A Dedicated Processor for the eta Pairing

Robert Ronan[1], Colm O hEigeartaigh[2], Colin Murphy[1], Michael Scott[2], Tim Kerins[1] and W. P. Marnane[1]

[1] Department of Electrical & Electronic Engineering, University College Cork, College Road, Cork, Ireland.
{robertr,cmurphy,timk}@rennes.ucc.ie
marnane@ucc.ie
[2] School of Computing, Dublin City University, Ballymun, Dublin 9, Ireland.
{coheigeartaigh,mike}@computing.dcu.ie

Abstract. The η pairing is an efficient computation technique based on a generalization of the Duursma Lee method for calculating the Tate pairing. The pairing can be computed very efficiently on genus 2 hyperelliptic curves. In this paper it is demonstrated that this pairing operation is well suited to a dedicated parallel hardware implementation. An η pairing processor is described in detail and the architectures required for such a system are discussed. Prototype implementation results are presented over a base field of $\mathbb{F}_{2^{103}}$ and the advantages of implementing the pairing on the dedicated processor are discussed.
keywords: pairing-based cryptosystem, η pairing, genus 2 hyperelliptic, characteristic 2, dedicated processor

1 Introduction

Cryptosystems based on bilinear pairings are becoming increasingly popular [6]. Pairings were originally introduced to cryptography as attacks capable of reducing the Discrete Logarithm Problem (DLP) on some elliptic curves to the DLP on finite fields [18, 8]. However, successful cryptographic applications based on these pairings have followed, including a one-round tripartite Diffie-Helman (DH) protocol [12], Identity Based Encryption (IBE) schemes [3], Identity Based Identification schemes [15], signature schemes [22, 4] and key exchange schemes [25].

It follows that efficient implementation of bilinear pairings is vital for the practical application of these schemes. Pairings are used to map the DLP from the divisor class group of a curve $C(\mathbb{F}_q)$ to the multiplicative subgroup $\mathbb{F}_{q^k}^*$ of some extension of the basefield. The value k is known as the *embedding degree* or *security multiplier*. For security, k should be sufficiently large without compromising effective implementation. To date, the implementation aspects of the Tate pairing have attracted the most attention in the literature since it can be computed efficiently [19, 9, 1]. The key to efficient Tate pairing calculation over supersingular curves lies in the shrewd selection of points, yielded by a particular mapping (of the type discussed in [27]) from $C(\mathbb{F}_q)$ to $C(\mathbb{F}_{q^k})$. The so called *tower of extensions* idea (see [11, 21] for examples of successful usage) for performing operations in $\mathbb{F}_{q^k}$ in terms of operations in $\mathbb{F}_q$ is also employed.

The Tate pairing computation was improved and generalised to the case of hyperelliptic curves in [7], resulting in the Duursma Lee (DL) algorithm. Further optimisations of this algorithm appeared in [16, 24, 10]. In [2] it was clarified how the DL methods can be employed in a more general sense. They call their clarification the η *pairing*. A fast octupling on genus 2 hyperelliptic curves was also observed and this was incorporated into the genus 2 pairing computation. This results in a very efficient pairing computation on genus 2 hyperelliptic curves. This

is beneficial for two reasons. Firstly, genus 2 hyperelliptic curves offer a maximum security multiplier of 12, while elliptic curves offer a maximum of 6. This offers clear implications for a smaller field size in the genus 2 case for the same security. Secondly, the maximum security multiplier is achieved in characteristic 2 in the genus 2 case, whilst, in contrast, it is achieved in characteristic 3 in the elliptic case. From an implementation standpoint, the genus 2 case is preferable since field operations are generally easier to perform in characteristic 2. However, it must be noted that there is a trade-off between the reduced field size and the added complexity of computing the pairing in genus 2. Further work on speeding up the DL methods has appeared recently in [2], resulting in the η_T pairing.

In [14], observations were made on the parallelization of a characteristic 3 Modified Duursma Lee (MDL) Algorithm, based on optimisations presented in [16]. It was suggested that a significant reduction in computation time could be achieved through parallelization of operations on an embedded system. The η pairing is also an excellent candidate for implementation on a dedicated system. This paper describes how parallelization of the η pairing can be achieved in the characteristic 2 genus 2 case. An η pairing processor architecture is also described. Results returned by the architecture when prototyped on a Field Programmable Gate Array (FPGA) are also provided and discussed. The architectures are easily ported to an ASIC implementation. For a real-world system, it would be interesting to know how fast and costly an ASIC version would be for use in a server. For use in a hand-held device where small hardware is desirable, it would be interesting to know how small the hardware could be made if we are willing to tolerate having the computation time rise by some factor.

This paper is organised as follows: Section 2 briefly outlines some related work in the area. Section 3 provides the necessary mathematical preliminaries. Section 4 describes how the η pairing is computed in practice. Section 5 develops architectures required for implementing the pairing on a dedicated processor. Section 6 presents the results returned by the processor and compares them with results of other pairing implementations. Finally, Section 7 draws some conclusions from the work.

2 Related Work

Given the effort invested in algorithmically optimising the Tate pairing, and the well known security and speed advantages of dedicated cryptographic hardware [23], it is somewhat surprising to find that the implementation of these pairings on dedicated systems has received little attention in the literature to date. An architecture for a characteristic 3 implementation of the Tate pairing, based on the MDL algorithm, is proposed in [14]. Here, the suitability of MDL for parallelization on a dedicated hardware processor is discussed. However, only an estimate of the number of clock cycles an MDL Tate pairing would return on the proposed architecture is presented. To the authors' knowledge, no other dedicated implementation of a pairing based on genus 2 hyperelliptic curves has appeared in the literature to date. Furthermore, it appears that this paper presents the first hardware implementation of a bilinear pairing in characteristic 2.

3 Mathematical Preliminaries

Let $\mathbb{F}_q, q = p^m$ be a finite field, where p is known as the characteristic of the field. A hyperelliptic curve C of genus g over $\mathbb{F}_q$ is defined by an equation of the form $C : v^2 + h(u)v = f(u)$ where $f(u) \in \mathbb{F}_q[u]$ is a monic polynomial with degree

$2g+1$, and the degree of $h(u) \leq g$. A divisor is a finite formal sum of points on C, i.e. $D = \sum_{P_i} m_i P_i$ where $m_i > 0$. The degree of a divisor is the sum $\sum_{P_i} m_i$.

