Arithmetic Operators for Pairing-Based Cryptography

Jean-Luc Beuchat[1], Nicolas Brisebarre[2,3], Jérémie Detrey[3], and Eiji Okamoto[1]

[1] Graduate School of Systems and Information Engineering, University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki, 305-8573, Japan
[2] LAMUSE, Université J. Monnet, 23, rue du Dr P. Michelon, F-42023 Saint-Étienne Cedex, France
[3] LIP/Arénaire (CNRS – ENS Lyon – INRIA – UCBL), ENS Lyon, 46, allée d'Italie, F-69364 Lyon Cedex 07, France

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 first study an accelerator for the η_T pairing over $\mathbb{F}_3[x]/(x^{97} + x^{12} + 2)$. Our architecture is based on a unified arithmetic operator which performs addition, multiplication, and cubing over $\mathbb{F}_{3^{97}}$. This design methodology allows us to design a compact coprocessor (1888 slices on a Virtex-II Pro 4 FPGA) which compares favorably with other solutions described in the open literature. We then describe ways to extend our approach to any characteristic and any extension field.

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

1 Introduction

Introduced in cryptography for code-breaking purpose [11, 22], the Weil and Tate pairings are at the heart of an ever increasing number of protocols since the work of Joux [16] who first discovered their constructive properties. The interested reader should refer to the survey by Dutta, Barua, and Sarkar for further details [9]. According to [14, 20], when dealing with general curves providing common levels of security, the Tate pairing seems to be more efficient than the Weil pairing. Let E be a supersingular elliptic curve over $\mathbb{F}_{p^m}$ (see Theorem V.3.1 of [28] for a definition), 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 [28] for an account on 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 of order ℓ is the map $e_\ell : E(\mathbb{F}_{p^m})[\ell] \times E(\mathbb{F}_{p^{km}})[\ell] \rightarrow \mathbb{F}^*_{p^{km}}$ defined by $e_\ell(P,Q) = f_{\ell,P}(D_Q)^{(p^{km}-1)/\ell}$ (we give here the definition from [3], slightly different from the initial one given in [11]). It satisfies the following properties:

- Non-degeneracy. For all $P \in E(\mathbb{F}_{p^m})[\ell] \setminus \{\mathcal{O}\}$, there is some point $Q \in E(\mathbb{F}_{p^{km}})[\ell]$ such that $e_\ell(P,Q) \neq 1$.
- Bilinearity. For all P, P_1, $P_2 \in E(\mathbb{F}_{p^m})[\ell]$ and Q, Q_1, $Q_2 \in E(\mathbb{F}_{p^{km}})[\ell]$, $e_\ell(P_1+P_2,Q) = e_\ell(P_1,Q)e_\ell(P_2,Q)$ and $e_\ell(P,Q_1+Q_2) = e_\ell(P,Q_1)e_\ell(P,Q_2)$. Hence, for all $P \in E(\mathbb{F}_{p^m})[\ell]$ and $Q \in E(\mathbb{F}_{p^{km}})[\ell]$, and for all $a \in \mathbb{Z}$, $e_\ell([a]P,Q) = e_\ell(P,[a]Q) = e_\ell(P,Q)^a$.

In [3], Barreto *et al.* proved that this pairing can be computed as $e_\ell(P,Q) = f_{\ell,P}(Q)^{\frac{p^{km}-1}{\ell}}$, where $f_{\ell,P}$ is evaluated on a point rather than on a divisor.

In this paper, we deal with the characteristic three case and consider E^b, a supersingular elliptic curve over $\mathbb{F}_{3^m}$: $E^b : y^2 = x^3 - x + b$, with $b \in \{-1,1\}$. According to [3], curves over fields of characteristic three often offer the best possible ratio between security level and space requirements.

Different ways for computing the Tate pairing can be found in [3, 10, 12, 21]. In [2], Barreto *et al.* introduced the η_T pairing which extended and improved the Duursma-Lee techniques [10]. To do it, they first need to consider the following distortion map $\psi : E^b(\mathbb{F}_{3^m}) \rightarrow E^b(\mathbb{F}_{3^{6m}})$ defined, for all $R \in E^b(\mathbb{F}_{3^m})$ by $\psi(R) = \psi(x_r, y_r) = (-x_r + \rho, y_r\sigma)$, where σ and ρ belong to $\mathbb{F}_{3^{6m}}$ and respectively satisfy $\sigma^2 = -1$ and $\rho^3 = \rho + b$ (that concept of distortion map was introduced in [31]). We define the modified Tate pairing $\hat{e}$ by $\hat{e}(P,Q) = e(P,\psi(Q))$ for all P, $Q \in E(\mathbb{F}_{3^m})[\ell]$.

Moreover, following [17], we construct $\mathbb{F}_{3^{6m}}$ as an extension of $\mathbb{F}_{3^m}$ using the basis $(1, \sigma, \rho, \sigma\rho, \rho^2, \sigma\rho^2)$, which is equivalent to considering the tower $\mathbb{F}_{3^m}$, $\mathbb{F}_{3^{2m}} \simeq \mathbb{F}_{3^m}[y]/(y^2+1)$ and $\mathbb{F}_{3^{6m}} \simeq \mathbb{F}_{3^{2m}}[z]/(z^3-z-b)$. Hence, the computations over $\mathbb{F}_{3^{6m}}$ are replaced by computations over $\mathbb{F}_{3^m}$.

The η_T pairing is defined by $\eta_T(P,Q) = f_{T,P}(\psi(Q))$, for some $T \in \mathbb{Z}$ and for all P and $Q \in E(\mathbb{F}_{3^m})[\ell]$. To get a well-defined, non-degenerate, bilinear pairing, a final exponentiation is required: namely $\eta_T(P,Q)^W$ in our case, where $W = (3^{3m}-1)(3^m+1)(3^m-b3^{\frac{m+1}{2}}+1)$. Moreover, the η_T pairing is related to the modified Tate pairing by $(\eta_T(P,Q)^W)^{3T^2} = \hat{e}(P,Q)^Z$, where $T = -b3^{\frac{m+1}{2}} - 1$ and $Z = -b3^{\frac{m+3}{2}}$. If v denotes $\eta_T(P,Q)^W$, the modified Tate pairing can be computed as follows

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

The algorithm given in [2] for computing the η_T pairing halves the number of iterations used in the approach by Duursma and Lee [10] but has the drawback of using inverse Frobenius maps. In [7] 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 final exponentiation $\eta_T(P,Q)^W$ [6]. In this paper, we extend this approach to the computation of the full η_T pairing (*i.e.* including the final exponentiation). In Section 2, we present a compact implementation of the η_T pairing over the field $\mathbb{F}_{3^{97}}$. Then, we show in Section 3 that our approach can be generalized to any characteristic p and degree-m irreducible polynomial $f(x)$ over $\mathbb{F}_p$. That generalization is an interesting issue since larger extension degrees could probably be considered in a close future for guaranteeing the security of pairing-based cryptosystems.

