Hardware Implementation of the η_T Pairing in Characteristic 3

Robert Ronan[1], Colm Ó hÉigeartaigh[2], Colin Murphy[1], Tim Kerins[1] and Paulo S. L. M. Baretto[3]

[1] Department of Electrical & Electronic Engineering, University College Cork, College Road, Cork, Ireland.
`{robertr,cmurphy,timk}@rennes.ucc.ie`
[2] School of Computing, Dublin City University, Ballymun, Dublin 9, Ireland.
`coheigeartaigh@computing.dcu.ie`
[3] Escola Politécnica, Universidade de São Paulo, Brazil.
`pbarreto@larc.usp.br`

Abstract. Recently, there have been many proposals for secure and novel cryptographic protocols that are built on bilinear pairings. The η_T pairing is one such pairing and is closely related to the Tate pairing. In this paper we consider the efficient hardware implementation of this pairing in characteristic 3. All characteristic 3 operations required to compute the pairing are outlined in detail. An efficient, flexible and reconfigurable processor for the η_T pairing in characteristic 3 is presented and discussed. The processor can easily be tailored for a low area implementation, for a high throughput implementation, or for a balance between the two. Results are provided for various configurations of the processor when implemented over the field $\mathbb{F}_{3^{97}}$ on an FPGA. As far as we are aware, the processor returns the first characteristic 3 η_T pairing in hardware that includes a final exponentiation to a unique value.

Keywords $-$ η_T pairing, characteristic 3, elliptic curve, reconfigurable processor, FPGA

1 Introduction

Since the introduction of the Weil and Tate pairings for constructive cryptographic applications, there has been a flurry of proposals for novel cryptographic protocols based on pairings, including identity-based encryption (IBE) schemes, key exchange schemes and short signature schemes [7].

Bilinear pairings are used to map the Discrete Logarithm Problem (DLP) from the divisor class group of an elliptic or genus 2 hyperelliptic curve, defined over $\mathbb{F}_q$, to the multiplicative subgroup $\mathbb{F}^*_{q^k}$ of some extension of the base field. The value k is known as the *embedding degree* or *security multiplier*.

The viability of practical applications of these schemes relies on the efficient implementation of bilinear pairings. This has led to many suggestions for algorithmic optimisations. The Tate pairing was originally computed using Miller's Algorithm [14].

In 2002, significant improvements to this method were independently suggested in [3] and [9]. After this, the Tate pairing implementation was further optimised in [8] for a small class of curves, resulting in the Duursma-Lee algorithm for pairing implementation. In [2] it was clarified how these methods can be used on a more general set of curves. In [2] it was also demonstrated that an even faster bilinear pairing can be returned by using a smaller iterative loop and a slightly more complicated final exponentiation. They call this pairing the truncated *eta* pairing (denoted η_T). In this paper we describe the hardware implementation of this pairing in characteristic 3.

Even with these algorithmic optimisations, pairing calculation remains a complicated operation. Pairings are very well suited to hardware implementation for two reasons: Much of the arithmetic performed on the extension field $\mathbb{F}_{q^k}$ can be reduced to arithmetic on $\mathbb{F}_q$. Many of these $\mathbb{F}_q$ arithmetic operations can be performed in parallel in hardware, providing a large saving over implementations on general purpose serial processors. Furthermore, a pairing calculation consists of an iterative loop followed by a final exponentiation. If care is taken with the scheduling of operations through the loop, a hardware implementation can provide a large level of pipelining, further speeding up pairing computation.

An FPGA implementation of the characteristic 3 pairing methods described in [13] has appeared in [11], where the extension field operations required for pairing computation are hardwired. However, this leads to a large design and provides no level of flexibility for different applications (the authors concern themselves with high throughput only). In [10] a pairing processor for the implementation of the pairings described in [8] and [13] is presented. An $\mathbb{F}_{3^m}$ ALU is used to perform the operations required for pairing computation. This processor again provides no means for reconfiguration. A study on the scheduling of the Duursma-Lee algorithm in [8] has appeared in [4]. Recently, an implementation of the characteristic 3 η_T pairing has appeared in [6]. The operations necessary for the iterative loop of the η_T pairing are hardwired. This results in a high operational frequency. However, there is no means for exponentiation using the architecture in [6]. This exponentiation is required such that the pairing returns a unique value, which is required for cryptographic applications. Thus, the architecture in [6] must be supplemented with a coprocessor for exponentiation, which, at the time of writing, is a work in progress and details are not included.

This paper is organised as follows: Section 2 presents the necessary mathematical preliminaries along with algorithms for computation of the η_T pairing on a characteristic 3 elliptic curve. Section 3 outlines the implementation of arithmetic on $\mathbb{F}_3$, $\mathbb{F}_{3^m}$ and $\mathbb{F}_{3^{6m}}$. Section 4 describes the reconfigurable processor used to perform the η_T pairing. Section 5 presents results returned by the processor when generated on the field $\mathbb{F}_{3^{97}}$ and implemented on an FPGA in various configurations. Finally, Section 6 draws some conclusions from the work.

2 Mathematical Preliminaries

Let $\mathbb{F}_q = \mathbb{F}_{p^m}$ be a finite field of characteristic p. The group of points on an elliptic curve E defined over $\mathbb{F}_q$ is denoted by $E(\mathbb{F}_q)$. A subgroup of $E(\mathbb{F}_q)$ of prime order r

has *embedding degree* k if r divides $q^k - 1$, but does not divide $q^i - 1$ for any $0 < i < k$. A subgroup of order r is known as an r-torsion group, denoted $E(\mathbb{F}_q)[r]$.

Following [3] and [9], the Tate pairing of order r is a bilinear map between $E(\mathbb{F}_q)[r]$ and $E(\mathbb{F}_{q^k})[r]$ to an element of the multiplicative group $\mathbb{F}^*_{q^k}$:

$$e_r(P, Q) : E(\mathbb{F}_q)[r] \times E(\mathbb{F}_{q^k})[r] \rightarrow \mathbb{F}^*_{q^k} \tag{1}$$

The second argument can be generated over $E(\mathbb{F}_q)$ and then mapped to $E(\mathbb{F}_{q^k})$ using a distortion map (denoted ϕ) [16]. This leads to a reduction in cost since more operations can be performed on the sub-field. The incorporation of the distortion map yields the modified Tate pairing

$$\hat{e}_r(P, Q) = e_r(P, \phi(Q)) \tag{2}$$

where now $P, Q \in E(\mathbb{F}_q)[r]$. In [8], it was shown how the distortion map can be incorporated into the formulae for this modified pairing computation.