A group law is defined on C via the degree zero divisor class group of the curve. The divisor class group is isomorphic to the *Jacobian Variety* of C. A subgroup G of the divisor class is said to have *embedding degree* k if the subgroup order l divides $q^k - 1$, but does not divide $q^i - 1, \forall\ 0 < i < k$. A subgroup of order l is known as an l-torsion group.

Let $G = J_C(\mathbb{F}_{q^k})$ be the Jacobian Variety of C over $\mathbb{F}_{q^k}$, and G[l] be the l-torsion group and G/lG be the quotient group. The Tate pairing is a mapping:

$$\langle \cdot, \cdot \rangle : G[l] \times G/lG \rightarrow \mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^l \tag{1}$$

The Tate pairing is well-defined, non-degenerate and bilinear. A restricted version of the pairing, useful for cryptographic applications, can be used which retains all of these properties. Let $G_1 = J_C(\mathbb{F}_q)$, $G_2 = J_C(\mathbb{F}_{q^k})$, and let $G_1[l]$ and $G_2[l]$ be the respective l-torsion groups. The Tate pairing is now a mapping:

$$\langle \cdot, \cdot \rangle : G_1[l] \times G_2[l] \rightarrow \mathbb{F}^*_{q^k} \tag{2}$$

The Tate pairing is found as follows. Given points $P \in C[l](\mathbb{F}_q)$ and $Q \in C[l](\mathbb{F}_{q^k})$ it is clear that $lP = 0$ so that there is a rational function f_P such that the divisor of f_P, denoted (f_P) is equal to $l(P) - l(\infty)$. Now compute a divisor D which is equivalent to $(Q) - (\infty)$ such that the support of D is disjoint from the support of f_P. The value of the Tate pairing is then:

$$\langle P, Q \rangle_l = f_P(D) \tag{3}$$

where $f_P(D)$ can be evaluated as $\prod_i f_P(P_i)^{n_i}$ given $D = \sum n_i P_i$

The Tate pairing is only defined up to a multiple of an lth power. To obtain a unique value, a reduced pairing $e(\cdot, \cdot)$ must be defined:

$$e(P, Q) = \langle P, Q \rangle_l^{(q^k-1)/l} \tag{4}$$

The value of the security multiplier k must be large enough to provide an appropiate level of security for the DLP but as k increases so too does the computational complexity of the pairing. This is a trade-off that a randomly selected curve is unlikely to satisfy. The only classes of curves that are known to have appropiate embedding degrees are supersingular curves ($k \leq 6$ in elliptic case and $k \leq 12$ in genus 2 hyperelliptic case) and MNT curves.

Much work has been performed in optimising the pairing computation on supersingular curves [1, 9] and so from now on we will restrict ourselves to discussion of these curves. One of the most important properties of such curves is the existence of a *distortion map* ψ [27]. A distortion map is a non-trivial endomorphism which can be used to map an l-torsion point with coordinates in a base field to an l-torsion point with coordinates in an extension field. The Tate pairing can then be computed as $e(P, \psi(Q))$, where coordinates of both P and Q are elements of the base-field. If the distortion map is chosen carefully, then divisions in Miller's Algorithm [19] used to compute the pairing can be eliminated [1].

4 The η Pairing

Using the DL optimisations established in [7], in [2] the η pairing is defined, which is a generalisation of the DL algorithm for computing the Tate pairing. Let P and Q be points on a hyperelliptic curve C of genus g over $\mathbb{F}_{q^k}, q = p^m$. Assume that a

suitable distortion map ψ is available which gives rise to denominator elimination. Given an integer p and a point P, there exists a Miller function $f_{p,P}$ such that $(f_{p,P}) = p(P) - ([p]P) - (p-1)(\infty)$. Let D be the divisor $D = (P) - (\infty)$ having order dividing $q^k + 1$. The η pairing can then be defined as:

$$\eta(P,Q) = f_{q,P}(\psi(Q)) = \prod_{i=0}^{m-1} [f_{p,[p^i]P}(\psi(Q))]^{p^{m-1-i}} \tag{5}$$

This pairing is bilinear, and is related to the Tate pairing with:

$$\langle P, \psi(Q) \rangle_{q^k+1} = \eta(P,Q)^{kq^{k-1}} \tag{6}$$

As mentioned previously, the genus 2 hyperelliptic case is a good candidate for a dedicated implementation due to its high embedding degree ($k = 12$) in characteristic 2, which yields smaller hardware components for the same security. This section outlines how the η pairing can be computed in this case. We consider the curve $C : y^2 + y = x^5 + x^3 + d$, with $d = 0$ or 1 over $\mathbb{F}_{2^m}$, where m is coprime to 6.

4.1 The distortion map $\psi(Q)$

A distortion map ψ is required to map elements from $C(\mathbb{F}_{2^m})$ to $C(\mathbb{F}_{2^{12m}})$. Using the tower of extensions idea, elements of the field $\mathbb{F}_{2^{12m}}$ are represented as 12-tuples, i.e. $\mathbb{F}_{2^{12m}} \rightarrow \mathbb{F}_{((2^m)^6)^2}$. The irreducible polynomial on which the $\mathbb{F}_{2^{6m}}$ subfield is based in this case is $f(x) = x^6 + x^5 + x^3 + x^2 + 1$. If $w \in \mathbb{F}_{2^{6m}}$ is a root of this polynomial, then, using finite field arithmetic, $w^8 = w + 1$. Furthermore, define $s_1 = w^2 + w^4$, $s_2 = w^4 + 1$ and $s_0 \in \mathbb{F}_{2^{12}}$ as a solution of ${s_0}^2 + s_0 = w^5 + w^3$. Elements of $\mathbb{F}_{2^{12m}}$ are then represented with respect to the basis:

$$\{1, w, w^2, w^3, w^4, w^5, s_0, ws_0, w^2s_0, w^3s_0, w^4s_0, w^5s_0\}$$

The distortion map is then [2]:

$$\psi(x,y) = (x + w, y + s_2x^2 + s_1x + s_0) \tag{7}$$

4.2 The function $f_{p,P}$

A Miller function $f_{p,P}$ is required such that $(f_{p,P}) = p(P) - ([p]P) - (p-1)(\infty)$. A divisor of the form $D = (P) - \infty$ is known as *degenerate*. In general, multiplication of D by an integer p will not return another degenerate divisor. However, on supersingular curves, multiplication by p may have a special form which always returns degenerate divisors. It was observed in [2] that an octupling formula exists on the curve in the genus 2 case such that $8D = (P') - \infty$, where $P' = \phi\pi^6(P)$, π is the 2-power Frobenius map and $\phi(x,y) = (x + 1, y + x^2 + 1)$. It is seen that octupling requires only squaring and addition on $\mathbb{F}_{2^m}$, both of which are trivial in hardware (as will become evident later).