2 Calculation of the η_T Pairing in Characteristic Three

The bilinearity of $\eta_T(P,Q)^W$ ensures that:

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

Beuchat *et al.* proposed an algorithm for the calculation of $\eta_T(P,Q)^{3^{(m+1)/2}}$ in characteristic three without any inverse Frobenius map [7]. Therefore, inexpensive pre- and post-processing steps allow one to perform the original η_T pairing. Recall that, for $(x_p,y_p)\in E^b(\mathbb{F}_{3^m})$, $[3](x_p,y_p)=(x_p^9-b,-y_p^9)$ (see for instance [3]). Thus, the computation of $\left[3^{(m-1)/2}\right]P$ involves only $2m-2$ cubings and $(m-1)/2$ additions over $\mathbb{F}_{3^m}$. The 3^m-th root over $\mathbb{F}_{3^{6m}}$ is a straightforward operation requiring only seven additions (or subtractions) over $\mathbb{F}_{3^m}$ (see for instance [7]). The final exponentiation is carried out according to a novel algorithm introduced by Shirase, Takagi, and Okamoto in [26]. This scheme involves additions, cubings, multiplications, and a single inversion over $\mathbb{F}_{3^m}$.

In this section we will consider the field $\mathbb{F}_{3^{97}}=\mathbb{F}_3[x]/(x^{97}+x^{12}+2)$ and the curve $y^2=x^3-x+1$ over $\mathbb{F}_{3^{97}}$ (*i.e.* $b=1$; a straightforward adaptation makes it possible to address the $b=-1$ case). This choice of parameters allows us to easily compare our work against the many pairing accelerators for $m=97$ described in the open literature. Instead of embedding dedicated hardware to perform the inversion over $\mathbb{F}_{3^{97}}$ according to the Extended Euclidean Algorithm (EEA), Beuchat *et al.* [6] proposed an algorithm based on Fermat's little theorem and on Itoh and Tsujii's work [15] for $\mathbb{F}_{3^{97}}$. It involves 96 cubings and 9 multiplications. Algorithm 1 summarizes the computation of the full pairing. It is worth noticing that $\eta_T(P,Q)^W$ can be computed only by means of additions (or subtractions), multiplications, and cubings over $\mathbb{F}_{3^{97}}$. In the following, we describe the implementation of Algorithm 1 on a Virtex-II Pro 4 Field-Programmable Gate Array (FPGA) and compare our pairing accelerator against results published by other researchers.

Algorithm 1 Computation of $\eta_T(P,Q)^W$ for $b = 1$ [7].

Input: $P = (x_p, y_p)$ and $Q = (x_q, y_q) \in E(\mathbb{F}_{3^m})[l]$. 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: $\eta_T(P,Q)^{(3^{3m}-1)(3^m+1)(3^m+1-3^{(m+1)/2})}$.

for $i = 0$ to $\frac{m-1}{2} - 1$ **do**
  $x_p \leftarrow x_p^9 - 1$; $y_p \leftarrow -y_p^9$;
end for
$y_p \leftarrow -y_p$; $d \leftarrow 1$;
$r_0 \leftarrow x_p + x_q + d$;
$R_0 \leftarrow -y_p r_0 + y_q \sigma + y_p \rho$;
$R_1 \leftarrow -r_0^2 + y_p y_q \sigma - r_0 \rho - \rho^2$;
$R_0 \leftarrow (R_0 R_1)^3$;
for $i = 0$ to $\frac{m-1}{2} - 1$ **do**
  $y_p \leftarrow -y_p$; $x_q \leftarrow x_q^9$; $y_q \leftarrow y_q^9$; $d \leftarrow (d-1) \bmod 3$;
  $r_0 \leftarrow x_p + x_q + d$;
  $R_1 \leftarrow -r_0^2 + y_p y_q \sigma - r_0 \rho - \rho^2$;
  $R_0 \leftarrow (R_0 R_1)^3$;
end for
$R_0 \leftarrow R_0^{(3^{3m}-1)(3^m+1)(3^m+1-3^{(m+1)/2})}$;
$R_0 \leftarrow \sqrt[3^m]{R_0}$;
return R_0;

2.1 An Accelerator for the η_T Pairing Calculation

Beuchat *et al.* [6] designed a unified arithmetic operator able to perform addition, multiplication, and cubing over $\mathbb{F}_3[x]/(f(x))$, where $f(x) = x^{97} + x^{12} + 2$. The operator is based on the array multiplier architecture proposed by Shu, Kwon, and Gaj in [27] (see [5, 29] for an introduction to array multipliers). Since such multipliers process D coefficients of an operand at each clock cycle, they mainly consist of D Partial Product Generators (PPGs), a D-operand adder, and an accumulator. Figure 1 illustrates the architecture of this operator for $D = 3$; it is controlled by eleven bits labelled c_i. Let $a(x)$ and $b(x)$ belong to $\mathbb{F}_3[x]/(f(x))$. In order to compute $a(x) \times b(x)$, one has to load $a(x)$ in the shift register *R0*, and $b(x)$ in registers *R1* and *R2*. Multiplication is then carried out in $\lceil m/D \rceil = \lceil 97/3 \rceil = 33$ clock cycles. The first iteration computes $p(x) = a_{96} b(x)$ ($c_4 = c_6 = c_7 = c_8 = 1$, $c_{10} = 0$). Then, we update $p(x)$ as follows:

$$p(x) \leftarrow x^3 p(x) \bmod f(x) + a_{3i+2} x^2 b(x) \bmod f(x) + \\ a_{3i+1} x b(x) \bmod f(x) + a_{3i} b(x),$$

where i ranges from 31 downto 0. Addition is somewhat more complex and we will use the toy example proposed in [6] to illustrate how the operator works. Let us assume we have to compute $-a(x) + b(x)$. We respectively load $a(x)$ and $b(x)$ in registers *R2* and *R1* and define a control word stored in *R0* so that $\mathrm{d0}_{3i} = 2$, $\mathrm{d0}_{3i+1} = 1$, and $\mathrm{d0}_{3i+2} = 0$. We will thus compute $(2a(x) + b(x) + 0 \cdot a(x)) \bmod f(x) = (-a(x) + b(x)) \bmod f(x)$. Beuchat *et al.* [6] noticed that $a(x)^3 = \nu_0(x) + \nu_1(x) + \nu_2(x)$, where $\nu_0(x)$, $\nu_1(x)$, and $\nu_2(x)$ belong to $\mathbb{F}_{3^{97}}$ (see Appendix B). Thus, cubing requires the addition of three operands as well as some wiring to compute the $c_i(x)$'s. It suffices to load $a(x)$ in registers *R1* and *R2*. Depending on the control word stored in *R0*, the operator returns $a(x)^3$ or $-a(x)^3$. In order to efficiently implement successive cubings, a feedback mechanism allows one to load *R1* and *R2* with the result of a cubing (multiplexers controlled by c_0 and c_2 on Figure 1).