The value $\hat{e}_r(P, Q)$ is only defined up to r-th powers. A final exponentiation must be performed to obtain a unique value for cryptographic purposes. The reduced modified Tate pairing is defined as

$$\hat{e}(P, Q) = \hat{e}_r(P, Q)^{(q^k - 1)/r} \tag{3}$$

In this paper we compute the η_T pairing on a characteristic 3 elliptic curve given by

$$E : y^2 = x^3 - x + b, \ b \in -1, 1 \tag{4}$$

over a field $\mathbb{F}_q = \mathbb{F}_{3^m}$ with $m \mod 12 \equiv 1, m \mod 6 \equiv 1$. This curve has an embedding degree of $k = 6$. The Tate pairing is computed with an iterative algorithm incorporating the intermediate tangent, chord and vertical line functions on the curve associated with the elliptic curve additive group operation. The η_T pairing reduces the number of iterations required to build a bilinear pairing at the expense of a more complicated final exponentiation when compared to the Tate pairing. In most cases, the reduction in computation time yielded by the smaller number of iterations far outweighs the slightly more costly exponentiation.

In characteristic 3, the η_T pairing is related to the Tate pairing with

$$\left(\eta_T(P, Q)^M\right)^{3T^2} = (\hat{e}_N(P, Q)^M)^L \tag{5}$$

where

$$\begin{aligned}
N &= 3^m + 1 + b.3^{(m+1)/2} \\
M &= (3^{6m} - 1)/N = (3^{3m} - 1)(3^m + 1)(3^m - b.3^{(m+1)/2} + 1) \\
T &= q - N = -1 - b.3^{(m+1)/2} \\
L &= -b.3^{(m+3)/2}
\end{aligned}$$

The interested reader is referred to [2] for the derivation of Eq. (5) and for more details on the proof of the η_T pairing.

The function $\eta_T(P,Q)^M$ is itself bilinear and returns a non-degenerate and unique value. It is, therefore, suitable as a basis for secure cryptographic protocols. The remainder of this paper deals with the efficient implementation of $\eta_T(P,Q)^M$ in hardware. We refer to the exponentiation to M as the *final exponentiation*. If compatibility with the Tate pairing is required, then the powering to $3T^2$ can be performed with a very small number of calculations (that are insignificant when compared to the overall cost of the pairing computation).

2.1 The Characteristic 3 η_T Pairing

In this section we describe the calculation of the η_T pairing on the curve $E : x^3 - x + b$ in affine coordinates defined on the field $\mathbb{F}_{3^m}$ in the $m \mod 12 \equiv 1,\ m \mod 6 \equiv 1$, $b = -1$ case. Note that other cases for m and b require only slight modifications (but no significant additional operations). Again, consider the points $P = (x_P, y_P)$, $Q = (x_Q, y_Q) \in E(\mathbb{F}_{3^m})[r]$ with $x_P, y_P, x_Q, y_Q \in \mathbb{F}_{3^m}$.

A suitable distortion map ϕ on this curve is

$$\phi(Q) = \phi(x_Q, y_Q) = (\rho - x_Q, \sigma y_Q) \tag{6}$$

where $\sigma \in \mathbb{F}_{3^2}$ and $\rho \in \mathbb{F}_{3^3}$ such that $\sigma^2 + 1 = 0$ and $\rho^3 - \rho + 1 = 0$.

A basis must be chosen for the representation of elements in $\mathbb{F}_{3^{6m}}$. The choice of basis is motivated by the desire to simplify the arithmetic operations in this field as much as possible. We choose to represent elements of $\mathbb{F}_{3^{6m}}$ using the basis $(1, \sigma, \rho, \sigma\rho, \rho^2, \sigma\rho^2)$. An element $A \in \mathbb{F}_{3^{6m}}$ is then represented as

$$\begin{aligned} A &= (a_0, a_1, a_2, a_3, a_4, a_5) \\ &= a_0 + a_1\sigma + a_2\rho + a_3\sigma\rho + a_4\rho^2 + a_5\sigma\rho^2 && (7) \\ &= \hat{a}_0 + \hat{a}_1\rho + \hat{a}_2\rho^2 && (8) \end{aligned}$$

where $a_0, a_1, a_2, a_3, a_4, a_5 \in \mathbb{F}_{3^m}$ and $\hat{a}_0 = a_0 + a_1\sigma,\ \hat{a}_1 = a_2 + a_3\sigma,\ \hat{a}_2 = a_4 + a_5\sigma$ with $\hat{a}_0, \hat{a}_1, \hat{a}_2 \in \mathbb{F}_{3^{2m}}$.

This is equivalent to a tower field representation [9] of $\mathbb{F}_{3^{6m}}$, i.e. $\mathbb{F}_{3^{6m}} \equiv \left((\mathbb{F}_{3^m})^2\right)^3$ where σ and ρ are zeros of $\sigma^2 + 1$ and $\rho^3 - \rho + 1$ as defined by the distortion map. The $\mathbb{F}_{3^{6m}}$ field is generated as

$$\mathbb{F}_{3^{2m}} = \mathbb{F}_{3^m}[y]/g[y] \tag{9}$$

where $g(y) = y^2 + 1$ is an irreducible polynomial over $\mathbb{F}_{3^m}$ and

$$\mathbb{F}_{3^{6m}} = \mathbb{F}_{3^{2m}}[z]/h[z] \tag{10}$$

where $h(z) = z^3 - z + 1$ is an irreducible polynomial over $\mathbb{F}_{3^{2m}}$.

The operations required to compute $\eta_T(P,Q)^M$, as described in [2], are shown in Algorithm 1. Note that using the tower of extensions, all arithmetic operations in Algorithm 1 are performed on $\mathbb{F}_{3^m}$. The algorithm begins with a precomputation stage in which powers of cubes of x_Q and y_Q are calculated and stored in memory. This means that the calculation of cube roots during loop iteration is avoided and the appropiate powers of x_Q and y_Q are instead extracted on each iteration. Although the calculation