To take advantage of this Miller's Algorithm is defined to the base 8, requiring the computation of $(f_{8,P}) = 8(P) - ([8]P) - (7)(\infty)$. Given $P = (x_P, y_P)$ and $\psi(Q) = (x,y)$, this function is defined in [2] as:

$$f_{8,P}(\psi(Q)) = (y + b_4(x))^2(y + b''_8(x)) \tag{8}$$

where

$$b_4(x) = x^3 + (x_P^8 + x_P^4)x^2 + (x_P^4)x + y_P^4 \tag{9}$$

and

$$b''_8(x) = (x_P^{32} + 1)x^2 + (x_P^{32} + x_P^{16})x + (y_P^{16} + x_P^{16} + x_P^{48} + 1) \tag{10}$$

In [2] it is shown how the distortion map may be quite easily incorporated into $f_{8,P}$. Following the notation in [2], define $\alpha = (y + b_4(x))^2 \circ \psi$ and $\beta = (y + b''_8(x)) \circ \psi$. Then let $f_{8,P} = \alpha\beta$ where $(\alpha, \beta) \in \mathbb{F}_{2^{6m}}$ and the distortion map has been incorporated into the function $f_{8,P}$.

4.3 Optimising the η pairing

To incorporate the fast octupling operation, we now need to compute:

$$\eta(P,Q) = \prod_{i=0}^{m-1} [f_{8,[8^i]P}(\psi(Q))]^{8^{m-1-i}} \tag{11}$$

In [2], the authors suggest the following optimisations for this calculation:

Precomputation: On each iteration of the loop, the point P is updated with the function $f_{8,P}$. This function requires squarings of the current coordinates of P on a particular loop iteration. In [2], the authors suggest a methodology for re-defining these operations in terms of the co-ordinates (x_P, y_P) of the original pairing input P. To perform this optimisation, all values $(x_1[i], x_2[i]) = ({x_P}^{2^i}, {y_P}^{2^i}), \forall\, i \;\; 0 < i < m$ are precomputed before loop iteration. The α and β terms are then re-defined in terms of powers of squares of the original input coordinates.

Absorbing powers of 8: The calculation of (11) requires that the $\mathbb{F}_{2^{12m}}$ component $f_{8,2^{3i}P}(\psi(Q))$ be raised to $2^{3(m-1-i)}$ on each iteration. In [2], the authors devise a method of absorbing this exponent into the calculation of f. To achieve this, α and β must be mapped to $\alpha^{2^{3(m-1-i)}}$ and $\beta^{2^{3(m-1-i)}}$ respectively. This replaces the exponentiation on $\mathbb{F}_{2^{12m}}$ with squarings of the input coordinates on $\mathbb{F}_{2^m}$. Note that, to do this, all powers of squares of coordinates of the point Q up to *m-1* must now also be found.

Unroll $\alpha\beta$ and Perform Karatsuba Multiplication The $\mathbb{F}_{2^{6m}}$ elements α and β must be multiplied together to find f. The α and β terms can be unrolled using the basis defined in Section 4.1. Karatsuba multiplication [13] can then be employed on the $\mathbb{F}_{2^m}$ elements of α and β, replacing some $\mathbb{F}_{2^m}$ multiplications with $\mathbb{F}_{2^m}$ additions. These additions are trivial in hardware and so the trade-off is worthwhile.

Following the treatment in this section, we compute the η pairing according to Algorithm 1. Note that the squarings of the input elements are stored in arrays so that, for example, $x_1[k_4]$ references the x_1 array of squares using the array index $k_4 = 3i$.

4.4 Mapping to the reduced Tate pairing

The η pairing can be mapped to the reduced Tate pairing as described in Section 3. From [2], in this case the Tate pairing is related to the η pairing with:

$$\langle P, \psi(Q)\rangle_{2^{6m}+1} = \eta(P,Q)^{2q} \tag{12}$$

From (4), we require the reduced pairing $e(P,Q) = \langle P,Q\rangle_l^{(q^k-1)/l}$. Given $l = 2^{6m+1}$, $q = 2^m$ and $k = 12$, it is easily shown that:

$$e(P,Q) = \langle P,Q\rangle_{2^{6m+1}}^{(q^6-1)} \tag{13}$$

The exponentiations to q are computed using the efficient methods described in [1]. A modified version of the Extended Euclidean Algorithm (EEA) for inversions on $\mathbb{F}_{p^m}$, proposed in [17], was chosen to implement the required inversion. This method reduces the number of subfield inversions required to just 1 $\mathbb{F}_{2^m}$ inversion, at the expense of extra multiplications, which will be optimised in the processor.

5 An Architecture for the η Pairing

The suitability of a dedicated core for computation of the η pairing was investigated since an embedded system implementation can offer a widescale parallelization of operations that is generally unavailable in software. In particular, an

Algorithm 1 The η pairing

INPUT: P=(x_P, y_P),Q=$(x_Q, y_Q) \in J_C(\mathbb{F}_{2^m})$
OUTPUT: $\eta(P, Q) \in \mathbb{F}_{2^{12m}}$