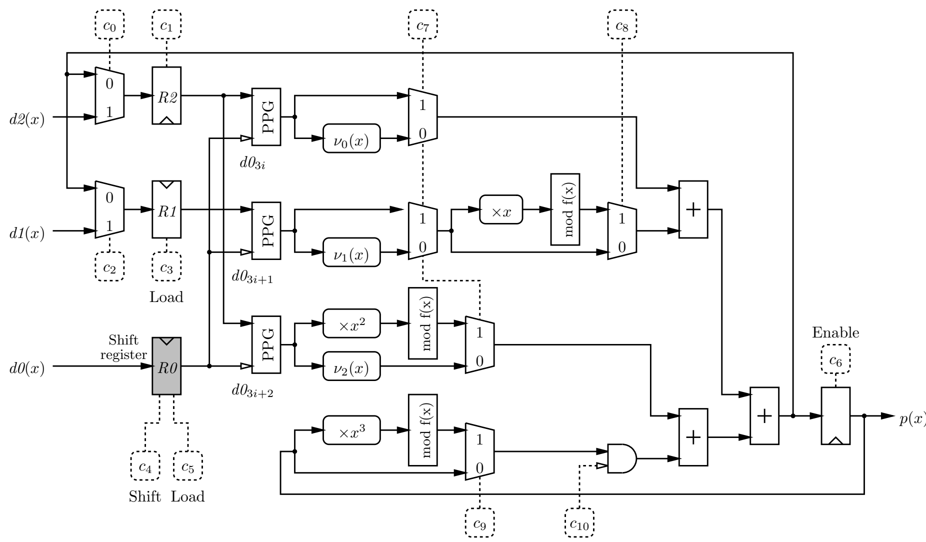

Fig. 1. Operator for addition, multiplication, and cubing over $\mathbb{F}_3[x]/(x^{97} + x^{12} + 2)$ introduced in [6]. Boxes with rounded corners involve only wiring.

Figure 2 describes the architecture of our η_T pairing coprocessor, which is mainly based on the hardware accelerator for the final exponentiation introduced in [6]. It consists of a single processing element (unified operator for addition, multiplication, and cubing), registers implemented by means of a dual-port RAM (six Virtex-II Pro SelectRAM+ blocks), and a control unit which consists of a Finite State Machine (FSM) and an instruction memory (ROM). The main difference with [6] lies in the control unit and the register file: in order to deal with the computation of the η_T pairing, our coprocessor needs a slightly more complex FSM as well as eight additional registers to store control words for additions and cubings of the pairing calculation. Each instruction consists of four fields: a control 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 counter which indicates how many times the instruction must be repeated. This approach makes it possible for instance to execute the consecutive steps appearing in the multiplication over $\mathbb{F}_{3^{97}}$ with a single instruction. Note that our implementation of the η_T pairing for $m = 97$ and $D = 3$ does not require the 2^6 values of the counter. It is therefore possible to encode the required values with fewer bits in order to reduce the width of the instructions.

Since the implementation of the final exponentiation on such an architecture has already been discussed in [6], we will focus here on the computation of $\eta_T\left(\left[3^{(m-1)/2)}\right] P, Q\right)^{3^{(m+1)/2}}$. It is now well known that the tower field representation and Karatsuba-Ofman's algorithm allows one to replace a multiplication over $\mathbb{F}_{3^{6m}}$ by 18 multiplications and 58 additions over $\mathbb{F}_{3^m}$ (see for instance [6, 17]). Further optimizations are however possible in the case of the η_T pairing calculation. Multiplying $R_0 = -y_p r_0 + y_q \sigma + y_p \rho$ by $R_1 = -r_0^2 + y_p y_q \sigma - r_0 \rho - \rho^2$ involves for instance only 8 multiplications and 9 additions over $\mathbb{F}_{3^m}$ (see Algorithm 4 in Appendix A for details). As pointed out by Bertoni *et al* [4], the multiplication over $\mathbb{F}_{3^{6m}}$ occurring in the main loop of the pairing calculation (Algorithm 1) requires 13 multiplications over $\mathbb{F}_{3^m}$.

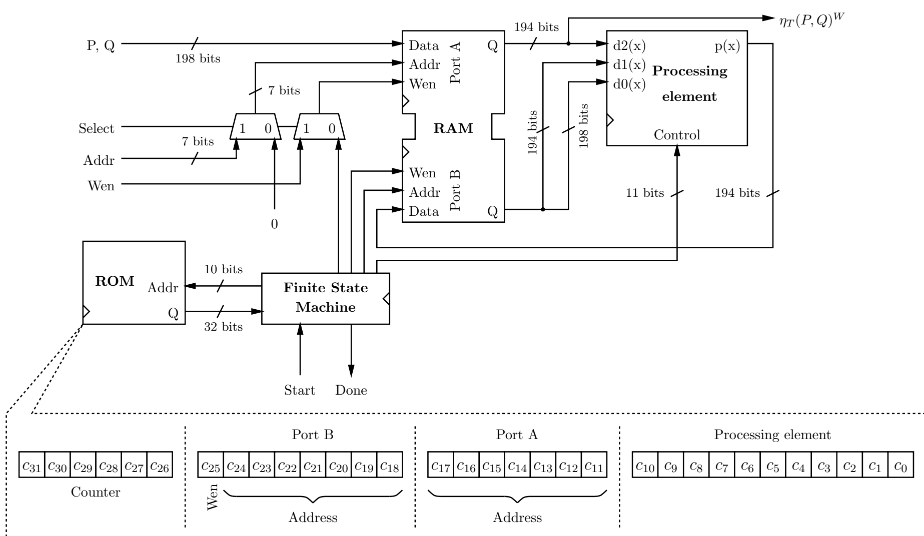

Fig. 2. Architecture of the η_T pairing accelerator.