Algorithm 1 Computation of $\eta_T(P,Q)^M$ on $E(\mathbb{F}_{3^m}) : y^2 = x^3 - x - 1, m \mod 12 \equiv 1, m \mod 6 \equiv 1$ case

```
INPUT: $P = (x_P, y_P),\ Q = (x_Q, y_Q),\ (P,Q) \in E(\mathbb{F}_{3^m})$
OUTPUT: $f = \eta_T(P,Q) \in \mathbb{F}^*_{3^{6m}}$
INITIALISE:
 1: $f \in \mathbb{F}_{3^{6m}}$
 2: $f \leftarrow 1$
PRECOMPUTE:
 3: for $i \leftarrow m-1$ downto $(m+1)/2+1$ do
 4:     ${x_Q} \leftarrow {x_Q}^3, y_Q \leftarrow {y_Q}^3$                          – $2\mathbb{F}_{3^m}$ cubings
 5: end for
 6: for $i \leftarrow (m+1)/2$ downto $0$ do
 7:     $x_Q \leftarrow {x_Q}^3, y_Q \leftarrow {y_Q}^3$                           – $2\mathbb{F}_{3^m}$ cubings
 8:     $x'_Q[i] \leftarrow x_Q, y'_Q[i] \leftarrow y_Q$
 9: end for
RUN:
10: for $i \leftarrow 1$ to $(m+1)/2$ do
11:     $u \leftarrow x_P + x'_Q[i] + 1$
12:     $c_0 \leftarrow -u.u$                                                    – $1\mathbb{F}_{3^m}$ mul
13:     $c_1 \leftarrow -y_P.y'_Q[i]$                                            – $1\mathbb{F}_{3^m}$ mul
14:     $g \leftarrow c_0 + (c_1)\sigma + (-u)\rho + (0)\sigma\rho + (-1)\rho^2 + (0)\sigma\rho^2$
15:     $f \leftarrow f.g$                                                       – $13\mathbb{F}_{3^m}$ muls
16:     if $i < (m+1)/2$ then
17:         $x_P \leftarrow x_P^3, y_P \leftarrow y_P^3$                           – $2\mathbb{F}_{3^m}$ cubings
18:     end if
19:     $g \leftarrow -y_P.(x'_Q[i] + x_P + 1) + (-y'_Q[i])\sigma + (y_P)\rho$     – $1\mathbb{F}_{3^m}$ mul
20:     $f \leftarrow f.g$                                                       – $10\mathbb{F}_{3^m}$ muls
21: end for
22: return $f^{(3^{3m}-1)(3^m+1)(3^m+3^{(m+1)/2}+1)}$                             –See Algorithm 2
```

of cube roots in characteristic 3 has been simplified in [1], the roots are stored in this way to ensure an efficient use of area in the hardware implementation by avoiding the additional area cost that would be incurred by a cube rooting circuit.

The most costly part of the pairing is the loop iteration. Each iteration requires fifteen $\mathbb{F}_{3^m}$ multiplications, a number of $\mathbb{F}_{3^m}$ additions and subtractions and two $\mathbb{F}_{3^m}$ cubings. The scheduling of these operations through the hardware processor will be vital to computation speed. The multiplication of f by g can be performed with a regular $\mathbb{F}_{3^{6m}}$ multiplication (requiring 18 $\mathbb{F}_{3^m}$ multiplications). However, this does not make use of the fact that g has the special form $g = (g_0 + g_1\sigma + g_2\rho + (0)\sigma\rho + (-1)\rho^2 + (0)\sigma\rho$. If the $\mathbb{F}_{3^{6m}}$ multiplication is unrolled and examined, it becomes apparent that $f.g$ can be performed in only 13 multiplications, providing a saving of five multiplications. The $\mathbb{F}_{3^m}$ operations required for the multiplication of $f = (f_0, f_1, f_2, f_3, f_4, f_5)$ by $g = (g_0, g_1, g_2, 0, -1, 0)$, including the required reductions are shown in (11), (12) and (13). Note that all 13 $\mathbb{F}_{3^m}$ multiplications can be performed in parallel.

$$
\begin{array}{ll}
mul_0 \leftarrow f_0.g_0 & mul_6 \leftarrow (f_1 + f_3).(g_1) \\
mul_1 \leftarrow f_1.g_1 & mul_7 \leftarrow (f_0 + f_1 + f_2 + f_3).(g_0 + g_1 + g_2) \\
mul_2 \leftarrow (f_0 + f_1).(g_0 + g_1) & mul_8 \leftarrow (f_0 + f_4).(g_0 + 2) \\
mul_3 \leftarrow f_2.g_2 & mul_9 \leftarrow (f_1 + f_5).(g_1) \\
mul_4 \leftarrow f_3.g_2 & mul_{10} \leftarrow (f_0 + f_1 + f_4 + f_5).(g_0 + g_1 + 2) \\
mul_5 \leftarrow (f_0 + f_2).(g_0 + g_2) & mul_{11} \leftarrow (f_2 + f_4).(g_2 + 2) \\
 & mul_{12} \leftarrow (f_3 + f_5).(g_2 + 2)
\end{array}
\quad (11)
$$

Algorithm 2 Exponentiation of $\eta_T(P,Q)$ to $\eta_T(P,Q)^M$ on $E(\mathbb{F}_{3^m}) : y^2 = x^3 - x - 1$, $m\%12 \equiv 1$, $m\%6 \equiv 1$ case

```
INPUT: $f = \eta_T(P,Q) \in \mathbb{F}^*_{3^{6m}}$
OUTPUT: $f^{(3^{3m}-1)(3^m+1)(3^m+3^{(m+1)/2}+1)} = \eta_T(P,Q)^M \in \mathbb{F}^*_{3^{6m}}$
 1: INITIALISE:
 2: $n, d, w \in \mathbb{F}_{3^{6m}}$
 3: $w \leftarrow f$
 4: RUN:
 5: for $i \leftarrow 0$ to $(m+1)/2 - 1$ do
 6:     $w \leftarrow w^3$                         – $\mathbb{F}_{3^{6m}}$ cube
 7: end for
 8: $d \leftarrow f.w$                            – $1\mathbb{F}_{3^{6m}}$ mul
 9: $f \leftarrow f^q$                            – $1\mathbb{F}_{3^{6m}}$ powq
10: $n \leftarrow f$
11: $f \leftarrow f^q$                            – $1\mathbb{F}_{3^{6m}}$ powq
12: $d \leftarrow d.f$                            – $1\mathbb{F}_{3^{6m}}$ mul
13: $f \leftarrow f^q$                            – $1\mathbb{F}_{3^{6m}}$ powq
14: $t \leftarrow f^2$                            – $1\mathbb{F}_{3^{6m}}$ mul
15: $d \leftarrow d.t$                            – $1\mathbb{F}_{3^{6m}}$ mul
16: $f \leftarrow f^q$                            – $1\mathbb{F}_{3^{6m}}$ powq
17: $d \leftarrow d.f$                            – $1\mathbb{F}_{3^{6m}}$ mul
18: $f \leftarrow f^q$                            – $1\mathbb{F}_{3^{6m}}$ powq
19: $n \leftarrow n.f$                            – $1\mathbb{F}_{3^{6m}}$ mul
20: $f \leftarrow f^q$                            – $1\mathbb{F}_{3^{6m}}$ powq
21: $t \leftarrow f^3$                            – $1\mathbb{F}_{3^{6m}}$ cube
22: $n \leftarrow n.t$                            – $1\mathbb{F}_{3^{6m}}$ mul
23: $w \leftarrow w^{q^2}$                        – $1\mathbb{F}_{3^{6m}}$ powq$^2$
24: $n \leftarrow n.w$                            – $1\mathbb{F}_{3^{6m}}$ mul
25: $w \leftarrow w^q$                            – $1\mathbb{F}_{3^{6m}}$ powq
26: $n \leftarrow n.w$                            – $1\mathbb{F}_{3^{6m}}$ mul
27: $w \leftarrow w^{q^2}$                        – $1\mathbb{F}_{3^{6m}}$ powq$^2$
28: $d \leftarrow d.w$                            – $1\mathbb{F}_{3^{6m}}$ mul
29: $f \leftarrow n$
30: $f \leftarrow f/d$                            – $1\mathbb{F}_{3^{6m}}$ inv, $1\mathbb{F}_{3^{6m}}$ mul
31: return $f$
```