Initialisation: set $\gamma = 1$ ***if*** $m \equiv 1$ ***mod 4, otherwise*** $\gamma = 0$
Precomputation Stage: Find powers of squares of P and Q and store in arrays
$x_1[i] \leftarrow x_P^{2^i}$, $y_1[i] \leftarrow y_P^{2^i}$, $x_2[i] \leftarrow x_Q^{2^i}$, $y_2[i] \leftarrow x_Q^{2^i}, \forall\, 0 < i < m-1$ (* F_{2^m} sq *)
$f \leftarrow 1$
for i = 0 to m - 1 do
    All k_* ***in the next 2 lines to be considered modulo m***
    $k_1 \leftarrow (3m - 3 - 3i), k_2 \leftarrow (k_1 + 1), k_3 \leftarrow (k_2 + 1)$
    $k_4 \leftarrow (3i), k_5 \leftarrow (k_4 + 1), k_6 \leftarrow (k_5 + 1)$
    Calculate $\alpha \leftarrow a + bw + cw^2 + dw^4 + s_0$
    $d \leftarrow x_1[k_4] + x_1[k_5]$
    $a \leftarrow y_2[k_2] + (x_1[k_4] + 1 + x_2[k_3]).x_2[k_2] + d.x_2[k_3] + y_1[k_4] + \gamma$ (* F_{2^m} mul *)
    $b \leftarrow x_2[k_3] + x_2[k_2]$
    $c \leftarrow x_2[k_3] + x_1[k_4] + 1$
    Calculate $\beta \leftarrow e + f_2 w + g w^2 + h w^4 + s_0$
    $f_2 \leftarrow x_1[k_5] + x_1[k_6]$
    $e \leftarrow y_2[k_1] + x_1[k_6].x_2[k_2] + f_2.x_2[k_1] + y_1[k_5] + (x_1[k_6] + 1).x_1[k_5] + \gamma$ (* F_{2^m} mul *)
    $g \leftarrow x_2[k_1] + x_1[k_6] + 1$
    $h \leftarrow x_2[k_2] + x_2[k_1]$
    Unroll $\alpha\beta$ ***multiplication using karatsuba method,*** $u_1 = \alpha\beta$
    $dh \leftarrow d.h, dg \leftarrow d.g, ch \leftarrow c.h, cg \leftarrow c.g, ae \leftarrow a.e, bf_2 \leftarrow b.f_2$ (* F_{2^m} mul *)
    $t_0 \leftarrow ae + ch + dg + dh$ (* F_{2^m} mul *)
    $t_1 \leftarrow (a + b)(f_2 + e) + ae + bf_2 + dh$ (* F_{2^m} mul *)
    $t_2 \leftarrow (a + c)(g + e) + ae + cg + bf_2 + ch + dg$ (* F_{2^m} mul *)
    $t_3 \leftarrow (b + c)(g + f_2) + bf_2 + cg + ch + dg + 1$ (* F_{2^m} mul *)
    $t_4 \leftarrow (a + d)(h + e) + ae + dh + cg$ (* F_{2^m} mul *)
    $t_5 \leftarrow (b + d)(h + f_2) + bf_2 + dh + ch + dg + 1$ (* F_{2^m} mul *)
    $u_1 \leftarrow (t_0, t_1, t_2, t_3, t_4, t_5, a + e + 1, b + f_2, c + g, 0, d + h, 0)$
    $f \leftarrow f.u1$ (* $F_{2^{12m}}$ mul *)
end for
return $f = \eta(P, Q)$

external accelerator can greatly reduce computation time through parallelization of the most time consuming operations. This section details the creation of this architecture. The architecture was created with the primary design goal being the optimization of the number of multiplication cycles required within the loop as these are the most time consuming operations.

In general terms, creation of the system required the design of:-

1. A Precomputation Block (PB) to compute powers of the squares of the input points before loop iteration.
2. An Arithmetic Unit (AU), which is composed of a General Arithmetic Unit (GAU) for general computations on $\mathbb{F}_{2^m}$ and a Multiplicative Arithmetic Unit (MAU) that optimises multiplications required by the η pairing.
3. A Control Unit (CU) to sequence the operations required by the pairing.
4. A Memory Bank (MB) to store intermediate values and a multiplexor and encoder bank controlled by the CU to handle data transfer to and from the AU.

5.1 The Precomputation Block (PB)

As seen in Algorithm 1, precomputation of $(x_1[i], y_1[i]) = ({x_P}^{2^i}, {y_P}^{2^i})$ and $(x_2[i], y_2[i]) = ({x_Q}^{2^i}, {y_Q}^{2^i})$ is required for all i, $0 < i < m$.

Calculation of the $x_j[i], j = 1, 2$ values is achieved using modules of the type **DUAL_PORT_SQUARE_BLOCK** seen in Figure 1(b). Each module contains $m \times m$ bit block RAM. Squaring on $\mathbb{F}_{2^m}$ is performed in one clock cycle using the parallel squaring architecture presented in [20]. **SQUARE_BLOCK_CONTROL** iterates i from 0 to $m-1$ and controls the writing of $(x_j[i], y_j[i])$ to RAM.

Once this precomputation step is complete the loop can begin. On every iteration, values of $(x_j[i], y_j[i]), j = 1, 2$ must be read according to the array indices defined in Algorithm 1. These indices must be calculated *modulo m*. Modular reduction, however, can be quite an expensive operation in hardware. Fortunately, the array indices depend only on *i* and *m*, and are constant over the loop for a certain field size. This facilitates the generation of all indices by a software program, written in C, which outputs all required indices according to field size. These indices are stored as arrays of constants on the embedded system and can be read throughout the loop according to the current value of *i*.

As seen in Algorithm 1, each iteration requires the reading of 2 $x_1[i]$ values, 2 $x_2[i]$ values and one each of $y_1[i]$ and $y_2[i]$. For this reason, the $x_j[i]$ values are stored in dual port block RAM while the $y_j[i]$ values are allocated only single port RAM. The **DUAL_PORT_SQUARE_BLOCK** modules each have a *read_mode* control input. This is supplied by the general control unit and decides whether the α or β array values are read at any particular time. The full precomputation block, **PRECOMP_BLOCK**, is illustrated as Figure 1(a). Note that the elements d,b,c and, on the second read, f_2,g,h (see Algorithm 1) are calculated directly at the outputs of the square block modules. This saves valuable clock cycles on every loop iteration at the expense of very little hardware.