The implementation of Algorithm 1 on this architecture takes 895 instructions which are executed in 32618 clock cycles (229 instructions for the computation of $\eta_T(3^{(m-1)/2}P, Q)^{3^{(m+1)/2}}$; 666 instructions for the final exponentiation and the 3^m-th root over $\mathbb{F}_{3^{6m}}$). The inversion over $\mathbb{F}_{3^{97}}$ is performed by means of 96 cubings and 9 multiplications over $\mathbb{F}_{3^{97}}$ [6]. Eighteen control words, stored in the dual-port RAM, manage all additions and cubings involved in the computation of the full pairing. Table 1 summarizes the operations over $\mathbb{F}_{3^m}$ needed in the computation of $\eta_T(P,Q)^W$. The last column indicates the number of clock cycles during which only load/store operations are performed. When $m = 97$, our coprocessor is for instance idle during 1704 clock cycles (*i.e.* 5.2% of the total computation time).

Table 1. Operations over $\mathbb{F}_{3^m}$ involved in the computation of $\eta_T(P,Q)^W$.

	Additions	**Cubings**	**Multiplications**	**Inversion**	**Idle**
Point tripling	$\frac{m-1}{2}$	$2m-2$	–	–	5
Pairing	$25m-6$	$5m+1$	$15 \cdot \frac{m-1}{2} + 8$	–	$14m-4$
Final exp.	477	$3m+3$	78	1	344
$\sqrt[3^m]{}$	7	–	–	–	1
Total	$51 \cdot \frac{m-1}{2} + 503$	$10m+2$	$15 \cdot \frac{m-1}{2} + 86$	1	$14m+346$

2.2 Results and Comparisons

The architecture described by Figure 2 was captured in the VHDL language and prototyped on a Xilinx Virtex-II Pro 4 device (XC2VP4-6FF672). Both synthesis and place-and-route steps were performed with ISE WebPACK 8.2.03i. Our processor requires 1888 slices and 6 memory blocks. Since a Virtex-II Pro 4 does not have enough I/Os for parallel communications with a computer, the number of slices reported here includes shift registers to receive/send data in a serial fashion. The clock frequency of 147 MHz allows one to compute $\eta_T(P,Q)^W$ according to Algorithm 1 in 222 μs. Table 2 provides the reader with a comparison against architectures proposed by other researchers for $p = 3$ and $m = 97$.

Grabher and Page designed a coprocessor dealing with arithmetic over $\mathbb{F}_{3^m}$, which is controlled by a general purpose processor [13]. 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 [1]. This architecture occupies 4481 slices and allows one to perform the Duursma-Lee algorithm and its final exponentiation in 432.3 μs. The main advantage is maybe that the control can be compiled using a re-targeted GCC tool-chain 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 2.3.

Another implementation of the Duursma-Lee algorithm was proposed by Kerins *et al.* in [17]. 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 [4] for the pairing calculation. Therefore, the hardware architecture consists of 18 multipliers and 6 cubing circuits over $\mathbb{F}_{3^{97}}$, along with, quoting [17], "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 [27]. The approach proposed in this paper reduces the area and the computation time by 29 and 3.8 respectively.

Beuchat *et al.* described a fast architecture for the computation of the η_T pairing [7]. 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 [6]. 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 EP2C35F672C6 FPGA with Quartus II 6.0 Web Edition. Our design occupies 2846 LEs and the maximal clock frequency of 125 MHz allows one to compute a pairing in 261 μs. The architecture proposed in this paper is therefore 8 times slower, but 6.3 times smaller. Note that the critical path is located in the control unit: the glue logic generated by Quartus II to interconnect M4K memory blocks storing the instructions seems to slow the whole design down. It is possible to further pipeline the control unit and to compute the full pairing in 222 μs.

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 [25]. An architecture embedding three multipliers processing $D = 8$ coefficients at each clock cycle computes for instance a full pairing in 178 μs. Though 1.25 times 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.

3 Arithmetic over $\mathbb{F}_{p^m}$

The unified operator for arithmetic over $\mathbb{F}_3[x]/(x^{97} + x^{12} + 2)$ introduced in [6] allows us to present in this paper the smallest FPGA-based pairing accelerator in the open literature. However, 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. We wrote a C++ program which automatically generates a synthesizable VHDL description of a unified operator according to the characteristic and the irreducible polynomial $f(x)$.

3.1 Addition, Multiplication, and Frobenius Map over $\mathbb{F}_{p^m}$

The architecture of the operators generated by our program is directly inspired from the unified operator given in Figure 1 and can be adapted to any prime characteristic p and any irreducible polynomial $f(x)$ of degree m.

Table 2. Comparisons against FPGA-based accelerators over $\mathbb{F}_{3^{97}}$. The parameter D refers to the number of coefficients processed at each clock cycle by a multiplier.

	Grabher and Page [13]	**Kerins *et al.* [17]**	**Beuchat *et al.* [6, 7]**
Algorithm	Duursma-Lee	Duursma-Lee	η_T pairing
FPGA	Virtex-II Pro 4	Virtex-II Pro 125	Cyclone II EP2C35
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	850 μs	33 μs

	Ronan *et al.* [25]		**Proposed architecture**
Algorithm	η_T pairing	η_T pairing	η_T pairing
FPGA	Virtex-II Pro 100	Virtex-II Pro 100	Virtex-II Pro 4
Multiplier(s)	3 ($D = 8$)	2 ($D = 8$)	1 ($D = 3$)
Area	10000 slices	7491 slices	1888 slices
Clock cycles	15113	17190	32618
Clock frequency	70.4 MHz	70.4 MHz	147 MHz
Calculation time	178 μs	203 μs	222 μs

Addition over $\mathbb{F}_p[x]/(f(x))$ is performed in the same way as in the operator over $\mathbb{F}_{3^{97}}$ presented in [6]: the digits of the two operands are all added in parallel, thus requiring m additions over $\mathbb{F}_p$. In the current version of the generator, those additions over $\mathbb{F}_p$ are implemented as simple look-up tables addressed by the bits of the two operands, particularly suited for small values of p (typically $p = 2$ to 7). For higher characteristics, it will be necessary to resort to more complex methods for modular addition [24].