$$\begin{array}{ll}
t00_r \leftarrow mul_0 - mul_1 & t00_i \leftarrow mul_2 - mul_0 - mul_1 \\
t11_r \leftarrow mul_3 & t11_i \leftarrow mul_4 \\
t22_r \leftarrow -f_4 & t22_i \leftarrow -f_5 \\
t01_r \leftarrow mul_5 - mul_6 & t01_i \leftarrow mul_7 - mul_5 - mul_6 \\
t02_r \leftarrow mul_8 - mul_9 & t02_i \leftarrow mul_{10} - mul_8 - mul_9 \\
t12_r \leftarrow mul_{11} & t12_i \leftarrow mul_{12}
\end{array} \tag{12}$$

$$\begin{array}{l}
c_0 \leftarrow t00_r - t12_r + t11_r + t22_r \\
c_1 \leftarrow t00_i - t12_i + t11_i + t22_i \\
c_2 \leftarrow t01_r - t00_r + t11_r + t12_r + t22_r \\
c_3 \leftarrow t01_i - t00_i + t11_i + t12_i + t22_i \\
c_4 \leftarrow t02_r - t00_r + t11_r \\
c_5 \leftarrow t02_i - t00_i + t11_i
\end{array} \tag{13}$$

The computation $\eta_T(P,Q)$ finishes with an exponentiation to M. This exponentiation is vital since it ensures a unique output value for the overall computation. The exponent M is unrolled and the operations required to perform the final exponentiation are detailed in Algorithm 2. The exponentiation requires $(m+1)/2+1$ $\mathbb{F}_{3^{6m}}$ cubings, seven powerings to $q = 3^m$, two powerings to q^2, eleven $\mathbb{F}_{2^{4m}}$ multiplications and an $\mathbb{F}_{3^{6m}}$ inversion. The computation of these operations is performed using the tower of extensions idea and will be detailed later. Note that due to the serial nature of the exponentiation, it is vital that these operations are performed efficiently if a low computation time for $\eta_T(P,Q)^M$ is to be returned.

3 Characteristic 3 Arithmetic

In this section we consider the hardware implementation of $\mathbb{F}_3$, $\mathbb{F}_{3^m}$ and $\mathbb{F}_{3^{6m}}$ arithmetic for the computation of $\eta_T(P,Q)^M$. We provide results when the arithmetic operations are implemented over a base field $\mathbb{F}_{3^{97}}$. This base field was chosen as it affords us the opportunity to compare our implementation with others in the literature. The irreducible polynomial used is $x^{97}+x^{16}-1$. However, the components described in this section can be reconfigured for any other suitable field size or irreducible polynomial. The FPGA on which the arithmetic is implemented is a *Xilinx Virtex-II Pro* FPGA (xc2vp100-6ff1696) with 45120 slices. All components have been realised using VHDL and synthesised and placed & routed using *Xilinx ISE* Version 8.1i.

3.1 $\mathbb{F}_3$ Arithmetic

Unlike the characteristic 2 field on which the additive and multiplicative operations can be performed using only one xor- gate and one and-gate respectively, $\mathbb{F}_3$ arithmetic operations are slightly more complicated to perform in hardware. Each element $\{0,1,2\} \in \mathbb{F}_3$ must be represented using two hardware bits. We choose the representation $0 = \{00, 01\}$, $1 = 10$ and $2 = 11$. This representation means that the *check if zero* operation can be performed by simply checking the high bit of an $\mathbb{F}_3$ element.

On $\mathbb{F}_3$, the required arithmetic operations are addition, subtraction and multiplication (negation can be performed with a subtraction from zero). The underlying logic unit on FPGAs are function generators, which can be configured as $4:1$ Look-Up tables (LUTs). We make explicit use of these LUTs in generating circuitry for the additive and multiplicative $\mathbb{F}_3$ operations. The input-output map for each of the arithmetic operations can be placed on two $4:1$ look up tables, or one FPGA slice.

3.2 $\mathbb{F}_{3^m}$ Arithmetic

The $\mathbb{F}_{3^m}$ operations that are required for the computation of $\eta_T(P,Q)^M$ are addition, subtraction, cubing and multiplication. An $\mathbb{F}_{3^m}$ inversion is also required for the final extension field division in Algorithm 2. Note that $\mathbb{F}_{3^m}$ squaring is performed using multiplication circuitry. The arithmetic components used to implement $\mathbb{F}_{3^m}$ are described in Table 1.

Addition, subtraction and cubing in $\mathbb{F}_{3^m}$ are combinatorial and incur a relatively small area cost. Inversion, however, is generally a complicated operation in any field. A hardware inverter costs 1939 slices and returns a result in $2m$ clock cycles. Fortunately, inversion must only be performed once. Multiplication can be performed using two

Op	Description	Cycles	Area(Slices)
Add/Sub	Performed using m $\mathbb{F}_3$ addition or subtraction units. Combinatorial logic only.	1	97
Cube	Created as an xor-gate array using methods from [5]. Combinatorial logic only.	1	120
Inv	Performed using the Extended Euclidean Algorithm (EEA). The EEA operations are modified for a characteristic 3 implementation [12].	$2m$	1939
Mult			
D≡1	A serial *Most Significant Coefficient First* (MSC) Multiplier [12] is used. Only one input coefficient operated on at a time.	m	637

D>1	Digit Multipliers of type described in [15] are used. D coefficients of the inputs are processed in parallel, where D is known as the *Digit Size*. The area of the multiplier increases with D. These multipliers offer a direct trade-off between speed and area.	m/D	D=4: 1224
			D=8: 2049
			D=16: 3737

Table 1. Implementation of Arithmetic on $\mathbb{F}_{3^m}$

different types of multipliers. The first is a low area cost serial approach utilising a *Most Significant Coefficient First* (MSC) multiplier. In this case, coefficients of the inputs are operated on one at a time. The MSC multiplier returns a result in m clock cycles. The second approach is to use *Digit Multipliers* such that D coefficients of the inputs are operated on in parallel. These multipliers return results in m/D clock cycles and provide a speed/area trade-off.