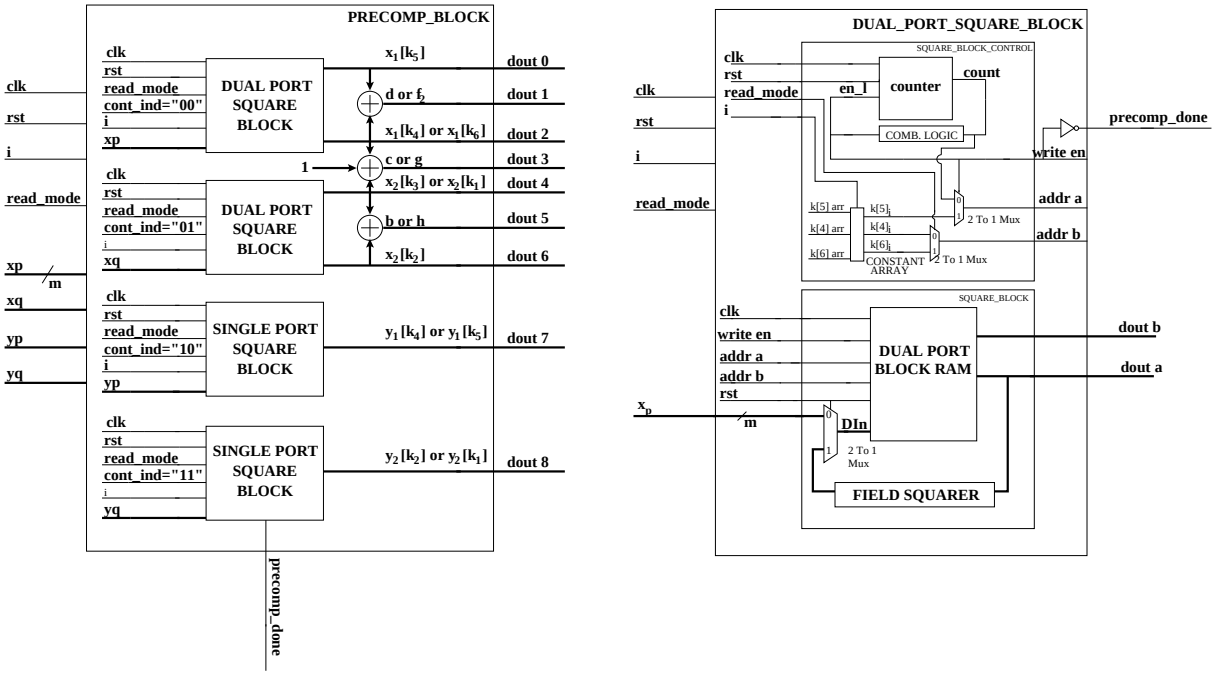

Fig. 1. (a) The Precomputation Block, and (b) the Dual Port Square Block module for x_P.

5.2 The Arithmetic Unit (AU)

The AU consists of an optimised Multiplication Arithmetic Unit (MAU) and a General Arithmetic Unit (GAU), which is used for all non-multiplicative operations on

$\mathbb{F}_{2^m}$. This separation allows for $\mathbb{F}_{2^m}$ operations to be performed in the GAU while waiting for a multiplication to finish.

The General Arithmetic Unit (GAU) The GAU consists of modules for performing the following field operations:

FIELD ADDITION: This can be performed in 1 clock cycle using m-bit xor gates. The GAU contains a number of m-bit XOR gates since addition is performed regularly and a parallelization of this operation reduces intermediate reads and writes from and to memory respectively.

FIELD SQUARING: This operation is performed less regularly than addition and so only 1 squaring module is included in the GAU. Squaring on $\mathbb{F}_{2^m}$ is performed using only combinatorial logic and can be completed in 1 clock cycle [20].

FIELD INVERSION: Inversion on any finite field is a relatively costly operation. Fortunately, $\mathbb{F}_{2^m}$ inversion only needs to be performed once (this is in the mapping to the Tate pairing) and so only one module is included in the GAU. Inversion is performed using a modified version of the Extended Euclidean Algorithm (EEA) discussed in [5]. This algorithm was chosen since it provides a low calculation time of $2m$ clock cycles and employs relatively simple underlying field operations that can be implemented efficiently in hardware.

The Multiplication Arithmetic Unit (MAU) During loop iteration, 2 distinct sets of multiplications are necessary: the unrolled $\mathbb{F}_{2^m}$ multiplications required to compute $\alpha\beta$, and those to multiply the cumulative function f by $\alpha\beta$ on $\mathbb{F}_{2^{12m}}$ at the end of every iteration. The $\mathbb{F}_{2^{12m}}$ multiplication is performed using the tower of extensions idea in conjunction with Karatsuba multiplication in a similar fashion to that used for the elliptic case in [9]. The $\mathbb{F}_{2^{12m}}$ multiplication is first reduced to 3 $\mathbb{F}_{2^{6m}}$ multiplications. Each of the $\mathbb{F}_{2^{6m}}$ multiplications is further reduced to 3 degree 2 polynomial Karatsuba multiplications. Thus, in total, we decompose an $\mathbb{F}_{2^{12m}}$ multiplication into 9 degree 2 Karatsuba multiplications. A degree 2 Karatsuba multiplication is computed as illustrated in Algorithm 2. The parallel implementation of the $\mathbb{F}_{2^{12m}}$ multiplications form a cornerstone of the η pairing processor architecture.

Algorithm 2 Degree 2 Karatsuba multiplication on $\mathbb{F}_{2^m}$

INPUT: Two elements $A = (a_0, a_1, a_2)$ and $B = (b_0, b_1, b_2)$, all $a_i, b_i \in \mathbb{F}_{2^m}$
OUTPUT: The element $C = (c_0, c_1, c_2, c_3, c_4)$ such that C=A.B using Kar. Mul.

1: $c_0 \leftarrow a_0.b_0$ (* $\mathbb{F}_{2^m} mul$ *)
2: $c_2 \leftarrow a_1.b_1$ (* $\mathbb{F}_{2^m} mul$ *)
3: $c_4 \leftarrow a_2.b_2$ (* $\mathbb{F}_{2^m} mul$ *)
4: $c_1 \leftarrow (a_0 + a_1).(b_0 + b_1) + c_2 + c_0$ (* $\mathbb{F}_{2^m} mul$ *)
5: $c_3 \leftarrow (a_1 + a_2).(b_1 + b_2) + c_2 + c_4$ (* $\mathbb{F}_{2^m} mul$ *)
6: $c_2 \leftarrow c_2 + (a_0 + a_2).(b_0 + b_2) + c_0 + c_4$ (* $\mathbb{F}_{2^m} mul$ *)
7: **Return** $C = (c_0, c_1, c_2, c_3, c_4)$

A dual mode Multiplication Arithmetic Unit (MAU) was created to accomodate an optimized general multiplication for $\alpha\beta$ and an optimization of the degree 2 Karatsuba multiplications. This MAU is illustrated as Figure 2.