Also as in the original operator, multiplication over $\mathbb{F}_p[x]/(f(x))$ relies on a parallel-serial algorithm, with D digits of the multiplier being processed at each iteration. The generation of the partial products, which consists in multiplying all the digits of the multiplicand with each digit of the multiplier, requires m multiplications over $\mathbb{F}_p$ in parallel for each of the D partial products. Here also, the multiplications over $\mathbb{F}_p$ are directly tabulated, as this is the best solution for small characteristics. Once the D partial products are computed, the $D - 1$ most significant ones along with the accumulator are then multiplied by x^k (where k ranges from 1 to D) and reduced modulo $f(x)$. After the modular reductions, the D partial products and the accumulator are added thanks to a binary tree of adders over $\mathbb{F}_{p^m}$. Consequently, in order to optimize the critical path of this multioperand adder, one should choose a parameter D of the form $2^n - 1$ (typically $D = 3$, 7, 15 or 31).

Concerning the Frobenius map, which consists in raising the operand $a(x)$ to the pth power, our generator first computes the normal form of $a(x)^p \bmod f(x)$, for a generic polynomial $a(x)$, by reducing the following expression modulo $f(x)$:

$$a(x)^p \bmod f(x) = \sum_{i=0}^{m-1} a_i^p x^{ip} \bmod f(x) = \sum_{i=0}^{m-1} a_i x^{ip} \bmod f(x).$$

This general expression of the Frobenius map can then be seen as a sum of elements of $\mathbb{F}_{p^m}$. The coefficients of those polynomials can be directly matched to the coefficients of the operand, possibly multiplied by a constant. As presented in [6], it is possible to reuse the partial product generation hardware of the multiplication in order to compute those polynomials, only some extra wiring being required for the permutation of the coefficients. The sum of all the polynomials can then be computed by the final multi-operand adder.

In order to decrease the number of partial products necessary to compute the Frobenius map, a simple decomposition technique can be applied to share the maximum amount of hardware between these partial products. In case this is still not enough, a second technique can further pack the partial products, at the expense of some additions over $\mathbb{F}_p$. The intuition behind these two techniques is given in a simple example in Appendix B.

3.2 Inverse Frobenius Map

Although the algorithm we present here for the η_T pairing over $\mathbb{F}_{3^m}$ does not require to compute any inverse Frobenius map (*i.e.* $\sqrt[p]{a(x)}$), some other algorithms still rely on this function. To also support those algorithms, the generic unified operator proposed in this paper is available in two flavors: namely either only addition, multiplication and Frobenius map as presented in the previous section, or a four-in-one operator with extra hardware for the inverse Frobenius map. This function is computed exactly in the same way as the Frobenius map: first, the normal form of $\sqrt[p]{a(x)} \bmod f(x)$ is obtained by solving the m-dimensional linear system given by the equation $\left(\sqrt[p]{a(x)}\right)^p \bmod f(x) = a(x)$. The result is then expressed as a sum of polynomials, each one being a permutation of the coefficients of the operand $a(x)$ multiplied by a constant. Note that the reduction techniques presented for the Frobenius map also apply in the case of the inverse map.

3.3 Inversion over $\mathbb{F}_{p^m}$

Recall that, in the case of $\mathbb{F}_{3^{97}}$ [6], our pairing accelerator performs the inversion required for the final exponentiation according to Fermat's little theorem and Itoh and Tsujii's work [15] by means of 96 cubings and 9 multiplications. We propose here a generalization of this algorithm to any characteristic and extension degree.

General algorithm. The inversion scheme summarized in Algorithm 2 is often applied for inversion in optimal extension fields [8]. Starting with an element a of $\mathbb{F}_{p^m}$, we first raise it to the power of the base-p repunit $(p^{m-1}-1)/(p-1)$ to obtain r. This particular powering can be achieved using only $m-2$ Frobenius maps and a few multiplications over $\mathbb{F}_{p^m}$ as detailed below.

By applying another Frobenius map to r and then multiplying the result by a, we successively obtain

$$s = a^{(p^m-p)/(p-1)}, \text{ and}$$
$$t = a^{(p^m-1)/(p-1)}.$$

Since $t \neq 0$ and $t^{p-1} = a^{p^m-1} = 1$, $t \in \mathbb{F}_p$ and we define u as $t^{p-2} = t^{-1}$. Therefore, the final product gives us the result $s \cdot u = s \cdot (s \cdot a)^{-1} = a^{-1}$.

Algorithm 2 Inversion over $\mathbb{F}_{p^m}$.

Input: A prime number p, a positive integer m, and $a \in \mathbb{F}_{p^m}$, $a \neq 0$.
Output: $a^{-1} \in \mathbb{F}_{p^m}$.

1: $r \leftarrow a^{(p^{m-1}-1)/(p-1)}$;
2: $s \leftarrow r^p$;
3: $t \leftarrow s \cdot a$;
4: $u \leftarrow t^{p-2}$;
5: **return** $s \cdot u$;

Several cases need to be considered, depending on the characteristic p:

- When $p = 2$, we do not have to compute t and u, as $s = a^{(p^m-p)/(p-1)} = a^{2^m-2} = a^{-1}$. Thus, the inversion only requires to compute r and one extra Frobenius map, the operator directly returning s.
- When $p = 3$, we have $u = t^{p-2} = t$. Compared to the case $p = 2$, this only requires two additional multiplications over $\mathbb{F}_{3^m}$ for the products $t = s \cdot a$ and $a^{-1} = s \cdot t$.
- In the general case $p > 3$, we also have to explicitly compute t^{-1} as t^{p-2} by means of $\lfloor \log_2(p-2) \rfloor + \mathrm{wt}(p-2) - 1$ successive multiplications (where $\mathrm{wt}(k)$ is the Hamming weight of the binary representation of k).

However, in the case $p \geq 3$, remarking that $t, u \in \mathbb{F}_p$, we propose several modifications of the shift register of our unified operator (register *R0* in Figure 1) in order to simplify the computation of the products involving t and u.

The first modification, referred to as (A) in the following, consists in adding an extra control bit and a multiplexer to select the value of the coefficient $d0_{3i}$ of the shift register between its normal value (the third-to-most significant coefficient of the multiplier) and the order-0 coefficient of the multiplier. Indeed, as $u \in \mathbb{F}_p$, all its coefficients u_i are zero for all $i \neq 0$. Therefore, we only need u_0 to compute the final multiplication $s \cdot u = s \cdot u_0$. As our multiplier operates in a most-significant-coefficient-first fashion, instead of performing the full multiplication over $\mathbb{F}_{p^m}$, this multiplexer allows us to bypass the whole shift register mechanism and compute the product $s \cdot u$ in a single iteration of the multiplier. This modification also allows simpler computation of t^{p-2} in the case $p > 3$.