3.3 $\mathbb{F}_{3^{6m}}$ Arithmetic

As seen in Algorithm 2, various extension field operations must be performed during the final exponentiation. However, using the tower of extensions idea, arithmetic on $\mathbb{F}_{3^{6m}}$ can be reduced to operations on $\mathbb{F}_{3^m}$. This means that the components described in the previous section can be reused to perform the $\mathbb{F}_{3^{6m}}$ operations. The $\mathbb{F}_{3^{6m}}$ operations required by Algorithm 2 are shown in Table II.

Note that the $\mathbb{F}_{3^{6m}}$ cubing, *powq* and *powq2* operations require only $\mathbb{F}_{3^m}$ cubes and additions/subtractions, which are combinatorial. Extension field multiplication and inversion are, however, more costly to compute.

4 The η_T processor

This section describes the reconfigurable processor designed to compute the secure bilinear pairing $\eta_T(P,Q)^M$. The processor was designed with flexibility and versatility in mind and is shown in Figure 1.

Rather than hardwiring the logic to compute the pairing, the processor is implemented with an ALU containing a number of $\mathbb{F}_{3^m}$ arithmetic components. The ALU

Op	**Method of Computation**	**Cost on** $\mathbb{F}_{3^m}$
Cube	Let $A = \hat{a}_0 + \hat{a}_1\rho + \hat{a_2}\rho^2$ where $\hat{a}_0 = (a_0 + a_1\sigma),\ \hat{a}_1 = (a_2 + a_3\sigma),\ \hat{a}_2 = (a_4 + a_5\sigma)$ (see (8)) $\Rightarrow A^3 = \hat{a}_0^3 + \hat{a}_1^3\rho^3 + \hat{a_2}^3\rho^6$ But $\rho^3 = \rho - 1$ and $\rho^6 = \rho^2 + 1$. Reduce: (see (6)) $\Rightarrow A^3 = (\hat{a}_0^3 - \hat{a}_1^3 + \hat{a}_2^3) + ((\hat{a}_1^3 + \hat{a}_2^3)\rho + \hat{a}_3^3\rho^2$ But $\sigma^2 = -1, \Rightarrow \hat{a}_0^3 = ({a_0}^3 - \sigma {a_1}^3),\ \hat{a}_1^3 = ({a_2}^3 - \sigma {a_3}^3), \hat{a}_2^3 = ({a_4}^3 - \sigma {a_5}^3)$ $\Rightarrow A^3 = ({a_0}^3 - {a_2}^3 + {a_4}^3) + ({a_3}^3 - {a_1}^3 - {a_5}^3)\sigma + ({a_2}^3 + {a_4}^3)\rho + (-{a_3}^3 - {a_5}^3)\sigma\rho$ $+({a_4}^3)\rho^2 + (-{a_5}^3)\sigma\rho^2$	6 cubings, 8 sub/adds
PowQ	Let $A = \hat{a}_0 + \hat{a}_1\rho + \hat{a_2}\rho^2$ again: $\Rightarrow A^q = A^{3^m} = \hat{a}_0^{3^m} + \hat{a}_1^{3^m}\rho^{3^m} + \hat{a_2}^3(\rho^2)^{3^m}$ $\Rightarrow A^q = \hat{a}_0 + \hat{a}_1\rho^3 + (\hat{a}_1)\rho^6$ $\Rightarrow A^q = (a_0 - a_2 + a_4) + (a_3 - a_1 - a_5)\sigma + (a_2 + a_4)\rho + (-a_3 - a_5)\sigma\rho$ $+(a_4)\rho^2 + (-a_5)\sigma\rho^2$	8 sub/adds
$PowQ^2$	Reapply *powq*: $\Rightarrow A^{q^2} = (a_0 + a_2 + a_4) + (a_3 + a_1 + a_5)\sigma + (a_2 - a_4)\rho + (a_3 - a_5)\sigma\rho$ $+(a_4)\rho^2 + (a_5)\sigma\rho^2$	6 sub/adds
Mul	Given $A = (\hat{a}_0 + \hat{a}_1\rho + \hat{a_2}\rho^2)$, $B = (\hat{b}_0 + \hat{b}_1\rho + \hat{b_2}\rho^2)$ perform Karatsuba Multiplication of $A.B$ over $\mathbb{F}_{3^{2m}}$ followed by a reduction by $\rho^3 = \rho - 1$. The Karatsuba Multiplication costs six $\mathbb{F}_{3^{2m}}$ multiplications. Each $\mathbb{F}_{3^{2m}}$ multiplication requires three multiplications on $\mathbb{F}_{3^m}$. This means that, in total, 18 $\mathbb{F}_{3^m}$ multiplications are required for $\mathbb{F}_{3^{6m}}$ multiplication. All multiplications can be performed in parallel if desired. See [11] for more details.	18 muls, 72 add/subs
Inv	Use method in [11]. Change representation from $\{1, \sigma, \rho, \sigma\rho, \rho^2, \sigma\rho^2\}$ to $\{1, \rho, \rho^2, \sigma, \sigma\rho, \sigma\rho^2\}$, i.e. $A = \hat{a_0} + \hat{a_1}\sigma$ where $\hat{a_0} = a_0 + a_1\rho + a_2\rho^2$ and $\hat{a_1} = a_3 + a_4\rho + a_5\rho^2$. Recalling that $\sigma^2 = -1$, the inversion can be carried out efficiently using conjugate methods. Only one $\mathbb{F}_{3^m}$ inversion is required.	33 muls, 4 cubings, 67 add/subs, 1 inv

Table 2. Implementation of Arithmetic on $\mathbb{F}_{3^{6m}}$

contains one arithmetic unit for $\mathbb{F}_{3^m}$ addition, $\mathbb{F}_{3^m}$ subtraction, $\mathbb{F}_{3^m}$ cubing and $\mathbb{F}_{3^m}$ inversion. As multiplication is performed so often and is a relatively time consuming operation (requiring m/D clock cycles), the processor has been designed such that the number of multipliers in the ALU can be reconfigured with little impact on the overall architecture. By varying the number of multipliers, and indeed their digit sizes, the processor can easily be tailored for a low area implementation, a high throughput implementation, or a desired compromise between the two.