The **FIELD_DIGIT_BLOCK** in Figure 2 contains digit multipliers of the type introduced in [26]. These multipliers allow D coefficients of the inputs to be operated on in parallel, where D is known as the *Digit Size*. Multiplication is performed

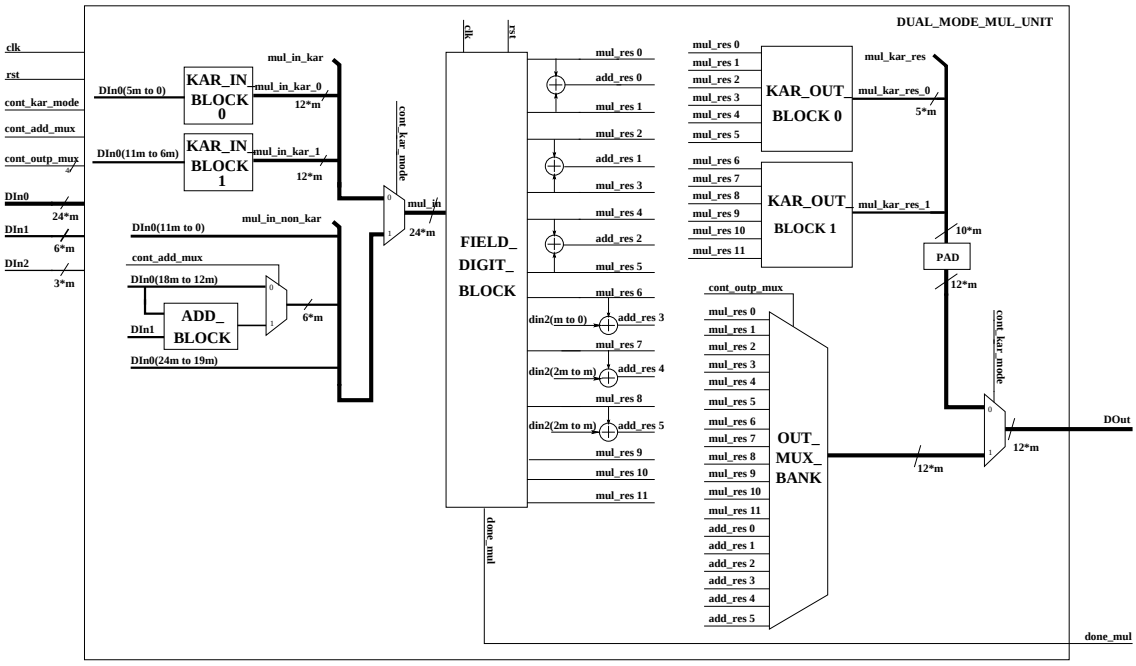

Fig. 2. The Dual Mode Multiplication Unit

in m/D clock cycles. As D increases, so too does the area of the multiplier. Thus, there is a trade-off between speed and area.

In total, 12 of these multipliers operate in parallel. This quantity offers a significant parallelization of operations while minimizing the time that any particular multiplier may lie idle. It was found that the optimum number of multipliers for parallelization of the $\alpha.\beta$ operations approached 12. It also makes sense for the number of multipliers to be a multiple of 6. Each degree 2 Karatsuba multiplication requires 6 $\mathbb{F}_{2^m}$ multiplications. Thus 2 of these operations can be performed in parallel while utilizing all available multipliers.

The multiplier mode is determined by the Control Unit. In *General Purpose Mode*, 12 field multiplications can be performed in parallel. Conversely, in *Karatsuba Mode* two parallel Karatsuba multiplications are performed. The modes operate as follows:

GENERAL PURPOSE MODE: This mode is optimized for the $\alpha.\beta$ multiplication. In this case, the *mul_in_non_kar* bus is selected as input to the digit multiplier block. Some of the *DIn0* inputs can be dynamically added to the *DIn1* inputs using the *cont_add_mux* control bit. Note also that the results of multiplications can be immediately added to both the results of other multiplications and another set of input bits. The ability to perform these immediate additions was found to greatly reduce the computation time throughout the loop by eliminating many intermediate additions and read/writes between multiplication cycles. The correct outputs are selected from the output of the **OUT_MUX_BANK** module using the *cont_outp_mux* control bits.

KARATSUBA MODE: Here, all additions required for degree 2 Karatsuba multiplication according to Algorithm 2 are performed within the multiplication module. The inputs to the degree 2 multiplication are sent to the **KAR_IN_BLOCK** modules for addition and re-organisation. The outputs of these blocks are sent to the digit multiplier block using the *cont_kar_mode* control bit. Once the multiplications have completed, the results enter the **KAR_OUT_BLOCK** modules for

further additions. These blocks mean that the results of 2 degree 2 Karatsuba multiplications can be collected at the output of the multiplication block $m/D+2$ clock cycles after Karatsuba multiplication is initialised by the Control Unit, thereby eliminating any intermediate additions,reads and writes that would have been required. The structures of the **KAR_IN_BLOCK** and **KAR_OUT_BLOCK** modules are shown in Figure 3.

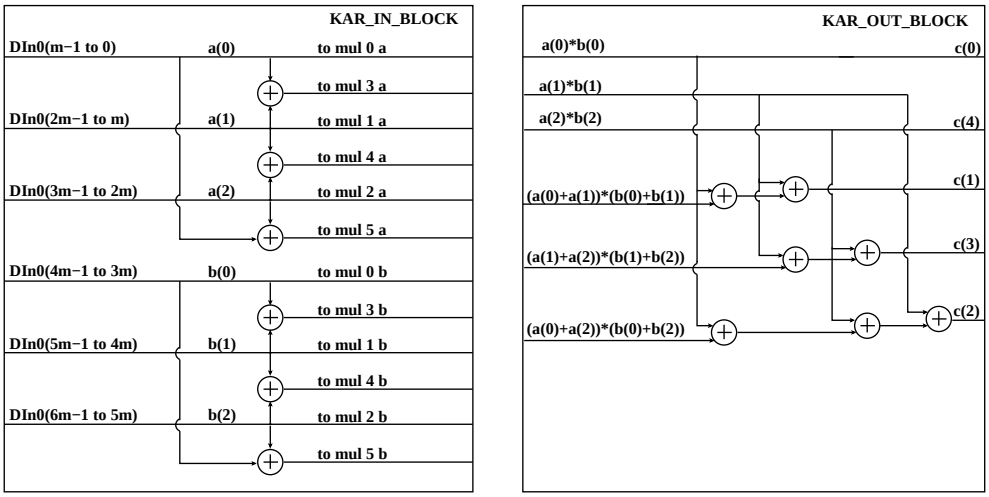

Fig. 3. (a) The **KAR_IN_BLOCK** and (b) the **KAR_OUT_BLOCK**

5.3 The η Processor

The complete η processor **ETA_PROC** is shown in Figure 4.