Also in this case $p > 3$, we can modify further the shift register in order to include the computation of $t^{p-2} = t^{-1}$ at the level of the multiplexer introduced by modification (A). In the following, this modification will be referred to as (B). As $u = t^{-1} = t_0^{-1}$, we can tabulate the inversion over $\mathbb{F}_p$, and, loading t in the shift register *R0*, select the coefficient $d0_{3i} = t_0^{-1}$ thanks to the multiplexer. Loading s in the parallel register *R2*, we can then directly perform the final product $s \cdot t^{-1} = a^{-1}$.

Addition chains to compute $a^{(p^{m-1}-1)/(p-1)}$. As already shown in [32] and [23], additions chains can prove to be perfectly suited to raise elements of $\mathbb{F}_{p^m}$ to particular powers, such as the radix-p repunit $(p^{m-1}-1)/(p-1)$ required by our inversion algorithm.

An addition chain S of length l is a sequence of l pairs of integers $S = ((j_1, k_1), \ldots, (j_l, k_l))$ such that $1 \le j_i \le k_i < i$ for all $1 \le i \le l$. We can then construct another sequence $(n_0, \ldots, n_l)$ satisfying

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

S is said to *compute* n_l, the last element of the sequence. For more details, see for instance [19].

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

$$a^{(p^{n+n'}-1)/(p-1)} = a^{(p^n-1)/(p-1)} \cdot \left(a^{(p^{n'}-1)/(p-1)}\right)^{p^n}.$$

Consequently, given an addition chain S of length l for $m-1$, we can compute the required $a^{(p^{m-1}-1)/(p-1)}$ as shown in Algorithm 3. This algorithm simply ensures that, for each iteration i, we have $z_i = a^{(p^{n_i}-1)/(p-1)}$, where $(n_0, ..., n_l)$ is the integer sequence associated with the addition chain S, verifying $n_l = m-1$.

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

Input: A prime number p, a positive integer m, $a \in \mathbb{F}_{p^m}$, and an addition chain $S = ((j_1, k_1), \ldots, (j_l, k_l))$ for $m - 1$.
Output: $a^{(p^{m-1}-1)/(p-1)} \in \mathbb{F}_{p^m}$.

```
1: z_0 ← a;
2: for i = 1 to l do
3:     z_i ← z_{j_i} · z_{k_i}^{p^{j_i}};
4: end for
5: return z_l;
```

Each iteration of the loop requires j_i Frobenius maps and one multiplication over $\mathbb{F}_{p^m}$, which gives a total cost of at least $m - 2$ Frobenius maps and l

multiplications. If S is a Brauer-type addition chain (*i.e.* $k_i = i - 1$ for all $1 \leq i \leq l$), the number of Frobenius maps is exactly $m-2$ [19]. With the intent of minimizing the number of operations, we have adapted some efficient algorithms from the literature [30] to find the shortest Brauer-type addition chain for any value of $m-1$. It is to be noted that Brauer-type chains are proved to be optimal for $m - 1$ up to and including 12508 [19], which is an acceptable limitation of our method for the time being.

Cost analysis. The overall cost of our inversion scheme is summarized in Table 3, according to the characteristic p and the possible modification of the unified operator. In this table, l represents the length of the shortest Brauer-type addition chain for $m - 1$, and $c(k)$ denotes the quantity $\lfloor \log_2(k) \rfloor + \mathrm{wt}(k) - 1$, the number of multiplications required to compute t^k.

Table 3. Overall cost of the inversion algorithm.

p	**Mod.**	**Mult. over $\mathbb{F}_{p^m}$** ($\lfloor m/D \rfloor$ cycles)	**Mult. over $\mathbb{F}_p$** (1 cycle)	**Frobenius maps** (1 cycle)
$p = 2$	–	l	0	$m - 1$
$p = 3$	–	$l + 2$	0	$m - 1$
	(A)	$l + 1$	1	$m - 1$
$p > 3$	–	$l + c(p - 2) + 2$	0	$m - 1$
	(A)	$l + 1$	$c(p - 2) + 1$	$m - 1$
	(B)	$l + 1$	1	$m - 1$

Table 4 provides the reader with a comparison between Algorithm 2 and the EEA in characteristic three. We assume that the accelerator embeds a single unified operator and carries out the pairing calculation according to Algorithm 1. Recall that the EEA performs an inversion over $\mathbb{F}_{3^m}$ in $2m$ clock cycles [18]. Then, Table 1 and the previous cost analysis allow us to find out the number of clock cycles and to give examples for $D = 3$ and 7. Our results indicate that supplementing our coprocessor with dedicated hardware for the EEA would only improve performance by less than 1%. Furthermore, an EEA-based inversion over $\mathbb{F}_{3^{97}}$ occupies 2210 slices on a Virtex-II Pro FPGA [18] and would more than double the area of the accelerator.

3.4 Results

Our VHDL code generator as well as the general formulas from Table 4 allowed us to estimate the cost of the full η_T pairing computation for several extension fields. Table 5 summarizes these estimations. Note that the reported figures do not take the control unit into account. However, this should not impact on the critical path.

Table 4. Relationship between the choice of an inversion algorithm and the calculation time of a full pairing according to Algorithm 1. The cost of the multiplication over $\mathbb{F}_3$ is neglected: only full $\mathbb{F}_{3^m}$ multiplications are considered.

(a) Arithmetic over $\mathbb{F}_{3^{97}}$ ($l = 7$).

Inversion		**Clock cycles for the full pairing**		
Algorithm	**Cost**	**General formula**	**D = 3**	**D = 7**
Algo. 2	96 cubings, 9 mult.	$5723 + 815 \cdot \lceil 97/D \rceil$	32618	17133
Algo. 2, mod. (A)	96 cubings, 8 mult.	$5723 + 814 \cdot \lceil 97/D \rceil$	32585	17119
EEA	$2 \cdot m = 194$ clock cycles	$5821 + 806 \cdot \lceil 97/D \rceil$	32419	17105

(b) Arithmetic over $\mathbb{F}_{3^{193}}$ ($l = 8$).

Inversion		**Clock cycles for the full pairing**		
Algorithm	**Cost**	**General formula**	**D = 3**	**D = 7**
Algo. 2	192 cubings, 10 mult.	$10571 + 1536 \cdot \lceil 193/D \rceil$	110411	53579
Algo. 2, mod. (A)	192 cubings, 9 mult.	$10571 + 1535 \cdot \lceil 193/D \rceil$	110346	53551
EEA	$2 \cdot m = 386$ clock cycles	$10765 + 1526 \cdot \lceil 193/D \rceil$	109955	53493

Table 5. Estimated area, frequency, and full pairing computation time for various extension fields (such as considered in [1,6] and values for the parameter D (Virtex-II Pro family).