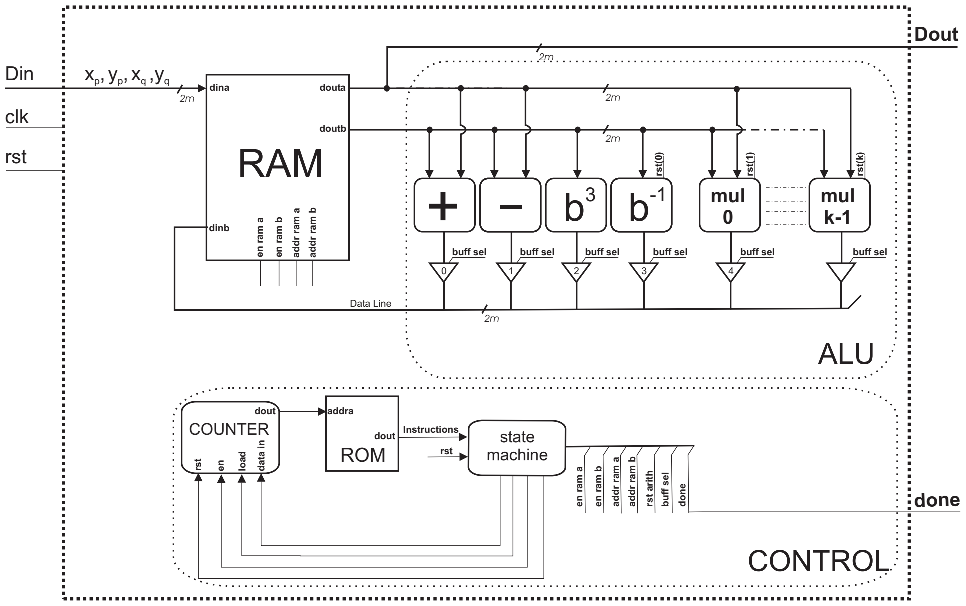


Fig. 1. The characteristic 3 elliptic curve η_T processor

Dual port RAM is used to store the intermediate variables required for computation. The required input coordinates are read serially into the RAM before computation begins. During computation, two $2m$-bit data signals bring variables to the ALU to be operated on. Tri-state buffers at the output of the arithmetic components are used to select which result is written to RAM when the $\mathbb{F}_{3^m}$ operation has been completed.

A control unit consisting of a ROM, a counter and a simple state machine is used to perform the pairing operation. An instruction set sequencing the operations required for the η_T pairing is loaded into the ROM. A counter controls the address of the ROM. When the processor is reset, the counter begins to iterate through the instruction set. A simple state machine checks bits of the instruction vector and halts the counter for m/D clock cycles and m clock cycles when a multiplication and an inversion are being performed respectively so that the correct result will be written to RAM. The counter also contains a *load* control bit such that the counter can jump to particular addresses in the instruction set when required (for example jumping from the end of the loop to the beginning of the loop in Algorithm 1 when required). The state machine also handles the data load. This control methodology ensures flexibility as it eliminates the requirement for a large fixed state machine. When a new instruction set is required, it can simply be loaded into the ROM.

4.1 Processor Generation

To facilitate ease of processor reconfiguration and to ensure flexibility, the VHDL code for the processor is generated using a C program. Using C code the field size, the number of multipliers and their digit sizes, the instruction set and, indeed, the size of the memory blocks, can be automatically reconfigured according to the application.

The instruction set that implements the bilinear pairing is also generated in C code and written to a file that is loaded into the ROM. The instruction set is generated very efficiently in C through the use of operator overloading. Arithmetic operators are overloaded such that an instruction of the type *X=Y+Z* written in C code will automatically generate instructions that send X from RAM port a, Y from RAM port b and set the tri-state buffers, the RAM enable signals and the RAM address signals such that the result is written from the addition circuitry to X after a clock cycle. This yields a very rapid generation of an instruction set for a particular application. Additional operations such as elliptic curve point scalar multiplication as required by some pairing-based protocols can also be added to the instruction set with ease.

4.2 Operation Scheduling

It is vital that the scheduling of operations be as efficient as possible to ensure that valuable clock cycles are not wasted. Operations are scheduled such that, if possible, additions, subtractions and cubings are performed and their results written to RAM while multiplication or inversion is in progress. In particular, the scheduling of operations through the loop of Algorithm 1 is vital since the loop is performed $(m + 1)/2$ times. To achieve a fast throughput through the loop, hardware pipelining is performed. Before the loop begins to iterate, the values of $c_0 = -u_1.u_1$ and $c_1 = -y_P.y'_Q[i]$ are computed and subsequently stored. On the first iteration of the loop, the calculation of $f.g$ can then proceed almost immediately as the inputs to the operation will be available. The values of c_0 and c_1 that will be required on the following iteration can be calculated in parallel with these multiplications and stored. This means that after finding the initial values of c_0 and c_1, all 15 $\mathbb{F}_{3^m}$ multiplications can be performed in parallel if desired. Note also that the additions required to compute f, as per (12) and (13), are also pipelined in a similar manner by calculating the f coefficients of the previous loop while multiplications are being performed during the current iteration. It is worthwhile noting that only a small level of this form of parallel scheduling can be achieved during the final exponentiation to M as described in Algorithm 2. This means that while the number of operations required for exponentiation is far less than for the loop iterations, the time required for exponentiation is not trivial due to its serial nature and effort should be made to minimise the number of operations required as much as possible.

5 Results

The Tate pairing processor described in the previous section was implemented over the same $\mathbb{F}_{3^{97}}$ base field and on the same FPGA described at the beginning of the previous section. Note that the processor is fully reconfigurable for any suitable field size.

