are performed with free design tools, the price of our system is comparable to that of an entry level desktop computer. Our results demonstrate a one hundred-fold improvement on software implementations, and a ten-fold improvement on the best known FPGA implementation in characteristic three. We achieve the same calculation time than the fastest published accelerator in characteristic two, while requiring less hardware resources. Further work will include the design of a small processing unit responsible for final exponentiation.

Acknowledgement

Authors would like to thank the anonymous reviewers for their useful comments. This work was supported by the New Energy and Industrial Technology Development Organization (NEDO), Japan.

References

[1] Altera. *Cyclone II Device Handbook*, 2006. Available from Altera's web site (`http://altera.com`).

[2] Altera. *DE2 Development and Education Board – User Manual*, 2006. Available from Altera's web site (`http://altera.com`).

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

[4] P. S. L. M. Barreto, S. Galbraith, C. Ó hÉigeartaigh, and M. Scott. Efficient pairing computation on supersingular Abelian varieties. Cryptology ePrint Archive, Report 2004/375, 2004.

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

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

[7] J.-L. Beuchat, M. Shirase, T. Takagi, and E. Okamoto. An algorithm for the η_T pairing calculation in characteristic three and its hardware implementation. Cryptology ePrint Archive, Report 2006/327, 2006.

Table 2. Comparison against previous FPGA implementations. The parameter D refers to the number of coefficients processed at each clock cycle by a multiplier.

	Shu, Kwon, and Gaj [22]	Ronan *et al.* [20]	Grabher and Page [14]	Kerins *et al.* [17]	Proposed architecture	
					$D = 3$	$D = 4$
Algorithm	η_T pairing	η_T pairing	Duursma-Lee	Duursma-Lee	Algorithm 2	
Underlying field	$\mathbb{F}_{2^{239}}$	$\mathbb{F}_{2^{103}}$	$\mathbb{F}_{3^{97}}$	$\mathbb{F}_{3^{97}}$	$\mathbb{F}_{3^{97}}$	
Curve	Elliptic	Hyperelliptic	Elliptic	Elliptic	Elliptic	
FPGA	Virtex-II Pro 100	Virtex-II Pro 125	Virtex-II Pro 4	Virtex-II Pro 125	Cyclone II EP2C35	
Free design tools	No	No	Yes	No	Yes	
Controller	Hardwired logic	Hardwired logic	Microprocessor	Hardwired logic	Hardwired logic	
Multiplier(s)	6 (over $\mathbb{F}_{2^{239}}$)	12 (over $\mathbb{F}_{2^{103}}$)	1 (over $\mathbb{F}_{3^{97}}$)	18 (over $\mathbb{F}_{3^{97}}$)	9 (over $\mathbb{F}_{3^{97}}$)	
Area	25287 slices	43986 slices	4481 slices	55616 slices	14895 LEs	18553 LEs
Clock cycles	–	–	–	12866	4849	3959
Clock frequency	84 MHz	32.3 MHz	150 MHz	15 MHz	149 MHz	147 MHz
Calculation time	34 μs	749 μs	399.4 μs	850 μs	33 μs	27 μs
Final exponentiation	Yes	Yes	No	Yes	No	No

[8] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, number 2139 in Lecture Notes in Computer Science, pages 213–229. Springer, 2001.

[9] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, number 3621 in Lecture Notes in Computer Science, pages 258–275. Springer, 2005.

[10] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, number 2248 in Lecture Notes in Computer Science, pages 514–532. Springer, 2001.

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

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

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

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

[15] K. Harrison, D. Page, and N. P. Smart. Software implementation of finite fields of characteristic three, for use in pairing-based cryptosystems. *LMS Journal of Computation and Mathematics*, 5:181–193, Nov. 2002.

[16] Y. Kawahara, T. Takagi, and E. Okamoto. Efficient implementation of Tate pairing on a mobile phone using Java. Cryptology ePrint Archive, Report 2006/299, 2006.

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

[18] S. Kwon. Efficient Tate pairing computation for supersingular elliptic curves over binary fields. Cryptology ePrint Archive, Report 2004/303, 2004.

[19] V. S. Miller. Short programs for functions on curves. Unpublished manuscript available at http://crypto.stanford.edu/miller/miller.pdf, 1986.

[20] 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 Information Technology: New Generations (ITNG'06)*. IEEE Computer Society, 2006.

[21] M. Scott, N. Costigan, and W. Abdulwahab. Implementing cryptographic pairings on smartcards. Cryptology ePrint Archive, Report 2006/144, 2006.

[22] C. Shu, S. Kwon, and K. Gaj. FPGA accelerated Tate pairing based cryptosystem over binary fields. Cryptology ePrint Archive, Report 2006/179, 2006.

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

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

[25] D. Zuras. More on squaring and multiplying large integers. *IEEE Transactions on Computers*, 43(8):899–908, Aug. 1994.