Polynomial	$D = 3$	$D = 7$
$x^{97} + x^{12} + 2$	1402 slices – 147 MHz – 222 μs	2189 slices – 117 MHz – 146 μs
$x^{97} + x^{16} + 2$	1392 slices – 151 MHz – 216 μs	2246 slices – 116 MHz – 148 μs
$x^{193} + x^{64} + 2$	2811 slices – 126 MHz – 877 μs	4450 slices – 108 MHz – 495 μs

4 Conclusion

We proposed a compact implementation of the 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, without impacting too severely on the performances. We also showed that our approach can be generalized to any characteristic p and degree-m irreducible polynomial $f(x)$ over $\mathbb{F}_p$. Moreover, our VHDL code generator allows one to rapidly explore the trade-off between computation time and circuit resource usage for a large set of architectural parameters (*e.g.* p, m, $f(x)$).

However, even though we now have automatic tools to generate unified operators, the main difficulty still lies in the scheduling of all the instructions required for the η_T pairing calculation. The next step will therefore be to develop an *ad-hoc* compiler for architectures based on such unified operators.

Acknowledgments

The authors would like to thank Francisco Rodríguez-Henríquez and the anonymous referees for their valuable comments. This work was supported by the New Energy and Industrial Technology Development Organization (NEDO), Japan.

The authors would also like to express their deepest gratitude to the Carthusian Monks of the Grande Chartreuse in the French Alps for their succulent herbal products which fueled our efforts in writing this article.

References

1. P. S. L. M. Barreto. A note on efficient computation of cube roots in characteristic 3. Cryptology ePrint Archive, Report 2004/305, 2004.
2. 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.
3. P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, number 2442 in Lecture Notes in Computer Science, pages 354–368. Springer, 2002.
4. 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.
5. G. Bertoni, J. Guajardo, S. Kumar, G. Orlando, C. Paar, and T. Wollinger. Efficient GF(p^m) arithmetic architectures for cryptographic applications. In M. Joye, editor, *Topics in Cryptology – CT-RSA 2003*, number 2612 in Lecture Notes in Computer Science, pages 158–175. Springer, 2004.
6. 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. Cryptology ePrint Archive, Report 2007/045, 2007.
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.
8. H. Cohen and G. Frey, editors. *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Chapman & Hall/CRC, 2005.
9. R. Dutta, R. Barua, and P. Sarkar. Pairing-based cryptographic protocols: A survey. Cryptology ePrint Archive, Report 2004/64, 2004.
10. I. Duursma and H. S. Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In C. S. Laih, editor, *Advances in Cryptology – ASIACRYPT 2003*, number 2894 in Lecture Notes in Computer Science, pages 111–123. Springer, 2003.
11. G. Frey and H.-G. Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62(206):865–874, April 1994.
12. S. D. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In C. Fieker and D.R. Kohel, editors, *Algorithmic Number Theory – ANTS V*, number 2369 in Lecture Notes in Computer Science, pages 324–337. Springer, 2002.

13. P. Grabher and D. Page. Hardware acceleration of the Tate Pairing in characteristic three. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, number 3659 in Lecture Notes in Computer Science, pages 398–411. Springer, 2005.
14. R. Granger, D. Page, and N. P. Smart. High security pairing-based cryptography. Cryptology ePrint Archive, Report 2006/059, 2006.
15. T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in $\mathrm{GF}(2^m)$ using normal bases. *Information and Computation*, 78:171–177, 1988.
16. A. Joux. A One Round Protocol for Tripartite Diffie-Hellman. In W. Bosma, editor, *Algorithmic Number Theory Symposium, ANTS-IV*, number 1838 in Lecture Notes in Computer Science, pages 385–394. Springer, 2000.
17. T. Kerins, W. P. Marnane, E. M. Popovici, and P.S.L.M. Barreto. Efficient hardware for the Tate Pairing calculation in characteristic three. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, number 3659 in Lecture Notes in Computer Science, pages 412–426. Springer, 2005.
18. T. Kerins, E. Popovici, and W. Marnane. Algorithms and architectures for use in FPGA implementations of identity based encryption schemes. In J. Becker, M. Platzner, and S. Vernalde, editors, *Field-Programmable Logic and Applications*, number 3203 in Lecture Notes in Computer Science, pages 74–83. Springer, 2004.
19. D. E. Knuth. *The Art of Computer Programming*, volume 2, *Seminumerical Algorithms*. Addision-Wesley, 3rd edition, 1998.
20. N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. In N. P. Smart, editor, *Cryptography and Coding*, number 3796 in Lecture Notes in Computer Science, pages 13–36. Springer, 2005.
21. S. Kwon. Efficient Tate pairing computation for supersingular elliptic curves over binary fields. Cryptology ePrint Archive, Report 2004/303, 2004.
22. A. Menezes, T. Okamoto, and S. A. Vanstone. Reducing elliptic curves logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, September 1993.
23. F. Rodríguez-Henríquez, G. Morales-Luna, N. A. Saqib, and N. Cruz-Cortés. A parallel version of the Itoh-Tsujii multiplicative inversion algorithm. In P. C. Diniz, E. Marques, K. Bertels, M. M. Fernandes, and J. M. P. Cardoso, editors, *Reconfigurable Computing: Architectures, Tools and Applications – Proceedings of ARC 2007*, number 4419 in Lecture Notes in Computer Science, pages 226–237. Springer, 2007.
24. F. Rodríguez-Henríquez, N. A. Saqib, A. D. Pérez, and Ç. K. Koç. *Cryptographic Algorithms on Reconfigurable Hardware*. Springer, 2006.
25. R. Ronan, C. Ó hÉigeartaigh, C. Murphy, T. Kerins, and P. S. L. M. Barreto. Hardware implementation of the η_T pairing in characteristic 3. Cryptology ePrint Archive, Report 2006/371, 2006.
26. M. Shirase, T. Takagi, and E. Okamoto. Some efficient algorithms for the final exponentiation of η_T pairing. Cryptology ePrint Archive, Report 2006/431, 2006.
27. C. Shu, S. Kwon, and K. Gaj. FPGA accelerated Tate pairing based cryptosystem over binary fields. Cryptology ePrint Archive, Report 2006/179, 2006.
28. J. H. Silverman. *The Arithmetic of Elliptic Curves*. Number 106 in Graduate Texts in Mathematics. Springer-Verlag, 1986.
29. 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.
30. E. G. Thurber. Efficient generation of minimal length addition chains. *Journal on Computing*, 28(4):1247–1263, August 1999.