Results returned by the processor when $\eta_T(P, Q)^M$ is implemented with 1, 2, 3, 4, 5 and 8 multipliers are provided in Tables III and IV. Note that an implementation with either 6 or 7 multipliers would not yield an efficient use of resources. This is because

	Number of Multipliers					
D_m	**1**	**2**	**3**	**4**	**5**	**8**
	Area of the η_T Processor (slices)					
1	3610	4420	5190	6069	7093	9652
4	4125	5657	7265	8887	10540	15401
8	4995	7491	10000	12520	15056	22632
16	7036	11635	16290	20943	25626	39553
	Execution Time for $\eta_T(P,Q)$ (μs)					
1	862	482	322	272	221	177
4	302	185	137	127	120	119
8	198	130	114	115	113	113
16	200	162	151	154	151	155
	Execution Time for Exp. to M (μs)					
1	308	177	133	119	104	91
4	130	89	76	72	67	64
8	96	73	65	63	60	59
16	109	88	81	79	77	79
	Execution Time for $\eta_T(P,Q)^M$ (μs)					
1	1170	659	454	391	325	268
4	432	275	213	199	187	183
8	294	203	178	178	173	172
16	309	250	232	233	228	227

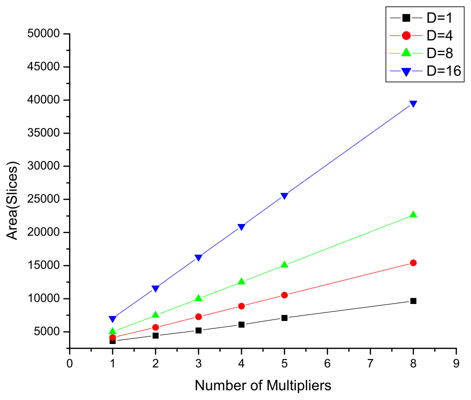
Table 3. Required area with execution times returned by the processor for $\eta_T(P,Q)$, the final exponentiation and for computation of the full $\eta_T(P,Q)^M$ pairing, f_{CLK}=91.2,84.8,70.4 and 61.6MHz for D=1,4,8 and 16 respectively

	Number of Multipliers					
D_m	**1**	**2**	**3**	**4**	**5**	**8**
	Area Time Product (slices.μs)					
1	4.22	2.91	2.36	2.37	2.31	2.59
4	1.78	1.55	1.55	1.77	1.97	2.82
8	1.47	1.52	1.78	2.23	2.60	3.88
16	2.17	2.90	3.78	4.88	5.85	8.97
	Clock Cycles for $\eta_T(P,Q)$					
1	78653	44007	29350	24787	20124	16147
4	25589	15711	11638	10795	10140	10089
8	16745	11043	9646	9783	9538	9581
16	12323	9984	9338	9483	9330	9569
	Clock Cycles for Exp. to M					
1	28061	16143	12103	10842	9529	8320
4	10997	7575	6415	6090	5713	5440
8	8153	6147	5467	5298	5077	4870
16	6731	5433	4993	4902	4759	4876
	Clock Cycles for $\eta_T(P,Q)^M$					
1	106714	60150	41453	35629	29653	24467
4	36586	23286	18053	16885	15853	15529
8	24898	17190	15113	15081	14615	14451
16	19054	15417	14331	14385	14089	14445

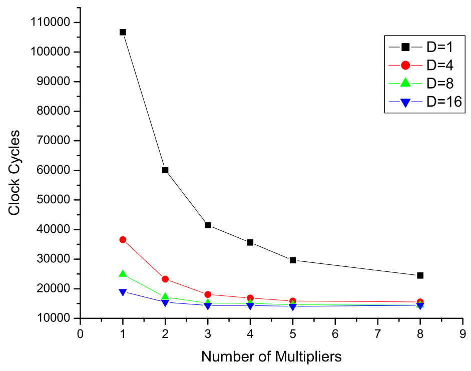
Table 4. Area Time Product and Clock Cycles Required by the Processor for the execution of $\eta_T(P,Q)$, the final exponentiation and for computation of the full $\eta_T(P,Q)^M$ pairing.

the multiplications required within the loop of Algorithm 1 dominate the pairing calculation time. The latency of these 15 multiplications does not improve between 5 and 7 (it remains at $3m/D$ until a configuration incorporating 8 multipliers is instantiated). Operations required by the $\mathbb{F}_{3^{6m}}$ multiplication are also scheduled to make use of the various numbers of multipliers.

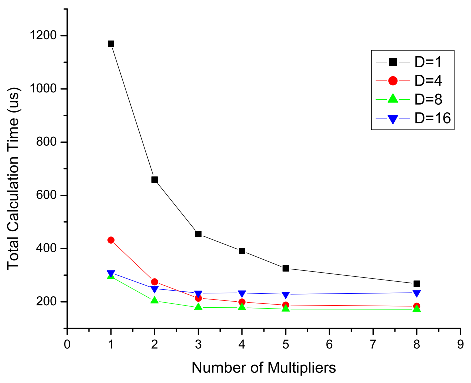
Results are returned by the processor when implemented with serial MSC multipliers ($D \equiv 1$) and for digit multipliers with digit sizes of 4, 8 and 16. Note that when



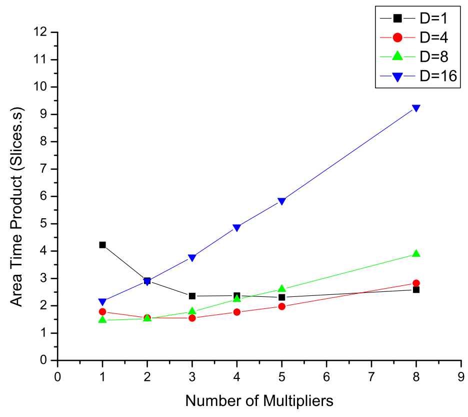
(a) Total area for the η_T Processor in slices



(b) Total number of clock cycles for $\eta_T(P,Q)^M$



(c) Total time for computation of $\eta_T(P,Q)^M$ in μs



(d) Area-Time Product for the η_T Processor in slices.s

Fig. 2. η_T processor results when implemented with 1, 2, 3, 4, 5 and 8 multipliers with D=1, 4, 8, 16

configured with the MSC multiplier, the maximum attainable post place and route clock frequency of the processor is 91.2MHz. The maximum frequencies for the processor when implemented with digit multipliers of D=4, 8 and 16 are 84.8MHz, 70.4MHz and 61.6MHz respectively. The computation times in Tables III and IV are returned by the processor at these post place & route frequencies.

Figure 2 illustrates trends in the area of the processor (slices), the total number of clock cycles required for $\eta_T(P,Q)^M$, the total computation time (μs) and the area-time product returned for the various configurations ($slices.s$). From Table III and Figure 2(a) it can be seen that the processor can be configured for a large range of area costs, depending on the required computation time and application (3610–39553 slices).

As seen in Figure 2(b) the total number of clock cycles decreases as both the digit size and number of multipliers increase. An increase in the number of multipliers in the $D = 1$ case provides for a dramatic reduction in the number of clock cycles required since in this case each multiplicative operation consumes a relatively large number (m) of clock cycles. The effect is less dramatic in the $D > 1$ cases since not only are the multiplicative operations less dominant but there is less time to pipeline intermediate additions while the multiplications are being performed.