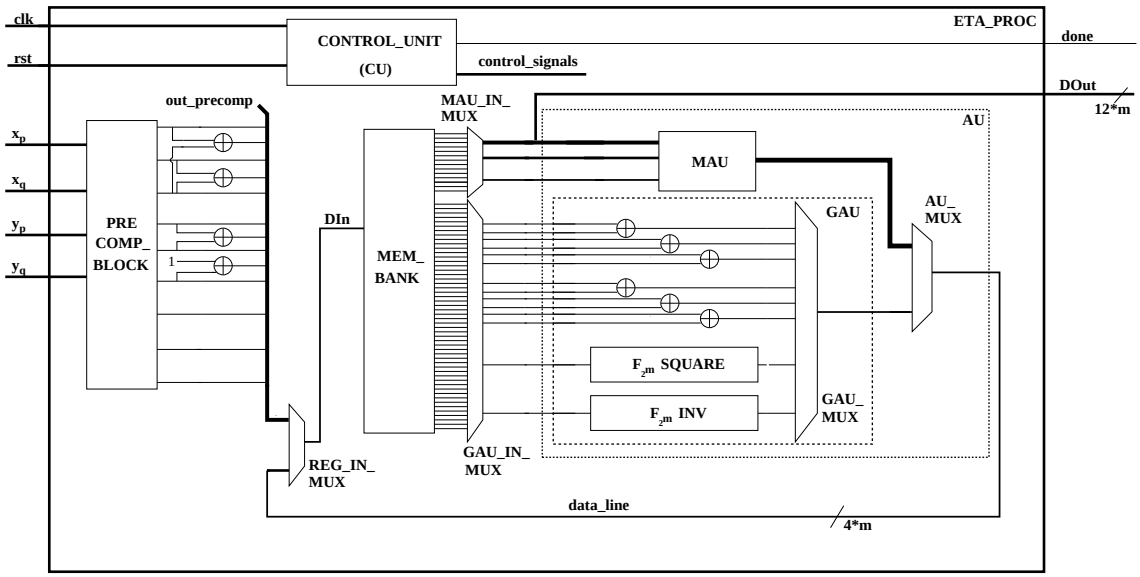

Fig. 4. The *eta* processor **ETA_PROC**

All operations required to compute the η pairing and its subsequent conversion to the reduced Tate pairing are performed in the **AU**. The **MAU** and the **GAU** receive their inputs from different multiplexor banks so that both can operate in parallel. The **AU_MUX** determines which set of results to put on the *data_line* bus.

Note that further additions are performed at the output of the **PRECOMP_BLOCK**. These additions are performed to ensure that the first multiplication of a new loop can start as soon as possible. The **REG_IN_MUX** module determines whether values from the **PRECOMP_BLOCK** or the **AU** are sent to the register bank **REG_BANK**.

The **CONTROL_UNIT** (CU) module schedules all operations required to perform the η pairing by generating the required control lines in the circuit. The CU is a finite state machine, designed through analysis of available intermediate variables throughout the computation. The state machine is designed with speed in mind such that, if intermediate operations can be performed while a multiplication is taken place, they will be scheduled to do so. The CU also controls encoders in the multiplexor banks of the Memory Bank (MB). These control signals are generated by C-code according to high-level user input. This means that the processor is readily reconfigured should subsequent changes be required.

6 Results and Comparisons

The architectures described in the previous section were captured in the VHDL hardware design language and prototyped on the Xilinx Virtex2Pro125 family of FPGA's [28]. These devices contain 55616 slices (each slice containing a number of 4-input look-up tables, storage elements and dedicated carry logic for high-speed arithmetic). FPGA prototyping was chosen due to the ease with which these devices can be reconfigured. The architectures described, however, are also suitable for ASIC implementation.

The processor was created for a pairing computation based on the hyperelliptic curve $C_d : y^2 + y = x^5 + x^3 + d$. A base-field $\mathbb{F}_{2^{103}}$ was chosen as this allows a fair comparison with the optimised software results listed for $\mathbb{F}_{2^{103}}$ in [2]. Note that $d = 0$ in this case for a secure group order of $13 \times (12 \times 103 + 1) = 16081$ [2].

To investigate different speed/area ratios of the system **ETA_PROC** was prototyped for this field size using 4 different multiplier digit sizes D_m. The results are presented in Table 1. The frequency values listed are for a Post Place and Route (PPR) clock frequency whilst the $\langle P, Q \rangle$ timing results are for a full pairing computation (including final exponentiations) at this frequency.

Table 1. Prototype **ETA_PROC** implementation results on the Xilinx Virtex2Pro125 device

D_m	FPGA slices	% device	PPR (MHz)	$\langle P, Q \rangle$ (μs)
8	40128	72	33.8	874
16	**43986**	**79**	**32.3**	**749**
24	47239	85	28.9	776
32	50893	92	28.1	767

Note that increasing the value of D_m beyond 16 actually serves to slow down the pairing calculation. This is due to the decrease in PPR frequency. This frequency reduction is due in part to routing delays through the processor as it becomes larger. It must be noted that slightly higher frequencies for the cases $D_m = 24$ and $D_m = 32$ may be achieved by utilising a more rigorous place and route process. However, for values of D_m greater than 16 (with a multiplication cycle requiring m/D_m clock cycles), the time taken for the completion of intermediate non-multiplicative operations is not insignificant when compared to the multiplication

cycle time and, in the authors' view, the relatively small increase in speed that would be returned by the $D_m = 24$ and $D_m = 32$ cases may not justify the additional area requirement.

To date the fastest software pairing computation time that has been reported in the literature (for similar security) is 1.87 ms. This is for the genus 2 characteristic 2 hyperelliptic implementation of the η_T pairing on $\mathbb{F}_{2^{103}}$ presented in [2]. The η pairing is computed in 3 ms for the same field size. These results are for an optimized C-code software implementation on a Pentium IV processor running at 3 GHz.