31. E. R. Verheul. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. *Journal of Cryptology*, 17(4):277–296, 2004.
32. J. von zur Gathen and M. Nöcker. Computing special powers in finite fields. *Mathematics of Computation*, 73(247):1499–1523, 2003.

A Computation of the η_T Pairing

We consider here the first multiplication over $\mathbb{F}_{3^{6m}}$ of the η_T pairing calculation (Algorithm 1). Let $A = (a_0, a_1, a_2, a_3, a_4, a_5) \in \mathbb{F}_{3^{6m}}$. We have to compute $a_0 + a_1\sigma + a_2\rho + a_3\sigma\rho + a_4\rho^2 + a_5\sigma\rho^2 = (-y_p r_0 + y_q\sigma + y_p\rho)(-r_0^2 + y_p y_q \sigma - r_0\rho - \rho^2)$. We assume here that $b = 1$. Since $\sigma^2 = 1$ and $\rho^3 = \rho + 1$, we obtain:

$$\begin{aligned} a_0 &= y_p r_0^3 - y_p y_q^2, & a_2 &= -y_p, & a_4 &= 0, \\ a_1 &= -y_p^2 y_q r_0 - y_q r_0^2, & a_3 &= -y_q r_0 + y_p^2 y_q, & a_5 &= -y_q. \end{aligned}$$

This multiplication over $\mathbb{F}_{3^{6m}}$ is carried out according to Algorithm 4 which requires 8 multiplications and 9 additions over $\mathbb{F}_{3^m}$. Note that the number of additions may depend on the architecture of the coprocessor.

Algorithm 4 First multiplication of the η_T pairing calculation.

Require: $R_0 = -y_p r_0 + y_q\sigma + y_p\rho$ and $R_1 = -r_0^2 + y_p y_q\sigma - r_0\rho - \rho^2 \in \mathbb{F}_{3^{6m}}$.
Ensure: $A = R_0 R_1 \in \mathbb{F}_{3^{6m}}$.
$e_0 \leftarrow r_0 r_0$; $e_1 \leftarrow y_q r_0$; $e_2 \leftarrow y_p r_0$;
$e_3 \leftarrow e_0 e_2$; $(e_3 = y_p r_0^3)$
$e_4 \leftarrow y_p y_q$;
$e_5 \leftarrow e_4 y_q$; $(e_5 = y_p y_q^2)$
$e_6 \leftarrow e_4 y_p$; $(e_5 = y_p^2 y_q)$
$e_7 \leftarrow -e_2 + y_q$; $(e_7 = -y_p r_0 + y_q)$
$e_8 \leftarrow -e_0 + e_4$; $(e_8 = -r_0^2 + y_p y_q)$
$e_9 \leftarrow e_7 e_8$; $(e_9 = (-y_p r_0 + y_q)(-r_0^2 + y_p y_q))$
$a_1 \leftarrow e_9 - e_3 - e_5$; $a_0 \leftarrow e_3 - e_5 - y_p$;
$a_3 \leftarrow -e_1 + e_6$; $a_2 \leftarrow -y_p$; $a_4 \leftarrow 0$; $a_5 \leftarrow -y_q$;

B Techniques for Reducing Partial Products in the Frobenius Map

For our unified operators to be able to compute Frobenius maps, we implement this function as a sum of elements of $\mathbb{F}_{p^m}$. With $p = 3$ and $f(x) = x^{97} + x^{12} + 2$, we obtain $a(x)^p \bmod f(x) = \mu_0(x) + \mu_1(x) + \mu_2(x) + 2 \cdot \mu_3(x)$, with

$$\begin{cases} \mu_0(x) = a_0 + a_{65}x + a_{33}x^2 + \ldots + a_{96}x^{94} + a_{64}x^{95} + a_{32}x^{96}, \\ \mu_1(x) = a_{89} + 0 + 0 + \ldots + a_{88}x^{94} + 0 + 0, \\ \mu_2(x) = a_{93} + 0 + 0 + \ldots + a_{92}x^{94} + 0 + 0, \\ \mu_3(x) = 0 + a_{61}x + 0 + \ldots + 0 + a_{60}x^{95} + 0 \end{cases}$$

Hence, the Frobenius map in this extension field can be mapped as the sum of four polynomials $\mu_0(x)$ to $\mu_3(x)$, the first three with the weight 1 and the last one with the weight 2. Directly implementing our unified operator from this expression therefore would require at least $D = 4$. However, as noticed by Beuchat *et al.* [6], for each degree i for which the coefficient for x^i in $\mu_3(x)$ is not zero, the corresponding coefficients in $\mu_1(x)$ and $\mu_2(x)$ are always null. Rewriting 2 as $1+1$, we can then distribute $2 \cdot \mu_3(x)$ and merge it to $\mu_1(x)$ and $\mu_2(x)$ to obtain the following expression, requiring only $D = 3$ partial product generators: $a(x)^p \bmod f(x) = \nu_0(x) + \nu_1(x) + \nu_2(x)$, with

$$\begin{cases} \nu_0(x) = a_0 + a_{65}x + a_{33}x^2 + \ldots + a_{96}x^{94} + a_{64}x^{95} + a_{32}x^{96}, \\ \nu_1(x) = a_{89} + a_{61}x + 0 + \ldots + a_{88}x^{94} + a_{60}x^{95} + 0, \\ \nu_2(x) = a_{93} + a_{61}x + 0 + \ldots + a_{92}x^{94} + a_{60}x^{95} + 0. \end{cases}$$

This technique was fully automatized and implemented in our generator, which can minimize the number of partial products necessary to compute Frobenius maps in any extension field $\mathbb{F}_p[x]/(f(x))$. However, in some cases where it is not possible to decrease the number of required partial products to an acceptable value, the generator can also insert adders over $\mathbb{F}_p$ in order to share each partial product between several polynomials with the same weight. For instance, in our example, we can rewrite the expression of $a(x)^p \bmod f(x)$ with only $D = 2$ partial products as: $a(x)^p \bmod f(x) = \pi_0(x) + \pi_1(x)$, with

$$\begin{cases} \pi_0(x) = \nu_0(x), \\ \pi_1(x) = \nu_1(x) + \nu_2(x). \end{cases}$$

Similar techniques can also be applied to the inverse Frobenius map $\sqrt[p]{a(x)}$.