Figure 2(c) illustrates trends in the time taken to compute $\eta_T(P,Q)$ and provides some interesting results. Here, the decrease in clock frequency as the digit size is increased has an effect on the computation. In the $D = 16$ case, the reduced frequency yielded by the larger digit size results in longer computation times than the $D = 8$ case and in the $\{D$=4, number of mults$>2\}$ cases since the multiplicative operations are taking a relatively small amount of time and are thus not so dominant when compared to the quantity of additive operations. From Table III the fastest pairing is returned in 172μs for the $\{$D=8, number of muls=8$\}$ case. However, note from Table III that increasing the number of multipliers from 3 to 8 for this digit size returns a very small decrease in computation time. A more efficient option for a fast throughput architecture is the $\{$D=8, number of muls=3$\}$ case, which requires only $10,000$ slices (22.6% of the FPGA) and returns a cryptographically secure pairing in 178μs.

Trends in the area-time (AT) product of the processor are illustrated in Figure 2(d) and are, perhaps, the most revealing since a measure of the efficiency of the processor in the various configurations is provided. In the $D \equiv 1$ case, the AT product reduces continuously until the number of multipliers is 5. Increasing the number of multipliers to 8 yields a less efficient utilisation of area as the extra 3 multipliers do not provide a large enough reduction in computation time to warrant their inclusion. The AT product floor is hit when the number of multipliers is 3 in the $D = 4$ case. For digit sizes of 8 and 16, the most efficient utilisation of resources is returned by a processor with just one multiplier as the latency through the multipliers is reduced and the additions begin to dominate the computation time. Overall, the most efficient use of resources is returned by the $\{D = 8$, number of muls=2$\}$ case. This configuration costs only 7491 slices (17% of the FPGA) and returns a fully secure pairing in 203μs.

In the future, data trends for various different field sizes will be generated to investigate the best configuration for the η_T processor at various security levels.

5.1 Comparisons

Consider, for example, our pairing result of 178μs or 15,113 clock cycles utilising 10,000 slices returning an area-time product of 1.78 $slices.s$ in the $\{$D=8, number of muls=2$\}$ case. The pairing computation time provides a large speedup over the software implementation time of 2.72ms on a Pentium IV processor running at 3GHz in [2]. It also provides a large improvement on the estimated pairing time of $0.85ms$ (at 15MHz) reported for a characteristic 3 hardware pairing implementation in [11]. The architecture in [11] contains hardwired extension field arithmetic components, which reduces the attainable frequency and returns a large area cost. In [10] the characteristic 3 pairing algorithms of [13] and [8] are performed in 64887 and 69543 clock cycles respectively with an area of 4481 slices. No post place and route frequency values are provided. Our implementation compares favourably with these results.

6 Conclusions

In this paper we have discussed the efficient implementation of the η_T pairing in characteristic 3. All $\mathbb{F}_3$, $\mathbb{F}_{3^m}$ and $\mathbb{F}_{3^{6m}}$ operations required to compute a secure cryptographic pairing (including final exponentiation) have been discussed and their hardware implementation considered. A reconfigurable hardware processor for the η_T pairing has been presented. As far as we are aware, this is the first processor in the literature that computes e bilinear pairing including final exponentiation using the η_T methods in characteristic 3. The cryptographic processor can be tailored for various applications as it is fully reconfigurable for any suitable field size, for various numbers of multipliers and for different multiplier digit sizes. The efficient scheduling of operations through this processor has been considered and a pipelining methodology outlined. Results returned by the processor when implemented over a field size of $\mathbb{F}_{3^{97}}$ have been presented. At present, the processor returns the fastest characteristic 3 cryptographic pairing returning a unique value in the literature.

In the future it will be interesting to analyse the results returned by the processor for various different field sizes. It would also be interesting to add an instruction set for the implementation of point scalar multiplication as this operation is required by many pairing based protocols.

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. Available from `http://eprint.iacr.org/2004/305`.
2. P. S. L. M. Barreto, S. Galbraith, C. Ó hÉigeartaigh, and M. Scott. Pairing computation on supersingular abelian varieties. Cryptology ePrint Archive, Report 2004/375, 2004. Available from `http://eprint.iacr.org/2004/375`.
3. 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*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–368. Springer-Verlag, 2002.
4. G. Bertoni, L. Breveglieri, P. Fragneto, and G. Pelosi. Parallel Hardware Architectures for the Cryptographic Tate Pairing. In *Information Technology: New Generations*, pages 186–191. 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 *Topics in Cryptology - CT RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 158–175. Springer-Verlag, 2003.
6. 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. `http://eprint.iacr.org/2006/327`.
7. R. Dutta, R. Barua, and P. Sarkar. Pairing-based cryptography: A survey. Cryptology ePrint Archive, Report 2004/064, 2004. `http://eprint.iacr.org/2004/064`.

8. I. Duursma and H.-S. Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In *Advances in Cryptology – Asiacrypt'2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 111–123. Springer-Verlag, 2003.
9. S. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In *Algorithmic Number Theory – ANTS V*, volume 2369 of *Lecture Notes in Computer Science*, pages 324–337. Springer-Verlag, 2002.
10. P. Grabher and D. Page. Hardware acceleration of the tate pairing in characteristic 3. In *Cryptographic Hardware and Embedded Systems (CHES)*, volume 3659 of *Lecture Notes in Computer Science*, pages 398–411. Springer-Verlag, 2005.
11. T. Kerins, W.P. Marnane, E.M. Popovici, and P.S.L.M. Barreto. Efficient hardware for the Tate pairing calculation in characteristic 3. In *Cryptographic Hardware and Embedded Systems (CHES)*, volume 3659 of LNCS, pages 412–426. Springer, 2005.
12. Tim Kerins, Emanuel M. Popovici, and William P. Marnane. Algorithms and architectures for use in FPGA implementations of identity based encryption schemes. In *FPL*, pages 74–83, 2004.
13. S. Kwon. Efficient Tate pairing computation for elliptic curves over binary fields. In *Australasian Conference on Information Security and Privacy – ACISP 2005*, volume 3574 of *Lecture Notes in Computer Science*, pages 134–145. Springer-Verlag, 2005.
14. V. S. Miller. Short programs for functions on curves. Unpublished manuscript, 1986. `http://crypto.stanford.edu/miller/miller.pdf`.
15. L. Song and K.K. Parhi. Low-energy digit-serial/parallel finite field multipliers. *Journal of VLSI Signal Processing Systems*, 2(22):1–17, 1997.
16. E. Verheul. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. In *Advances in Cryptology – Eurocrypt'2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 195–210. Springer-Verlag, 2001.