It can be seen that the results of the hardware architecture compare favourably to these implementations, offering an improvement over the software computation times presented in [2], showing that a significant speedup can be achieved through use of dedicated hardware for pairing-based applications.

7 Conclusions

This paper has described a dedicated processor for the η pairing and its subseqeuent translation to the Tate pairing. In the genus 2 hyperelliptic case, the η pairing offers a very efficient implementation of the Tate pairing in characteristic 2. To the authors' knowledge, the processor **ETA_PROC** is the first dedicated architecture created for a pairing in characteristic 2 and for a pairing performed on a genus 2 hyperelliptic curve. Different speed/area ratios can be returned using different multiplier digit sizes. Results show that a significant speed-up in the pairing computation can be achieved if a dedicated parallelized cryptographic processor is utilized instead of a general purpose serial one. Although this paper presents results for the field $\mathbb{F}_{2^{103}}$, the device is fully reconfigurable for other suitable field sizes.

References

1. P. S. L. M. Barreto, H.Y. Kim, B. Lynn, and M. Scott. Efficient implementation of pairing based cryptosystems. In *Advances in Cryptology - CRYPTO 2002*, volume 2442 of LNCS, pages 354–368. Springer, 2002.
2. P.S.L.M. Barreto, S. Galbraith, C. OhEigeartaigh, and M. Scott. Efficient Pairing Computation on Supersingular Abelian Varieties. Report 375/2004, Cryptology ePrint Archive, `http://eprint.iacr.org/2004/375`, 2004.
3. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *Society for Industrial and Applied Mathematics Journal of Computing*, 32(3):586–615, 2003.
4. D. Boneh, I. Mironov, and V. Shoup. A secure signature scheme from bilinear maps. In *Proceedings of RSA-CT'03*, volume 2612 of LNCS, pages 98–110. Springer, 2003.
5. H. Brunner, A. Curiger, and M. Hofstetter. On Computing Multiplicative Inverses in $GF(2^m)$. *IEEE Transactions on Computers*, 42:1010–1015, 1993.
6. R. Dutta, R.Barua, and P. Sarkar. Pairing-Based Cryptography: A Survey. Report 2004/064, Cryptology ePrint Archive, `http://eprint.iacr.org/2004/64`, 2004.
7. 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, pages 111–123. Springer, 2003.
8. G. Frey and G. Ruck. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 206:865–874, 1994.
9. S. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In *Algorithmic Number Theory Symposium - ANTS V*, volume 2369 of LNCS, pages 324–337. Springer, 2002.
10. R. Granger, D. Page, and M. Stam. On small characteristic Algebraic Tori in Pairing-Based Cryptography. Report 2004/132, Cryptology ePrint Archive, `http://eprint.iacr.org/2004/132`, 2004.

11. J. Guajardo and C. Paar. Efficient Algorithms for Elliptic Curve Cryptosystems. In *Advances in Cryptology - CRYPTO 1997*, volume 1294 of LNCS, pages 342–356. Springer, 1997.
12. A. Joux. A one-round protocol for tripartite Diffie-Hellman. In *Proceedings of Algorithmic Number Theory Symposium ANTS-V*, volume 1838 of LNCS, pages 385–394. Springer, 2002.
13. A. Karatsuba and Y. Ofman. Multiplication of Multidigit numbers on Automata. *Sov. Phys. Dokl (english translation)*, 7(7):595–596, 1963.
14. T. Kerins, W.P. Marnane, E.M. Popovici, and P.S.L.M. Barreto. Efficient Hardware for the Tate Pairing Calculation in Characteristic 3. Report 2005/065, Cryptology ePrint Archive, `http://eprint.iacr.org/2005/065`, 2005.
15. K. Kurosawa and S.H. Heng. Identity-Based identification without random oracles. `http://kuro.cis.ibaraki.ac.jp/~kurosawa/2005/ISH05(full.ps)`, 2005. To appear in ISH05.
16. S. Kwon. Efficient Tate pairing computation for supersingular elliptic curves over binary fields. Report 2004/303, Cryptology ePrint Archive, `http://eprint.iacr.org/2004/303`, 2004.
17. C.H. Lim and H.S. Hwang. Fast Implementation of Elliptic Curve Arithmetic in $GF(p^n)$. In *Third International Workshop on Practice and Theory in Public Key Cryptography - PKC 2000*, volume 1751 of LNCS, pages 405–421. Springer, 2000.
18. A. Menezes, T. Okamoto, and S. Vanstone. Reducing elliptic curve logarithms to logarithms on a finite field. *IEEE Transactions on Information Theory*, 39:1639–1646, 1993.
19. V.S. Miller. Short programs for functions on curves. Unpublished Manuscript, `http://crypto.stanford.edu/miller/miller.pdf`, 1986.
20. Gerardo Orlando and Christof Paar. A High Performance Reconfigurable Elliptic Curve Processor for $GF(2^m)$. In *The Second International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 1965 of LNCS, pages 41–56. Springer, 2000.
21. Christof Paar and Pedro Soria-Rodriguez. Fast Arithmetic Architectures for Public-Key Algorithms over Galois Fields $GF((2^n)^m)$. In *Advances in Cryptology - EUROCRYPT 1997*, volume 1233 of LNCS, pages 363–378. Springer, 1997.
22. R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *Proceedings of the Symposium on Cryptography and Information Security*. Okinawa, Japan, 2000.
23. B. Schneier. *Applied Cryptography*. John Wiley & Sons, second edition, 1996.
24. M. Scott and P.S.L.M. Baretto. Compressed Pairings. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of LNCS, pages 140–156. Springer, 2004. Updated Version: Report 2004/032, Cryptology ePrint Archive, `http://eprint.iacr.org/2004/032`.
25. N.P. Smart. An identity based authentication key agreement protocol based on the Weil pairing. *Electronic Letters*, 38:630–632, 2002.
26. 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.
27. E. R. Verhuel. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. In *Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of LNCS, pages 195–210. Springer, 2001.
28. Xilinx. Virtex-2Pro Platform FPGAs : Complete Data Sheet Ds083. `http://www.xilinx.com/bvdocs/publications/ds083.pdf`, 2004.